HL7apy Documentation

Release 1.3.2

CRS4 - Center for Advanced Studies, Research and Development

Nov 22, 2018

Contents

HL7apy is a lightweight Python package to intuitively handle HL7 v2 messages according to HL7 specifications.

The main features includes:

- Message parsing
- Message creation
- Message validation following the HL7 xsd specifications
- Access to elements by name, long name or position
- Support to all simple and complex datatypes
- Encoding chars customization
- Message encoding in ER7 format and compliant with MLLP protocol
- Support to message profile
- Support to Z-Elements
- Simple MLLP server implementation

Currently supported HL7 versions are: 2.2, 2.3, 2.3.1, 2.4, 2.5, 2.5.1, 2.6, 2.7, 2.8, 2.8.1, 2.8.2

Current version is 1.3.2

To get started visit the *Getting started* section

This project is not affiliated with the HL7 organization: the library is just consistent with their specification.

CHAPTER 1

Installation

HL7apy is platform independent and supports Python 2.7 and Python 3.4, 3.5, 3.6, 3.7

To install it get the latest release from GitHub and launch the following command:

```
python setup.py install
```

Alternatively you can use pip to install it from PyPI

```
pip install hl7apy
```

CHAPTER 2

Contents

2.1 Getting started

The following tutorial shows you the main features of the library.

2.1.1 Introduction

HL7apy implements classes for messages, groups, segments, fields, components and subcomponents as defined by the HL7 v2 standard. The elements have a hierarchical relationship and the API gives you the interface for adding, removing and visiting the tree nodes.

2.1.2 Create a message from scratch

You can create a new message by instantiating the *hl7apy.core.Message* class:

```
>>> from hl7apy.core import Message

>>> m = Message("ADT_A01")
>>> m2 = Message()
```

You can both create a message specifying a structure (e.g. ADT_A01) or create a new message with no predefined structure.

Your new message can be populated as follows:

```
>>> pid = Segment("PID")
>>> patient_group = Group("OML_O33_PATIENT")

# add a Segment instance
>>> m.add(pid)
```

(continues on next page)

(continued from previous page)

```
# add a Group instance
>>> m2.add(patient_group)

# create a Segment named MSA and add it to m2
>>> msa = m2.add_segment('MSA')

# create a Group named ADT_A01_INSURANCE and add it to m
>>> g = m.add_group("ADT_A01_INSURANCE")

# assign a Segment instance
>>> m.pid = pid

# assign a string
>>> m.pid = "PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^
↪ANN ARBOR^MI^^USA||555-555-2004~444-333-222|||M"
# equivalent to
>>> m.pid.value = "PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME␣
↪STREET^^ANN ARBOR^MI^^USA||555-555-2004~444-333-222|||M"

# copy from another_message child
>>> m.pid = m2.oml_o33_patient.pid
```

You can also populate your message without explicit creation of its children, as in the following example:

```
>>> from hl7apy.core import Message

>>> m = Message("ADT_A01")
>>> m.pid.pid_5.pid_5_1 = 'EVERYMAN'
>>> m.pid.pid_5.pid_5_2 = 'ADAM'
```

The `PID` segment is created during child traversal, as well as their related fields and components. The previous snippet of code is equivalent to:

```
>>> from hl7apy.core import Message

>>> m = Message("ADT_A01")
>>> pid = Segment("PID")
>>> pid_5 = Field("PID_5")
>>> pid_5.pid_5_1 = 'EVERYMAN'
>>> pid_5.pid_5_2 = 'ADAM'
>>> pid.add(pid_5)
>>> m.add(pid)
```

2.1.3 ADT_A01 example

Suppose you want to create the following ADT_A01 message:

```
MSH|^~\&|GHH_ADT||||20080115153000||ADT^A01^ADT_A01|0123456789|P|2.5||||AL
EVN||20080115153000||AAA|AAA|20080114003000
PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^ANN ARBOR^MI^^
↪USA||555-555-2004~444-333-222|||M
NK1|1|NUCLEAR^NELDA^W|SPO|2222 HOME STREET^^ANN ARBOR^MI^^USA
```

You can create it from scratch by using the core classes, or by using the *hl7apy.parser.parse_message()* function; in the following snippet of code, we show you a way to create it from scratch:

```
>>> from hl7apy.core import Message

>>> m = Message("ADT_A01", version="2.5")
>>> m.msh.msh_3 = 'GHH_ADT'
>>> m.msh.msh_7 = '20080115153000'
>>> m.msh.msh_9 = 'ADT^A01^ADT_A01'
>>> m.msh.msh_10 = "0123456789"
>>> m.msh.msh_11 = "P"
>>> m.msh.msh_16 = "AL"
>>> m.evn.evn_2 = m.msh.msh_7
>>> m.evn.evn_4 = "AAA"
>>> m.evn.evn_5 = m.evn.evn_4
>>> m.evn.evn_6 = '20080114003000'
>>> m.pid = "PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^
↪ANN ARBOR^MI^^USA||555-555-2004~444-333-222|||M"
>>> m.nk1.nk1_1 = '1'
>>> m.nk1.nk1_2 = 'NUCLEAR^NELDA^W'
>>> m.nk1.nk1_3 = 'SPO'
>>> m.nk1.nk1_4 = '2222 HOME STREET^^ANN ARBOR^MI^^USA'
```

2.1.4 Parsing

You can use the provided ER7 parsers to parse a message string:

```
>>> from hl7apy.parser import parse_message

>>> msh = "MSH|^~\&|GHH_ADT||||20080115153000||ADT^A01^ADT_A01|0123456789|P|2.
↪5||||AL\r"
>>> evn = "EVN||20080115153000||AAA|AAA|20080114003000\r"
>>> pid = "PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^ANN␣
↪ARBOR^MI^^USA||555-555-2004~444-333-222|||M\r"
>>> nk1 = "NK1|1|NUCLEAR^NELDA^W|SPO|2222 HOME STREET^^ANN ARBOR^MI^^USA\r"
>>> pv1 = "PV1|1|I|GHH PATIENT WARD|U||||^SENDER^SAM^^MD|^PUMP^PATRICK^
↪P|CAR||||2|A0||||||||||||||||||||||||||2008\r"
>>> in1 = "IN1|1|HCID-GL^GLOBAL|HCID-23432|HC PAYOR, INC.|5555 INSURERS CIRCLE^^ANN␣
↪ARBOR^MI^99999^USA||||||||||||||||||||||||||||||||||||||||||444-33-3333"

>>> s = msh + evn + pid + nk1 + pv1 + in1
>>> message = parse_message(s)
```

By default, *hl7apy.parser.parse_message()* assigns the segments found to the relevant HL7 group. You can disable this behaviour by passing `find_groups=False` to the function. In this case, the segments found are assigned as direct children of the *hl7apy.core.Message* instance.

ER7 parsers for segments, fields and components are also provided:

```
>>> from hl7apy.parser import parse_segment, parse_field, parse_component

>>> pid = "PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^ANN␣
↪ARBOR^MI^^USA||555-555-2004~444-333-222|||M\r"
>>> segment = parse_segment(pid)
>>> field = parse_field("EVERYMAN^ADAM^A") # it will return an instance of Field()
>>> component = parse_component("ID&TEST&TEST2") # it will return an instance of␣
↪Component()
```

Each parser will return an instance of the corresponding core class (e.g. *hl7apy.parser.parse_field()* will return a *hl7apy.core.Field* instance).

You can pass the `name` argument to both *hl7apy.parser.parse_field()* and *hl7apy.parser.parse_component()* functions to assign the name of the corresponding *hl7apy.core.Field* and *hl7apy.core.Component* instances returned by the functions, since it is not possible to infer their names by simply parsing the input strings:

```
>>> from hl7apy.parser import parse_field, parse_component

>>> field = parse_field("EVERYMAN^ADAM^A", name="PID_5") # it will return an instance␣
↪of Field("PID_5")
>>> component = parse_component("AUTH&1.3.6.1.4.1.21367.2011.2.5.17&ISO", name="CX_4
↪") # it will return an instance of Component("CX_4")
```

2.1.5 ER7 encoding

You can get the ER7-encoded string of `Message`, `Group`, `Segment`, `Field`, `Component` instances by simply calling the `hl7apy.Element.to_er7()` method:

```
>>> from hl7apy.parser import parse_segment

>>> pid = "PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^ANN␣
↪ARBOR^MI^^USA||555-555-2004~444-333-222|||M\r"
>>> segment = parse_segment(pid)
>>> print(segment.to_er7())
```

You can also use custom encoding chars:

```
>>> from hl7apy.parser import parse_segment

>>> custom_chars = {'FIELD': '!', 'COMPONENT': '@', 'SUBCOMPONENT': '%', 'REPETITION
↪': '~', 'ESCAPE': '$'}
>>> pid = "PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^ANN␣
↪ARBOR^MI^^USA||555-555-2004~444-333-222|||M\r"
>>> segment = parse_segment(pid)
>>> print(segment.to_er7(encoding_chars=custom_chars))
```

For `Message` objects, you can get the string ready to be sent using mllp, by calling `hl7apy.Element.to_mllp()` method:

```
>>> m = Message('OML_O33')
>>> m.to_mllp()
```

2.1.6 Datatypes

Library supports both base and complex datatypes according to standard specifications. Elements that can have a datatype are Field, Component and SubComponent, the latter supports only base datatypes. Components and SubComponents name are defined as follows:

- If the name is specified it must be <complex_datatype>_<position>
- If the name is not specified it is the name of the datatype

```
>>> f = Field('PID_1')
>>> f.datatype # it prints 'SI'
>>> f = Field('PID_3')
>>> f.datatype # it prints 'CX'
```

(continues on next page)

(continued from previous page)

```
>>> c = Component('CX_10') # the component is part of a complex datatype (CX)
>>> s = SubComponent('CWE_1') # the subcomponent is part of a complex datatype (CWE)
>>> c = Component(datatype='CWE') # the name is 'CWE'
>>> s = SubComponent(datatype='ST') # the name is 'ST'
```

The library implements base datatypes classes and validation of their values

```
>>> from hl7apy.v2_4 import ST, NM, DTM #...the list of datatypes depends on the␣
↪version

>>> s = ST('some information')
>>> s = ST(1000*'a') # it raises an exceptions since the given value exceeds the max␣
↪length for an ST datatype
>>> n = NM(111)
>>> n = NM(11111) # it raises an exceptions since the given value exceeds the max␣
↪length for a NM datatype
>>> d = DTM('20131010')
>>> d = DTM('10102013') # it raises an exceptions since the given value is not a␣
↪valid DTM value
```

In the case of SubComponent the `value` can also be an instance of a base datatype

```
>>> s = SubComponent(datatype="FT")
>>> s.value = FT('some information')
```

The `WD` datatype is not an actual datatype. It is used to identify Fields Withdrawn by the specification. If this field is present, STRICT validation fails.

2.1.7 Elements manipulation

You can visit an element's children in different ways:

- by name
- by long name (as defined in HL7 official structures)
- by position

```
>>> s = Segment('PID')
>>> s.pid_5 # by name
>>> s.patient_name # by long name
>>> s.pid_5.pid_5_1 # by position
```

Please note that child traversal is case insensitive (e.g. s.PATIENT_NAME is the same as s.patient_name)

By default the returned child is always the first, because usually an element have only one instance for a child. If you want to access to another child you have to specify the index

```
>>> s.pid_13 # it is the same as s.pid_13[0]
>>> s.pid_13[1] # it returns the second instance of pid_13 (if it exists)
```

If you want to access to a Field's children you can also use the following syntax:

```
>>> org_5 = Field('org_5') # the datatype is CX
>>> org_5.org_5_10 # it returns the tenth component of the field. It is the same as␣
↪org_5.cx_10
```

(continues on next page)

(continued from previous page)

```
>>> org_5.org_5_10_3 # it returns the third subcomponent of the tenth component of␣
↪the field. It is the same as org_5.cx_10.cwe_3

>>> org_4 = Field('ORG_4') # the datatype is ID
>>> org_4.org_4_1_1 # it raises an exception since org_4_1 is a base_datatype and␣
↪doesn't have a subcomponent
```

If you want to iterate over an element's children

```
>>> m = Message()
>>> for child in m.children:
>>>     # do something useful with child
```

You can also iterate over all the repetitions of a given child

```
>>> m = Message('OML_O33')
>>> for spm in m.spm: # in this case returns all the children named spm, not just the␣
↪first one
>>>     # do something useful with spm
```

You can delete a child from an elements

```
>>> m = Message('OML_O33')
>>> del m.MSA # it deletes the first msa
>>> del m.spm[1].spm_1 # it deletes the spm_1 field of the second spm segment
```

During children traversal if you try to access to an element which has not been created yet, it returns an empty list (if the child is valid)

```
>>> f = Field('PID_3')
>>> f.cx_10 # it returns []
>>> f.cx_30 # it raises an exception since cx_30 does not exist
>>> f.cx_10 = Component('CX_10')
>>> f.cx_10 # it returns [<Component CX_10>]
```

2.1.8 Version 2.7

Version 2.7 introduced the new delimiter # in MSH.2 which is optional. By default, when a version 2.7 (or newer) Message is created HL7apy includes the delimiter.

```
>>> m = Message('ADT_A01', version='2.7')
>>> print(m.to_er7())
'MSH|^~\\&#|||||20181024144452|||||2.7'
```

If the delimiter is not wanted it is possible to include the encoding chars without it

```
>>> from hl7apy import DEFAULT_ENCODING_CHARS
>>> m = Message('ADT_A01', version='2.7', encoding_chars=DEFAULT_ENCODING_CHARS)
>>> print(m.to_er7())
'MSH|^~\\&|||||20181024144452|||||2.7'
```

When a 2.7 message is parsed, the delimiter is included if present in the original message

```
>>> hl7_1 = "MSH|^~\&#|GHH_ADT||||20080115153000||ADT^A01^ADT_A01|0123456789|P|2.
↪7||||AL\r"
>>> hl7_2 = "MSH|^~\&|GHH_ADT||||20080115153000||ADT^A01^ADT_A01|0123456789|P|2.
↪7||||AL\r"
>>> m1 = parse_message(hl7_1)
>>> m1.to_er7()
'MSH|^~\\&#|GHH_ADT||||20080115153000||ADT^A01^ADT_A01|0123456789|P|2.7||||AL'
>>> m2 = parse_message(hl7_2)
'MSH|^~\\&|GHH_ADT||||20080115153000||ADT^A01^ADT_A01|0123456789|P|2.7||||AL'
```

2.1.9 Message Profiles

It is possible to create or parse a message using message profiles instead of the standard HL7 structures.

To use a message profile, first you need to create a file that HL7apy can interpret. The file must be created using the utility script `hl7apy_profile_parser` which needs the XML static definition of the profile as input.

The command below will create the file for `message_profile.xml`

```
python hl7apy_profile_parser message_profile.xml -o $HOME/message_profile
```

To create messages according to a message profile, it is necessary to load the corresponding file and pass it when instantiating of parsing a *Message*

```
>>> from hl7apy import load_message_profile
>>> mp = load_message_profile('$HOME/message_profile')
>>> m1 = Message('RSP_K21', reference=mp)
>>> m2 = parse_message(er7_str, message_profile=mp)
```

Now the children will be created using the profile specification

---

Important: The message profile can be specified just for the message and not for other elements. The structures of the children will be kept internally by the *Message*. This means that when populating the message, in case of message profile, in order to guarantee that the correct children references will be used, it is necessary to create each child using element's traversal or the specific *Element*'s methods (`add_group`, `add_segment`, ecc) instead of the `add()` method.

For example, let's consider a message profile that specifies the datatype of the PID.3 to be CWE (the official one is CX).

```
>>> mp = load_message_profile('$HOME/message_profile')
>>> m = Message('RSP_K21', reference=mp)
>>> m.pid.pid_3.cwe_1 = 'aaa'  # populate the first occurrence of pid_3.
>>> pid_3 = m.pid.add_field('PID_3')  # create a second occurrence
>>> pid_3.cwe_1 = 'bbb'
```

In this example, since we are using traversal and `add_field()` method, the library will use the PID.3 structure specified in the message profile. If we create the children separately the library will use the official HL7 structures.

```
>>> m = Message('RSP_K21', reference=mp)
>>> pid_3 = Field('PID_3')
>>> pid_3.cwe_1  #  this will raise an error, since the official datatype is 'CX'
```

Important: From version *1.3.0* the structure of message profiles has changed and the previous versions structures are not supported anymore. To use the new structure just recreate it with the *hl7apy_profile_parser*

2.1.10 Validation

The library supports 2 levels of validation: `STRICT` and `TOLERANT`.

In `STRICT` mode, the elements should completely adhere to the structures defined by HL7. In particular, the library checks:

- children name (e.g. a segment is not a valid child of a message according to the message's structure)
- children cardinality (e.g. a segment is mandatory and it is missing in the message)
- value constraints (e.g. a field of datatype ST that exceeds 200 chars)

Moreover, when using `STRICT` validation it is not possible to instantiate an unknown element - instantiating a `Message`, `Group`, `Field`, `Component` with `name=None` is not allowed.

The following examples will raise an exception in case of `STRICT` validation:

```
>>> from hl7apy.core import Message
>>> from hl7apy.consts import VALIDATION_LEVEL

>>> m = Message("ADT_A01", validation_level=VALIDATION_LEVEL.STRICT) # note that the␣
↪MSH segment is automatically created when instantiating a Message
>>> m.add_segment('MSH') # a Message cannot have more than 1 MSH segment
Traceback (most recent call last):
...
MaxChildLimitReached: Cannot add <Segment MSH>: max limit (1) reached for <Message␣
↪ADT_A01>

>>> m.msh.pid_1 = Field('PID_1')
Traceback (most recent call last):
...
ChildNotValid: <Field PID_1 (SET_ID_PID) of type SI> is not a valid child for
↪<Segment MSH>

>>> m.msh.msh_7 = 'abcde' # its value should be a valid DTM value (e.g. 20130101)
Traceback (most recent call last):
...
ValueError: abcde is not an HL7 valid date value
```

In `TOLERANT` mode, the library does not perform the checks listed above, but you can still verify if an element created with `TOLERANT` validation is compliant to the standard by calling the `hl7apy.core.Element.validate()` method:

```
>>> from hl7apy.core import Message

>>> m = Message("ADT_A01")
>>> m.validate()
```

When a message is created using a message profile, the validation will be performed using it as reference.

The validate method can also save a report file with all the errors and warnings occurred during validation. You just need to specify the file path as input

```
>>> m.validate(report_file='report')
```

2.1.11 Z Elements

The library supports the use of Z Elements which are Z messages, Z segments and Z fields

A Z Message can be created using a name starting with Z: both parts of the trigger event must start with a Z

```
>>> m = Message('ZBE_Z01') # This is allowed
>>> m = Message('ZBEZ01') # This is not allowed
>>> m = Message('ZBE_A01') # This is not allowed
```

You can add every kind of segment to a Z Message, both normal segment or Z segment. Also groups are allowed.

```
>>> m = Message('ZBE_Z01') # This is allowed
>>> m.pid = 'PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^
↪ANN ARBOR^MI^^USA||555-555-2004~444-333-222|||M\r'
>>> m.zin = 'ZIN|aa|bb|cc'
>>> m.add(Group('ADT_A01_INSURANCE'))
```

When encoding to ER7, segments and groups are encoded in the order of creation

```
>>> m = Message('ZBE_Z01') # This is allowed
>>> m.pid = 'PID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME STREET^^
↪ANN ARBOR^MI^^USA||555-555-2004~444-333-222|||M\r'
>>> m.zin = 'ZIN|aa|bb|cc'
>>> m.to_er7()
'MSH|^~\\&|||||20140731143925|||||2.5\rPID|1||566-554-3423^^^GHH^MR||EVERYMAN^ADAM^
↪A|||M|||2222 HOME STREET^^ANN ARBOR^MI^^USA||555-555-2004~444-333-
↪222|||M\rZIN|aa|bb|cc'
```

A Z segment is a segment that have the name starting with a Z

```
>>> s = Segment('ZBE') # This is allowed
>>> s = Segment('ZCEV') # This is not allowed
```

As other segments, you can add fields with the positional name or unknown fields, (the latter in `TOLERANT` only)

```
>>> s = Segments('ZIN')
>>> s.zin_1 = 'abc'
>>> s.add_field('zin_2')
>>> zin_3 = Field('ZIN_3', datatype='CX')
>>> s.add(zin_3)
```

Z fields are fields belonging to a Z segment. They're named with the name of the segment plus the position

```
>>> f = Field('ZIN_1')
```

By default a Z field's datatype is `ST`. When the value assigned to the `Field` contains more than one component, its datatype is converted to `None`

```
>>> f = Field('ZIN_1')
>>> f.datatype # 'ST'
>>> f.value = 'abc^def'
>>> f.datatype # None
```

Validation of Z elements follow the same rules of the other elements. So for example you can't a Field of datatype None is not validated

```
>>> f = Field('ZIN_1')
>>> f.value = 'abc^def'
>>> f.validate() # False
```

2.1.12 MLLP Server implementation

HL7apy provides an implementation of MLLP server that can be found in the module *hl7apy.mllp*. To manage different types of incoming messages, it is necessary to implement a specific handler for every kind of message. All handlers must be passed to *MLLPServer* in the `handlers` dictionary (see the *MLLPServer* documentation for details about `handlers`).

For example, let's consider a situation where we need to handle QBP^Q21^QBP_Q21 messages. We will create a class for this kind of message, subclassing *AbstractHandler*.

```
>>> from hl7apy.parser import parse_message
>>> from hl7apy.mllp import AbstractHandler
>>>
>>> class PDQHandler(AbstractHandler):
>>>     def reply(self):
>>>         msg = parse_message(self.incoming_message)
>>>         # do something with the message
>>>
>>>         res = Message('RSP_K21')
>>>         # populate the message
>>>         return res.to_mllp()
```

Then we instantiate the server with the correct `handlers`.

```
>>> from hl7apy.mllp import MLLPServer

>>> handlers = {
>>>     'QBP^Q22^QBP_Q21': (PDQHandler,) # value is a tuple
>>> }

>>> server = MLLPServer('localhost', 2575, handlers)
```

We can also implement a handler that accepts custom arguments. In the example below, the handler is provided with the name of the demographic database to retrieve the patients information from.

```
>>> from hl7apy.parser import parse_message
>>> from hl7apy.mllp import AbstractHandler
>>>
>>> class PDQHandler(AbstractHandler):
>>>     def __init__(self, msg, database_name):
>>>         super(PDQHandler, self).__init__(msg)
>>>         self.database_name = database_name
>>>
>>>     def reply(self):
>>>         msg = parse_message(self.incoming_message)
>>>         # do something with the message
>>>         res = Message('RSP_K21')
>>>         # populate the message
>>>         return res.to_mllp()
```

(continues on next page)

(continued from previous page)

```
>>>
>>> handlers = {
>>>     'QBP^Q22^QBP_Q21': (PDQHandler, 'db_name')
>>> }
```

It is also possible to implement a subclass of *AbstractErrorHandler* to handle exceptions that may occur (e.g., the reception of an unsupported message). The instance of the `Exception` can be accessed through the attribute `exc`.

```
>>> from hl7apy.mllp import UnsupportedMessageType
>>>
>>> class ErrorHandler(AbstractErrorHandler):
>>>     def reply(self):
>>>         if isinstance(self.exc, UnsupportedMessageType):
>>>             # return your custom response for unsupported message
>>>         else:
>>>             # return your custom response for general errors
>>>
>>>
>>> handlers = {
>>>     'QBP^Q22^QBP_Q21': (PDQHandler, 'demographic_db'),
>>>     'ERR': (ErrorHandler,)
>>> }
```

2.2 API Docs

HL7apy API documentation.

2.2.1 Library helper functions

hl7apy.**check_encoding_chars**(*encoding_chars*)
Validate the given encoding chars

Parameters **encoding_chars** (dict) – the encoding chars (see *hl7apy.set_default_encoding_chars()*)

Raises *hl7apy.exceptions.InvalidEncodingChars* if the given encoding chars are not valid

hl7apy.**check_validation_level**(*validation_level*)
Validate the given validation level

Parameters **validation_level** (int) – validation level (see *hl7apy.consts.VALIDATION_LEVEL*)

Raises *hl7apy.exceptions.UnknownValidationLevel* if the given validation level is unsupported

hl7apy.**check_version**(*version*)
Validate the given version number

Parameters **version** (str) – the version to validate (e.g. `2.6`)

Raises *hl7apy.exceptions.UnsupportedVersion* if the given version is unsupported

hl7apy.**find_reference**(*name*, *element_types*, *version*)
Look for an element of the given name and version into the given types and return its reference structure

Parameters

- **name** (str) – the element name to look for (e.g. 'MSH')
- **types** (list or tuple) – the element classes where to look for the element (e.g. (Group, Segment))
- **version** (str) – the version of the library where to search the element (e.g. '2.6')

Return type dict

Returns a dictionary describing the element structure

Raise *hl7apy.exceptions.ChildNotFound* if the element has not been found

```
>>> from hl7apy.core import Message, Segment
>>> find_reference('UNKNOWN', (Segment, ), '2.5')
Traceback (most recent call last):
...
ChildNotFound: No child named UNKNOWN
>>> find_reference('ADT_A01', (Segment,),  '2.5')
Traceback (most recent call last):
...
ChildNotFound: No child named ADT_A01
>>> r = find_reference('ADT_A01', (Message,),  '2.5')
>>> print('%s %s' % (r['name'], r['cls']))
ADT_A01 <class 'hl7apy.core.Message'>
```

hl7apy.**get_default_encoding_chars**(*version=None*)
Get the default encoding chars

Return type dict

Returns the encoding chars (see *hl7apy.set_default_encoding_chars()*)

```
>>> print(get_default_encoding_chars('2.6')['FIELD'])
|
```

hl7apy.**get_default_validation_level**()
Get the default validation level

Return type str

Returns the default validation level

```
>>> print(get_default_validation_level())
2
```

hl7apy.**get_default_version**()
Get the default version

Return type str

Returns the default version

```
>>> print(get_default_version())
2.5
```

hl7apy.**load_library**(*version*)
Load the correct module according to the version

Parameters **version** (str) – the version of the library to be loaded (e.g. '2.6')

Return type module object

hl7apy.**load_reference**(*name*, *element_type*, *version*)

Look for an element of the given type, name and version and return its reference structure

Parameters

- **element_type** (str) – the element type to look for (e.g. 'Segment')
- **name** (str) – the element name to look for (e.g. 'MSH')
- **version** (str) – the version of the library where to search the element (e.g. '2.6')

Return type dict

Returns a dictionary describing the element structure

Raise KeyError if the element has not been found

The returned dictionary will contain the following keys:

Key	Value
cls	an *hl7apy.core.Element* subclass
name	the Element name (e.g. PID)
ref	a tuple of one of the following format: ('leaf', <datatype>, <longName>, <table>) ('sequence', (<child>, (<min>, <max>), ...))

```
>>> load_reference('UNKNOWN', 'Segment', '2.5')
Traceback (most recent call last):
...
ChildNotFound: No child named UNKNOWN
>>> r = load_reference('ADT_A01', 'Message', '2.5')
>>> print(r[0])
sequence
>>> r = load_reference('MSH_3', 'Field', '2.5')
>>> print(r[0])
sequence
```

hl7apy.**set_default_encoding_chars**(*encoding_chars*)

Set the given encoding chars as default

Parameters **encoding_chars** (dict) – the new encoding chars

Raises *hl7apy.exceptions.InvalidEncodingChars* if the given encoding chars are not valid

The *encoding_chars* dictionary should contain the following keys:

Key	Default
GROUP	\r
SEGMENT	\r
FIELD	|
COMPONENT	^
SUBCOMPONENT	&
REPETITION	~
ESCAPE	\

```
>>> set_default_encoding_chars({'FIELD': '!'})
Traceback (most recent call last):
    ...
InvalidEncodingChars: Missing required encoding chars
>>> set_default_encoding_chars({'FIELD': '!', 'COMPONENT': 'C', 'SUBCOMPONENT': 'S
↪',                                    'REPETITION': 'R', 'ESCAPE': '\\'})
>>> print(get_default_encoding_chars('2.5')['FIELD'])
!
```

hl7apy.**set_default_validation_level**(*validation_level*)

Set the given validation level as default

Parameters **validation_level** (int) – validation level (see *hl7apy.consts.VALIDATION_LEVEL*)

Raises *hl7apy.exceptions.UnknownValidationLevel* if the given validation level is unsupported

```
>>> set_default_validation_level(3)
Traceback (most recent call last):
    ...
UnknownValidationLevel
>>> set_default_validation_level(VALIDATION_LEVEL.TOLERANT)
>>> print(get_default_validation_level())
2
```

hl7apy.**set_default_version**(*version*)

Set the given version as default

Parameters **version** (str) – the new default version (e.g. 2.6)

Raises *hl7apy.exceptions.UnsupportedVersion* if the given version is unsupported

```
>>> set_default_version('22')
Traceback (most recent call last):
    ...
UnsupportedVersion: The version 22 is not supported
>>> set_default_version('2.3')
>>> print(get_default_version())
2.3
```

2.2.2 Core classes

HL7apy - core classes

class hl7apy.core.**Element**(*name=None*, *parent=None*, *reference=None*, *version=None*, *validation_level=None*, *traversal_parent=None*)

Base class for all HL7 elements. It is not meant to be directly instantiated.

class hl7apy.core.**Message**(*name=None*, *reference=None*, *version=None*, *validation_level=None*, *encoding_chars=None*)

Class representing an HL7 message

Parameters

- **name** (str) – the HL7 name of the message (e.g. OML_O33)
- **name** – the HL7 name of the segment (e.g. PID)

- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *get_default_encoding_chars*)

add(*obj*)

Add an instance of *Element* subclass to the list of children

Parameters **obj** (*hl7apy.core.Element*) – an instance of *hl7apy.core.Element* subclass

```
>>> s = Segment('PID')
>>> f = Field('PID_5')
>>> f.value = 'EVERYMAN^ADAM'
>>> s.add(f)
>>> print(s.to_er7())
PID|||||EVERYMAN^ADAM
```

add_group(*name*)

Create an instance of *Group* having the given name

Parameters **name** – the name of the group to be created (e.g. OML_O33_PATIENT)

Returns an instance of *Group*

```
>>> m = Message('OML_O33')
>>> patient = m.add_group('OML_O33_PATIENT')
>>> print(patient)
<Group OML_O33_PATIENT>
>>> print(patient in m.children)
True
```

add_segment(*name*)

Create an instance of *Segment* having the given name

Parameters **name** – the name of the segment to be created (e.g. PID)

Returns an instance of *Segment*

```
>>> m = Message('QBP_Q11')
>>> qpd = m.add_segment('QPD')
>>> print(qpd)
<Segment QPD>
>>> print(qpd in m.children)
True
```

to_er7(*encoding_chars=None*, *trailing_children=False*)

Returns the HL7 representation of the *Element*. It adds the appropriate separator at the end if needed

Parameters **encoding_chars** (dict) – The encoding chars to use. If it is None it uses self.encoding_chars, which by default is the ones return by *get_default_encoding_chars* values

Return type str

Returns the HL7 representation of the *Element*

to_mllp(*encoding_chars=None*, *trailing_children=False*)

Returns the er7 representation of the message wrapped with mllp encoding characters

Parameters

- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *get_default_encoding_chars*)

- **trailing_children** (bool) – if True, trailing children will be added even if their value is None

Returns the ER7-encoded string wrapped with the mllp encoding characters

validate(*report_file=None*)

Validate the HL7 element using the *STRICT* validation level. It calls the *Validator.validate* method passing the reference used in the instantiation of the element.

Param report_file: the report file to pass to the validator

class hl7apy.core.**Group**(*name=None*, *parent=None*, *reference=None*, *version=None*, *validation_level=None*, *traversal_parent=None*)

Class representing an HL7 segment group

Parameters

- **name** (str) – the HL7 name of the message (e.g. RSP_K21_QUERY_RESPONSE)
- **parent** (an instance of *Message*, *Group* or None) – the parent
- **reference** – the reference structure (see *load_reference*)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *get_default_version*)
- **validation_level** – the validation level. Possible values are those defined in *VALIDATION_LEVEL* or None to use the default validation level (see *get_default_validation_level*)

add(*obj*)

Add an instance of *Element* subclass to the list of children

Parameters **obj** (*hl7apy.core.Element*) – an instance of *hl7apy.core.Element* subclass

```
>>> s = Segment('PID')
>>> f = Field('PID_5')
>>> f.value = 'EVERYMAN^ADAM'
>>> s.add(f)
>>> print(s.to_er7())
PID|||||EVERYMAN^ADAM
```

add_group(*name*)

Create an instance of *Group* having the given name

Parameters **name** – the name of the group to be created (e.g. OML_O33_PATIENT)

Returns an instance of *Group*

```
>>> m = Message('OML_O33')
>>> patient = m.add_group('OML_O33_PATIENT')
>>> print(patient)
<Group OML_O33_PATIENT>
>>> print(patient in m.children)
True
```

add_segment(*name*)

Create an instance of *Segment* having the given name

Parameters **name** – the name of the segment to be created (e.g. PID)

Returns an instance of *Segment*

```
>>> m = Message('QBP_Q11')
>>> qpd = m.add_segment('QPD')
>>> print(qpd)
<Segment QPD>
>>> print(qpd in m.children)
True
```

to_er7(*encoding_chars=None*, *trailing_children=False*)
Returns the HL7 representation of the *Element*. It adds the appropriate separator at the end if needed

Parameters **encoding_chars** (dict) – The encoding chars to use. If it is None it uses self.encoding_chars, which by default is the ones return by *get_default_encoding_chars* values

Return type str

Returns the HL7 representation of the *Element*

validate(*report_file=None*)
Validate the HL7 element using the *STRICT* validation level. It calls the *Validator.validate* method passing the reference used in the instantiation of the element.

Param report_file: the report file to pass to the validator

class hl7apy.core.**Segment**(*name=None*, *parent=None*, *reference=None*, *version=None*, *validation_level=None*, *traversal_parent=None*)
Class representing an HL7 segment.

Parameters

- **name** (str) – the HL7 name of the segment (e.g. PID)
- **parent** (an instance of *Message*, *Group* or None) – the parent
- **reference** – the reference structure (see *load_reference*)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *get_default_version*)
- **validation_level** – the validation level. Possible values are those defined in *VALIDATION_LEVEL* or None to use the default validation level (see *get_default_validation_level*)
- **traversal_parent** (an instance of *hl7apy.core.Message*, *hl7apy.core.Group* or None) – the temporary parent used during traversal

add(*obj*)
Add an instance of *Element* subclass to the list of children

Parameters **obj** (*hl7apy.core.Element*) – an instance of *hl7apy.core.Element* subclass

```
>>> s = Segment('PID')
>>> f = Field('PID_5')
>>> f.value = 'EVERYMAN^ADAM'
>>> s.add(f)
>>> print(s.to_er7())
PID|||||EVERYMAN^ADAM
```

add_field(*name*)
Create an instance of *Field* having the given name

Parameters **name** – the name of the field to be created (e.g. PID_1)

Returns an instance of *Field*

```
>>> s = Segment('PID')
>>> print(s.add_field('PID_1'))
<Field PID_1 (SET_ID_PID) of type SI>
```

to_er7(*encoding_chars=None*, *trailing_children=False*)
Return the ER7-encoded string

Parameters

- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *get_default_encoding_chars*)
- **trailing_children** (bool) – if True, trailing children will be added even if their value is None

Returns the ER7-encoded string

```
>>> pid = Segment("PID")
>>> pid.pid_1 = '1'
>>> pid.pid_5 = "EVERYMAN^ADAM"
>>> print(pid.to_er7())
PID|1||||EVERYMAN^ADAM
```

validate(*report_file=None*)
Validate the HL7 element using the *STRICT* validation level. It calls the *Validator.validate* method passing the reference used in the instantiation of the element.

Param report_file: the report file to pass to the validator

class hl7apy.core.**Field**(*name=None*, *datatype=None*, *parent=None*, *reference=None*, *version=None*, *validation_level=None*, *traversal_parent=None*)
Class representing an HL7 field.

Parameters

- **name** (str) – the HL7 name of the field (e.g. PID_5)
- **datatype** (str) – the datatype of the field (e.g. CE)
- **parent** (an instance of *hl7apy.core.Segment* or None) – the parent
- **reference** – the reference structure (see *load_reference*)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *get_default_version*)
- **validation_level** – the validation level. Possible values are those defined in *VALIDATION_LEVEL* or None to use the default validation level (see *get_default_validation_level*)
- **traversal_parent** (an instance of *Segment* or None) – the temporary parent used during traversal

add(*obj*)
Add an instance of *Component* to the list of children

Parameters **obj** – an instance of *Component*

```
>>> f = Field('PID_5')
>>> f.xpn_1 = 'EVERYMAN'
>>> c = Component('XPN_2')
```

(continues on next page)

(continued from previous page)

```
>>> c.value = 'ADAM'
>>> f.add(c)
>>> print(f.to_er7())
EVERYMAN^ADAM
```

add_component(*name*)

Create an instance of *Component* having the given name

Parameters **name** – the name of the component to be created (e.g. XPN_2)

Returns an instance of *Component*

```
>>> s = Field('PID_5')
>>> print(s.add_component('XPN_2'))
<Component XPN_2 (GIVEN_NAME) of type ST>
```

to_er7(*encoding_chars=None*, *trailing_children=False*)

Return the ER7-encoded string

Parameters

- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *get_default_encoding*)
- **trailing_children** (bool) – if True, trailing children will be added even if their value is None

Returns the ER7-encoded string

```
>>> msh_9 = Field("MSH_9")
>>> msh_9.value = "ADT^A01^ADT_A01"
>>> print(msh_9.to_er7())
ADT^A01^ADT_A01
```

validate(*report_file=None*)

Validate the HL7 element using the *STRICT* validation level. It calls the *Validator.validate* method passing the reference used in the instantiation of the element.

Param report_file: the report file to pass to the validator

class hl7apy.core.**Component**(*name=None*, *datatype=None*, *parent=None*, *reference=None*, *version=None*, *validation_level=None*, *traversal_parent=None*)

Class representing an HL7 component.

Parameters

- **name** (str) – the HL7 name of the component (e.g. XPN_2)
- **datatype** (str) – the datatype of the component (e.g. CE)
- **parent** (an instance of *hl7apy.core.Field* or None) – the parent
- **reference** – the reference structure (see *load_reference*)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *get_default_version*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* or None to use the default validation level (see *get_default_validation_level*)

- **traversal_parent** (an instance of `Field hl7apy.core.Field` or None) – the temporary parent used during traversal

add(*obj*)

Add an instance of *SubComponent* to the list of children

Parameters **obj** – an instance of *SubComponent*

```
>>> c = Component('CX_10')
>>> s = SubComponent(name='CWE_1', value='EXAMPLE_ID')
>>> s2 = SubComponent(name='CWE_4', value='ALT_ID')
>>> c.add(s)
>>> c.add(s2)
>>> print(c.to_er7())
EXAMPLE_ID&&&ALT_ID
```

add_subcomponent(*name*)

Create an instance of *SubComponent* having the given name

Parameters **name** – the name of the subcomponent to be created (e.g. CE_1)

Returns an instance of *SubComponent*

```
>>> c = Component(datatype='CE')
>>> ce_1 = c.add_subcomponent('CE_1')
>>> print(ce_1)
<SubComponent CE_1>
>>> print(ce_1 in c.children)
True
```

to_er7(*encoding_chars=None*, *trailing_children=False*)

Returns the HL7 representation of the *Element*. It adds the appropriate separator at the end if needed

Parameters **encoding_chars** (`dict`) – The encoding chars to use. If it is `None` it uses `self.encoding_chars`, which by default is the ones return by *get_default_encoding_chars* values

Return type `str`

Returns the HL7 representation of the *Element*

validate(*report_file=None*)

Validate the HL7 element using the *STRICT* validation level. It calls the *Validator.validate* method passing the reference used in the instantiation of the element.

Param report_file: the report file to pass to the validator

class `hl7apy.core.`**SubComponent**(*name=None*, *datatype=None*, *value=None*, *parent=None*, *reference=None*, *version=None*, *validation_level=None*, *traversal_parent=None*)

Class representing an HL7 subcomponent.

Parameters

- **name** (`str`) – the HL7 name of the subcomponent (e.g. CWE_1)
- **datatype** (`str`) – the datatype of the component (e.g. ST)
- **value** (`str` or instance of *BaseDataType*) – the value of the subcomponent (e.g. ADT_A01)
- **parent** (an instance of *Component* or None) – the parent
- **reference** – the reference structure (see *load_refernce*)

- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *get_default_version*)
- **validation_level** (int) – the validation level. Possible values are defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *get_default_validation_level*)
- **traversal_parent** (an instance of *Component* or None) – the temporary parent used during traversal

to_er7(*encoding_chars=None*, *trailing_children=False*)
Return the ER7-encoded string

Parameters

- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *get_default_encoding_chars*)
- **trailing_children** (bool) – if True, trailing children will be added even if their value is None

Returns the ER7-encoded string

```
>>> s = SubComponent("CE_1")
>>> s.value = "IDENTIFIER"
>>> print(s.to_er7())
IDENTIFIER
```

validate(*report_file=None*)
Validate the HL7 element using the *STRICT* validation level. It calls the *Validator.validate* method passing the reference used in the instantiation of the element.

Param report_file: the report file to pass to the validator

2.2.3 Consts

HL7apy - Constants

hl7apy.consts.**DEFAULT_ENCODING_CHARS = {'COMPONENT': '^', 'ESCAPE': '\\', 'FIELD': '|', 'GI**
Dictionary with default encoding characters as per standard specifications

hl7apy.consts.**DEFAULT_VERSION = '2.5'**
default hl7 version

class hl7apy.consts.**VALIDATION_LEVEL**
Allowed validation levels

STRICT = 1
Strict validation

TOLERANT = 2
Tolerant validation

class hl7apy.consts.**MLLP_ENCODING_CHARS**
MLLP encoding chars

CR = '\r'
Carriage return

EB = '\x1c'
End Block

SB = '\x0b'
Start Block

2.2.4 Parser

hl7apy.parser.**parse_message**(*message*, *validation_level=None*, *find_groups=True*, *message_profile=None*, *report_file=None*, *force_validation=False*)
Parse the given ER7-encoded message and return an instance of *Message*.

Parameters

- **message** (str) – the ER7-encoded message to be parsed
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *set_default_validation_level*)
- **find_groups** (bool) – if True, automatically assign the segments found to the appropriate *Groups* instances. If False, the segments found are assigned as children of the *Message* instance

Returns an instance of *Message*

```
>>> message = "MSH|^~\&|GHH_ADT||||20080115153000||OML^O33^OML_O33|0123456789|P|2.
↪5||||AL\rPID|1||"     "566-554-3423^^^GHH^MR||EVERYMAN^ADAM^A|||M|||2222 HOME␣
↪STREET^^ANN ARBOR^MI^^USA||555-555-2004|||M\r"
>>> m = parse_message(message)
>>> print(m)
<Message OML_O33>
>>> print(m.msh.sending_application.to_er7())
GHH_ADT
>>> print(m.children)
[<Segment MSH>, <Group OML_O33_PATIENT>]
```

hl7apy.parser.**parse_segments**(*text*, *version=None*, *encoding_chars=None*, *validation_level=None*, *references=None*, *find_groups=False*)
Parse the given ER7-encoded segments and return a list of *hl7apy.core.Segment* instances.

Parameters

- **text** (str) – the ER7-encoded string containing the segments to be parsed
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *set_default_version*)
- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *set_default_encoding_chars*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *validation_level*)
- **references** (list) – A list of the references of the *Segment*'s children
- **find_groups** (bool) – if True, automatically assign the segments found to the appropriate *Groups* instances. If False, the segments found are assigned as children of the *Message* instance

Returns a list of *Segment* instances

```
>>> segments = "EVN||20080115153000||||20080114003000\rPID|1||566-554-3423^^^GHH^
↪MR||EVERYMAN^ADAM^A|||M|||"     "2222 HOME STREET^^ANN ARBOR^MI^^USA||555-555-
↪2004|||M\r"
>>> print(parse_segments(segments))
[<Segment EVN>, <Segment PID>]
```

hl7apy.parser.**parse_segment**(*text*, *version=None*, *encoding_chars=None*, *validation_level=None*, *reference=None*)

Parse the given ER7-encoded segment and return an instance of *Segment*.

Parameters

- **text** (str) – the ER7-encoded string containing the segment to be parsed
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *set_default_version*)
- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *set_default_encoding_chars*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *set_default_validation_level*)
- **reference** (dict) – a dictionary containing the element structure returned by *load_reference*, *find_reference* or belonging to a message profile

Returns an instance of *Segment*

```
>>> segment = "EVN||20080115153000||||20080114003000"
>>> s =  parse_segment(segment)
>>> print(s)
<Segment EVN>
>>> print(s.to_er7())
EVN||20080115153000||||20080114003000
```

hl7apy.parser.**parse_fields**(*text*, *name_prefix=None*, *version=None*, *encoding_chars=None*, *validation_level=None*, *references=None*, *force_varies=False*)

Parse the given ER7-encoded fields and return a list of *hl7apy.core.Field*.

Parameters

- **text** (str) – the ER7-encoded string containing the fields to be parsed
- **name_prefix** (str) – the field prefix (e.g. MSH)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *set_default_version*)
- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *set_default_encoding_chars*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *set_default_validation_level*)
- **references** (list) – A list of the references of the *Field*'s children
- **force_varies** (bool) – flag that force the fields to use a varies structure when no reference is found. It is used when a segment ends with a field of type varies that thus support infinite children

Returns a list of *Field* instances

```
>>> fields = "1|NUCLEAR^NELDA^W|SPO|2222 HOME STREET^^ANN ARBOR^MI^^USA"
>>> nk1_fields = parse_fields(fields, name_prefix="NK1")
>>> print(nk1_fields)
[<Field NK1_1 (SET_ID_NK1) of type SI>, <Field NK1_2 (NAME) of type XPN>, <Field␣
↪NK1_3 (RELATIONSHIP) of type CE>, <Field NK1_4 (ADDRESS) of type XAD>]
>>> s = Segment("NK1")
>>> s.children = nk1_fields
>>> print(s.to_er7())
NK1|1|NUCLEAR^NELDA^W|SPO|2222 HOME STREET^^ANN ARBOR^MI^^USA
>>> unknown_fields = parse_fields(fields)
>>> s.children = unknown_fields
>>> print(s.to_er7())
NK1||||||||||||||||||||||||||||||||||||||||1|NUCLEAR^NELDA^W|SPO|2222 HOME STREET^
↪^ANN ARBOR^MI^^USA
```

hl7apy.parser.**parse_field**(*text*, *name=None*, *version=None*, *encoding_chars=None*, *validation_level=None*, *reference=None*, *force_varies=False*)

Parse the given ER7-encoded field and return an instance of *Field*.

Parameters

- **text** (str) – the ER7-encoded string containing the fields to be parsed
- **name** (str) – the field name (e.g. MSH_7)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *set_default_version*)
- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *set_default_encoding_chars*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *set_default_validation_level*)
- **reference** (dict) – a dictionary containing the element structure returned by *load_reference* or *find_reference* or belonging to a message profile
- **force_varies** (boolean) – flag that force the fields to use a varies structure when no reference is found. It is used when a segment ends with a field of type varies that thus support infinite children

Returns an instance of *Field*

```
>>> field = "NUCLEAR^NELDA^W"
>>> nk1_2 = parse_field(field, name="NK1_2")
>>> print(nk1_2)
<Field NK1_2 (NAME) of type XPN>
>>> print(nk1_2.to_er7())
NUCLEAR^NELDA^W
>>> unknown = parse_field(field)
>>> print(unknown)
<Field of type None>
>>> print(unknown.to_er7())
NUCLEAR^NELDA^W
```

hl7apy.parser.**parse_components**(*text*, *field_datatype='ST'*, *version=None*, *encoding_chars=None*, *validation_level=None*, *references=None*)

Parse the given ER7-encoded components and return a list of *Component* instances.

Parameters

- **text** (str) – the ER7-encoded string containing the components to be parsed
- **field_datatype** (str) – the datatype of the components (e.g. ST)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *set_default_version*)
- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *set_default_encoding_chars*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *set_default_validation_level*)
- **references** (list) – A list of the references of the *Component*'s children

Returns a list of *Component* instances

```
>>> components = "NUCLEAR^NELDA^W^^TEST"
>>> xpn = parse_components(components, field_datatype="XPN")
>>> print(xpn)
[<Component XPN_1 (FAMILY_NAME) of type FN>, <Component XPN_2 (GIVEN_NAME) of␣
↪type ST>, <Component XPN_3 (SECOND_AND_FURTHER_GIVEN_NAMES_OR_INITIALS_THEREOF)␣
↪of type ST>, <Component XPN_5 (PREFIX_E_G_DR) of type ST>]
>>> print(parse_components(components))
[<Component ST (None) of type ST>, <Component ST (None) of type ST>, <Component␣
↪ST (None) of type ST>, <Component ST (None) of type ST>, <Component ST (None)␣
↪of type ST>]
```

hl7apy.parser.**parse_component**(*text*, *name=None*, *datatype='ST'*, *version=None*, *encoding_chars=None*, *validation_level=None*, *reference=None*)

Parse the given ER7-encoded component and return an instance of *Component*.

Parameters

- **text** (str) – the ER7-encoded string containing the components to be parsed
- **name** (str) – the component's name (e.g. XPN_2)
- **datatype** (str) – the datatype of the component (e.g. ST)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *set_default_version*)
- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *set_default_encoding_chars*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *set_default_validation_level*)
- **reference** (dict) – a dictionary containing the element structure returned by *load_reference* or *find_reference* or belonging to a message profile

Returns an instance of *Component*

```
>>> component = "GATEWAY&1.3.6.1.4.1.21367.2011.2.5.17"
>>> cx_4 = parse_component(component, name="CX_4")
>>> print(cx_4)
<Component CX_4 (ASSIGNING_AUTHORITY) of type None>
>>> print(cx_4.to_er7())
GATEWAY&1.3.6.1.4.1.21367.2011.2.5.17
```

(continues on next page)

(continued from previous page)

```
>>> print(parse_component(component))
<Component ST (None) of type None>
```

hl7apy.parser.**parse_subcomponents**(*text*, *component_datatype='ST'*, *version=None*, *encoding_chars=None*, *validation_level=None*)

Parse the given ER7-encoded subcomponents and return a list of *SubComponent* instances.

Parameters

- **text** (str) – the ER7-encoded string containing the components to be parsed
- **component_datatype** (str) – the datatype of the subcomponents (e.g. ST)
- **version** (str) – the HL7 version (e.g. "2.5"), or None to use the default (see *set_default_version*)
- **encoding_chars** (dict) – a dictionary containing the encoding chars or None to use the default (see *set_default_encoding_chars*)
- **validation_level** (int) – the validation level. Possible values are those defined in *VALIDATION_LEVEL* class or None to use the default validation level (see *set_default_validation_level*)

Returns a list of *SubComponent* instances

```
>>> subcomponents= "ID&TEST&&AHAH"
>>> cwe = parse_subcomponents(subcomponents, component_datatype="CWE")
>>> print(cwe)
[<SubComponent CWE_1>, <SubComponent CWE_2>, <SubComponent CWE_4>]
>>> c = Component(datatype='CWE')
>>> c.children = cwe
>>> print(c.to_er7())
ID&TEST&&AHAH
>>> subs = parse_subcomponents(subcomponents)
>>> print(subs)
[<SubComponent ST>, <SubComponent ST>, <SubComponent ST>, <SubComponent ST>]
>>> c.children = subs
>>> print(c.to_er7())
&&&&&&&&&ID&TEST&&AHAH
```

2.2.5 Base datatypes

Warning: The HL7 versions can have different implementation of base datatypes; for example the ST base datatype of HL7 v2.6 is different from the v2.5 one. This module contains reference classes for all base datatypes but you should not import them directly from here. If you need an implementation for a particular version use the `get_base_datatypes()` function from a specific version's module For example if you're using version 2.4 and you need an *FT* base datatype do the following:

```
>>> from hl7apy.v2_4 import FT
>>> f = FT('some useful information')
```

class hl7apy.base_datatypes.**BaseDataType**(*value*, *max_length=None*, *validation_level=None*)

Generic datatype base class. It handles the value of the data type and its maximum length. It is meant to be extended and it should not be used directly

Parameters

- **value** – the value of the data type
- **max_length** (int) – The maximum length of the value. Default to None
- **validation_level** (int) – It must be a value from class *VALIDATION_LEVEL* If it is STRICT it checks that value doesn't exceed the attr:*max_length*

Raise *MaxLengthReached* When the value's length is greater than the max_length. Only if validation_level is STRICT

classname
The name of the class

to_er7(*encoding_chars=None*)
Encode to ER7 format

---

class hl7apy.base_datatypes.**TextualDataType**(*value*, *max_length=32*, *highlights=None*, *validation_level=None*)
Base class for textual data types. It is meant to be extended and it should not be used directly

Parameters

- **value** (str) – the value of the data type
- **max_length** (int) – the max length of the value (default to 32)
- **highlights** (tuple, list) – a list of ranges indicating the part of the value to be highlighted. e.g. ((0,5), (6,7)) The highlights cannot overlap, if they do an HL7Exception will be thrown when to_er7 method is called
- **validation_level** (int) – It has the same meaning as in BaseDatatype

Raise *MaxLengthReached* When the value's length is greater than max_length

Classes representing textual datatypes are:

class hl7apy.base_datatypes.**ST**(*value*, *highlights=None*, *validation_level=None*)
Class for ST datatype. It extends hl7apy.base_datatypes.TextualDatatype and the parameters are the same of the superclass

max_length is 199

class hl7apy.base_datatypes.**FT**(*value*, *highlights=None*, *validation_level=None*)
Class for FT datatype. It extends *hl7apy.base_datatypes.TextualDataType* and the parameters are the same of the superclass

max_length is 65536

class hl7apy.base_datatypes.**ID**(*value*, *highlights=None*, *validation_level=None*)
Class for ID datatype. It extends *hl7apy.base_datatypes.TextualDataType* and the parameters are the same of the superclass

max_length None

class hl7apy.base_datatypes.**IS**(*value*, *highlights=None*, *validation_level=None*)
Class for IS datatype. It extends *hl7apy.base_datatypes.TextualDataType* and the parameters are the same of the superclass

max_length is 20

class hl7apy.base_datatypes.**TX**(*value*, *highlights=None*, *validation_level=None*)
Class for TX datatype. It extends *hl7apy.base_datatypes.TextualDataType* and the parameters are the same of the superclass

max_length is 65536

class hl7apy.base_datatypes.**GTS**(*value*, *highlights=None*, *validation_level=None*)
Class for GTS datatype. It extends *hl7apy.base_datatypes.TextualDataType* and the parameters are the same of the superclass

max_length is 199

class hl7apy.base_datatypes.**WD**(*value*, *highlights=None*, *validation_level=None*)
Datatype class for withdraw fields. They are fields that has been withdrawn from specification and should not be used. It is implemented as a hl7apy.base_datatypes.TextualDatatype with max_length 0.

max_length is 0

---

class hl7apy.base_datatypes.**NumericDataType**(*value=None*, *max_length=16*, *validation_level=None*)
Base class for numeric data types. It is meant to be extended and it should not be used directly

Parameters

- **value** – the value of the data type. Default is None
- **max_length** (*int*) – The maximum number of digit in the value. Default is 16
- **validation_level** (int) – It has the same meaning as in *hl7apy.base_datatypes.BaseDataType*

Raise *hl7apy.exceptions.MaxLengthReached* When the *value*'s length is greater than *max_length*

Classes representing numeric datatypes are:

class hl7apy.base_datatypes.**NM**(*value=None*, *validation_level=None*)
Class for NM datatype. It extends hl7apy.base_datatypes.NumericDatatype and the parameters are the same of the superclass

max_length is 16.

The type of value must be decimal.Decimal or Real

Raise ValueError raised when the value is not of one of the correct type

class hl7apy.base_datatypes.**SI**(*value=None*, *validation_level=None*)
Class for NM datatype. It extends NumericDatatype and the parameters are the same of the superclass

max_length is 4.

The type of value must be *int* or numbers.Integral

Raise ValueError raised when the value is not of one of the correct type

---

class hl7apy.base_datatypes.**DateTimeDataType**(*value=None*, *out_format=''*)
Base class for datetime data types. It is meant to be extended and it should not be used directly. Children classes should at least override the allowed_formats tuple

Parameters

- **value** – a *datetime* date object. Default is None

- **out_format** (*str*) – the format that will be used converting the object to string. It must be an item of the `allowed_formats` tuple

Raise :exc:`InvalidDateFormat <hl7apy.exceptions.InvalidDateFormat>` if the ``format is not in the `allowed_formats` member

Classes representing datetime datatypes are:

class hl7apy.base_datatypes.**DT**(*value=None*, *out_format='%Y%m%d'*)
Class for DT base datatype. It extends DatetimeDatatype and it represents a time value with year, month and day. Parameters are the same of the superclass.

The `allowed_formats` tuple is `('%Y', '%Y%m', '%Y%m%d')`

class hl7apy.base_datatypes.**TM**(*value=None*, *out_format='%H%M%S.%f'*, *offset=''*, *microsec_precision=4*)
Class for TM base datatype. It extends DateTimeDatatype and it represents a time value with hours, minutes, seconds and microseconds. Parameters are the same of the superclass plus `offset`. Since HL7 supports only four digits for microseconds, and Python datetime uses 6 digits, the wanted precision must be specified.

The `allowed_formats` tuple is `('%H', '%H%M', '%H%M%S', '%H%M%S.%f')`. It needs also the `offset` parameter which represents the UTC offset

Parameters

- **offset** (`str`) – the UTC offset. By default it is ''. It must be in the form `'+/-HHMM'`
- **microsec_precision** (`int`) – Number of digit of the microseconds part of the value. It must be between 1 and 4

class hl7apy.base_datatypes.**DTM**(*value=None*, *out_format='%Y%m%d%H%M%S.%f'*, *offset=''*, *microsec_precision=4*)
Class for DTM base datatype. It extends TM and it represents classes DT and DTM combined. Thus it represents year, month, day, hours, minutes, seconds and microseconds. Parameters are the same of the superclass.

The `allowed_formats` tuple is `('%Y', '%Y%m', '%Y%m%d', '%Y%m%d%H', '%Y%m%d%H%M', '%Y%m%d%H%M%S', '%Y%m%d%H%M%S.%f')`

2.2.6 Datatype factories

hl7apy.factories.**date_factory**(*value*, *datatype_cls*, *validation_level=None*)
Creates a *DT* object

The value in input must be a string parsable with `datetime.strptime()`. The date format is chosen according to the length of the value as stated in this table:

Length	Format
4	`%Y`
6	`%Y%m`
8	`%Y%m%d`

Some examples that work are:

```
>>> from hl7apy.base_datatypes import DT
>>> date_factory("1974", DT)
<hl7apy.base_datatypes.DT object at 0x...>
>>> date_factory("198302", DT)
<hl7apy.base_datatypes.DT object at 0x...>
```

(continues on next page)

(continued from previous page)

```
>>> date_factory("19880312", DT)
<hl7apy.base_datatypes.DT object at 0x...>
```

If the value does not match one of the valid format it raises `ValueError`

Parameters

- **value** (`str`) – the value to assign the date object
- **value** – the *DT* class to use. It has to be one implementation of the different version modules
- **validation_level** (`int`) – It must be a value from class `validation_level` `VALIDATION_LEVEL` `hl7apy.consts.VALIDATION_LEVEL` or `None` to use the default value

Return type *hl7apy.base_datatypes.DT*

`hl7apy.factories.`**timestamp_factory**(*value*, *datatype_cls*, *validation_level=None*)

Creates a *TM* object

The value in input must be a string parsable with `datetime.strptime()`. It can also have an offset part specified with the format +/-HHMM. The offset can be added with all the allowed format The date format is chosen according to the length of the value as stated in this table:

Length	Format
2	`%H`
4	`%H%M`
6	`%H%M%S`
10-13	`%H%M%S.%f`

Some examples that work are:

```
>>> from hl7apy.base_datatypes import TM
>>> timestamp_factory("12", TM)
<hl7apy.base_datatypes.TM object at 0x...>
>>> timestamp_factory("12+0300", TM)
<hl7apy.base_datatypes.TM object at 0x...>
>>> timestamp_factory("1204", TM)
<hl7apy.base_datatypes.TM object at 0x...>
>>> timestamp_factory("120434", TM)
<hl7apy.base_datatypes.TM object at 0x...>
>>> timestamp_factory("120434-0400", TM)
<hl7apy.base_datatypes.TM object at 0x...>
```

If the value does not match one of the valid format it raises :exc:ValueError‘

Parameters

- **value** (`str`) – the value to assign the date object
- **value** – the *TM* class to use. It has to be one implementation of the different version modules
- **validation_level** (`int`) – It must be a value from class `validation_level` `VALIDATION_LEVEL` `hl7apy.consts.VALIDATION_LEVEL` or `None` to use the default value

Return type *TM*

hl7apy.factories.**datetime_factory**(*value*, *datatype_cls*, *validation_level=None*)
Creates a *hl7apy.base_datatypes.DTM* object

The value in input must be a string parsable with `datetime.strptime()`. It can also have an offset part specified with the format +HHMM -HHMM. The offset can be added with all the allowed format. The date format is chosen according to the length of the value as stated in this table:

Length	Format
4	`%Y`
6	`%Y%m`
8	`%Y%m%d`
10	`%Y%m%d%H`
12	`%Y%m%d%H%M`
14	`%Y%m%d%H%M%S`
18-21	`%Y%m%d%H%M%S.%f`

Some examples that work are:

```
>>> from hl7apy.base_datatypes import DTM
>>> datetime_factory("1924", DTM)
<hl7apy.base_datatypes.DTM object at 0x...>
>>> datetime_factory("1924+0300", DTM)
<hl7apy.base_datatypes.DTM object at 0x...>
>>> datetime_factory("19220430", DTM)
<hl7apy.base_datatypes.DTM object at 0x...>
>>> datetime_factory("19220430-0400", DTM)
<hl7apy.base_datatypes.DTM object at 0x...>
```

If the value does not match one of the valid format it raises `ValueError`

Parameters

- **value** (`str`) – the value to assign the date object
- **value** – the *DTM* class to use. It has to be one implementation of the different version modules
- **validation_level** (`int`) – It must be a value from class `validation_level VALIDATION_LEVEL hl7apy.consts.VALIDATION_LEVEL` or `None` to use the default value

Return type *DTM*

hl7apy.factories.**numeric_factory**(*value*, *datatype_cls*, *validation_level=None*)
Creates a *NM* object

The value in input can be a string representing a decimal number or a `float`. (i.e. a string valid for `decimal.Decimal()`). If it's not, a `ValueError` is raised Also an empty string or `None` are allowed

Parameters

- **value** (`str` or `None`) – the value to assign the numeric object
- **value** – the *NM* class to use. It has to be one implementation of the different version modules
- **validation_level** (`int`) – It must be a value from class `VALIDATION_LEVEL hl7apy.consts.VALIDATION_LEVEL` or `None` to use the default value

Return type *NM*

hl7apy.factories.**sequence_id_factory**(*value*, *datatype_cls*, *validation_level=None*)
Creates a *SI* object

The value in input can be a string representing an integer number or an int. (i.e. a string valid for int()). If it's not, a ValueError is raised Also an empty string or None are allowed

Parameters

- **value** (str or None) – the value to assign the date object
- **value** – the SI class to use. It has to be loaded from one implementation of the different version modules
- **validation_level** (int) – It must be a value from class VALIDATION_LEVEL hl7apy.consts.VALIDATION_LEVEL or None to use the default value

Return type *SI*

2.2.7 Exceptions

class hl7apy.exceptions.**HL7apyException**
Base exception class for hl7apy

class hl7apy.exceptions.**ParserError**
Error during parsing

```
>>> from hl7apy.parser import parse_message
>>> m = parse_message('NOTHL7')
Traceback (most recent call last):
...
ParserError: Invalid message
```

class hl7apy.exceptions.**UnsupportedVersion**(*version*)
Given version is not supported

```
>>> from hl7apy import set_default_version
>>> set_default_version("2.0")
Traceback (most recent call last):
...
UnsupportedVersion: The version 2.0 is not supported
```

class hl7apy.exceptions.**ChildNotFound**(*name*)
Raised when a child element is not found in the HL7 reference structures for the given version

```
>>> from hl7apy.core import Segment, Field
>>> s = Segment('MSH')
>>> s.unknown = Field()
Traceback (most recent call last):
...
ChildNotFound: No child named UNKNOWN
```

class hl7apy.exceptions.**ChildNotValid**(*child*, *parent*)
Raised when you try to assign an unexpected child to an *Element*

```
>>> from hl7apy.core import Segment, Field
>>> s = Segment('PID', validation_level=1)
>>> s.pid_1 = Field('PID_34')
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
ChildNotValid: <Field PID_34 (LAST_UPDATE_FACILITY) of type HD> is not a valid␣
↪child for PID_1
```

class hl7apy.exceptions.**UnknownValidationLevel**
Raised when the validation_level specified is not valid

It should be one of those defined in *VALIDATION_LEVEL*.

```
>>> from hl7apy import set_default_validation_level
>>> set_default_validation_level(3)
Traceback (most recent call last):
...
UnknownValidationLevel
```

class hl7apy.exceptions.**OperationNotAllowed**
Generic exception raised when something is not allowed

```
>>> from hl7apy.core import Segment
>>> s = Segment()
Traceback (most recent call last):
...
OperationNotAllowed: Cannot instantiate an unknown Segment
```

class hl7apy.exceptions.**MaxChildLimitReached**(*parent*, *child*, *limit*)
Raised when a child cannot be added to an instance of *Element* since the *Element* has already reached the maximum number of children allowed for the given child type (e.g. a *Message* should have at most 1 MSH segment)

```
>>> from hl7apy.core import Message, Segment
>>> m = Message("OML_O33", validation_level=1)
>>> m.add(Segment('MSH'))
Traceback (most recent call last):
...
MaxChildLimitReached: Cannot add <Segment MSH>: max limit (1) reached for
↪<Message OML_O33>
```

class hl7apy.exceptions.**MaxLengthReached**(*value*, *limit*)
Value length exceeds its datatype `max_length`.

```
>>> from hl7apy.v2_5 import get_base_datatypes
>>> from hl7apy.consts import VALIDATION_LEVEL
>>> SI = get_base_datatypes()['SI']
>>> st = SI(value=11111, validation_level=VALIDATION_LEVEL.STRICT)
Traceback (most recent call last):
...
MaxLengthReached: The value 11111 exceed the max length: 4
```

class hl7apy.exceptions.**InvalidName**(*cls*, *name*)
Raised if the reference for the given class/name has not been found

```
>>> from hl7apy.core import Message
>>> Message('Unknown')
Traceback (most recent call last):
...
InvalidName: Invalid name for Message: UNKNOWN
```

class hl7apy.exceptions.**InvalidDataType**(*datatype*)
Raised when the currently used HL7 version does not support the given datatype

```
>>> from hl7apy.factories import datatype_factory
>>> datatype_factory('TN', '11 123456', version="2.4")
<hl7apy.base_datatypes.TN object at 0x...>
>>> datatype_factory('GTS', '11 123456', version="2.4")
Traceback (most recent call last):
...
InvalidDataType: The datatype GTS is not available for the given HL7 version
```

class hl7apy.exceptions.**InvalidHighlightRange**(*lower_bound*, *upper_bound*)
Raised when the specified highlight range is not valid

For a description of highlight range see *hl7apy.base_datatypes.TextualDataType*

```
>>> from hl7apy.v2_5 import ST
>>> s = ST(value='some useful information', highlights=((5, 3),))
>>> s.to_er7()
Traceback (most recent call last):
...
InvalidHighlightRange: Invalid highlight range: 5 - 3
```

class hl7apy.exceptions.**InvalidDateFormat**(*out_format*)
Raised when the output format for a *hl7apy.base_datatypes.DateTimeDataType* is not valid

```
>>> from hl7apy.v2_5 import DTM
>>> DTM(value='10102013', out_format="%d%m%Y")
Traceback (most recent call last):
...
InvalidDateFormat: Invalid date format: %d%m%Y
```

class hl7apy.exceptions.**InvalidDateOffset**(*offset*)
Raised when the offset for a TM or *hl7apy.base_datatypes.DTM* is not valid

```
>>> from hl7apy.v2_5 import DTM
>>> DTM(value='20131010', out_format="%Y%m%d", offset='+1300')
Traceback (most recent call last):
...
InvalidDateOffset: Invalid date offset: +1300
```

class hl7apy.exceptions.**InvalidEncodingChars**
Raised when the encoding chars specified is not a correct set of HL7 encoding chars

```
>>> from hl7apy.core import Message
>>> encoding_chars = {'GROUP': '\r', 'SEGMENT': '\r', 'COMPONENT': '^',
↪                 'SUBCOMPONENT': '&', 'REPETITION': '~', 'ESCAPE': '\\'}
>>> m = Message('ADT_A01', encoding_chars=encoding_chars)
Traceback (most recent call last):
...
InvalidEncodingChars: Missing required encoding chars
```

2.2.8 Validation module

class hl7apy.validation.**Validator**(*level*)
Class that handles validation. It defines validation levels and validate an element using *VALIDATION.STRICT* validation level

static is_quiet(*level*)
Equal to *is_tolerant*. Kept for backward compatibility :param level: :rtype: bool :return: True if validation level is tolerant

static is_strict(*level*)
Check if the given validation level is strict

Parameters **level** (int) – validation level (see *VALIDATION_LEVEL*)

Return type bool

Returns True if validation level is strict

static is_tolerant(*level*)
Check if the given validation level is tolerant

Parameters **level** (int) – validation level (see *VALIDATION_LEVEL*)

Return type bool

Returns True if validation level is tolerant

static validate(*element*, *reference=None*, *report_file=None*)
Checks if the *Element* is a valid HL7 message according to the reference specified. If the reference is not specified, it will be used the official HL7 structures for the elements. In particular it checks:

- the maximum and minimum number of occurrences for every child
- that children are all allowed
- the datatype of fields, components and subcomponents
- the values, in particular the length and the adherence with the HL7 table, if one is specified

It raises the first exception that it finds.

If report_file is specified, it will create a file with all the errors that occur.

Parameters

- **element** – *Element*: The element to validate
- **reference** – the reference to use. Usually is None or a message profile object
- **report_file** – the name of the report file to create

Returns The True if everything is ok

Raises ValidationError: when errors occur

Raises ValidationWarning: errors concerning the values

2.2.9 MLLP Classes

class hl7apy.mllp.**MLLPServer**(*host*, *port*, *handlers*, *timeout=10*)
A TCPServer subclass that implements an MLLP server. It receives MLLP-encoded HL7 and redirects them to the correct handler, according to the handlers dictionary passed in.

The handlers dictionary is structured as follows. Every key represents a message type (i.e., the MSH.9) to handle, and the associated value is a tuple containing a subclass of *AbstractHandler* for that message type and additional arguments to pass to its constructor.

It is possible to specify a special handler for errors using the ERR key. In this case the handler should subclass *AbstractErrorHandler*, which receives, in addition to other parameters, the raised exception as the first argument. If the special handler is not specified the server will just close the connection.

The class allows to specify the timeout to wait before closing the connection.

Parameters

- **host** – the address of the listener
- **port** – the port of the listener
- **handlers** – the dictionary that specifies the handler classes for every kind of supported message.
- **timeout** – the timeout for the requests

class hl7apy.mllp.**AbstractHandler**(*message*)

Abstract transaction handler. Handlers should implement the *reply()* method which handle the incoming message. The incoming message is accessible using the attribute `incoming_message`

Parameters **message** – the ER7-formatted HL7 message to handle

reply()

Abstract method. It should implement the handling of the request message and return the response.

class hl7apy.mllp.**AbstractErrorHandler**(*exc*, *message*)

Abstract transaction handler for errors. It receives also the instance of the exception occurred, which will be accessible through the `exc` attribute. Specific exceptions that can be handled are `UnsupportedMessageType` and `InvalidHL7Message`

Parameters **exc** – the `Exception` occurred

2.2.10 Utility functions

hl7apy.utils.**check_date**(*value*)

Checks that the value is a valid HL7 date

Parameters **value** – the value to check

Returns *True* if the value is correct, *False* otherwise

hl7apy.utils.**check_datetime**(*value*)

Checks that the value is a valid HL7 datetime

Parameters **value** – the value to check

Returns *True* if the value is correct, *False* otherwise

hl7apy.utils.**check_timestamp**(*value*)

Checks that the value is a valid HL7 timestamp

Parameters **value** – the value to check

Returns *True* if the value is correct, *False* otherwise

hl7apy.utils.**get_date_info**(*value*)

Returns the datetime object and the format of the date in input

hl7apy.utils.**get_datetime_info**(*value*)

Returns the datetime object, the format, the offset and the microsecond of the datetime in input

hl7apy.utils.**get_timestamp_info**(*value*)

Returns the datetime object, the format, the offset and the microsecond of the timestamp in input

2.3 Examples

Basic usage examples can be found in the *Tutorial*. Advanced examples can be found in the `examples/` subdirectory of the HL7apy sources.

2.4 Release Notes

New in 1.3.2

- Fixed a bug with some fields structures

New in 1.3.1

- Fixed a bug with version 2.2 and 2.3

New in 1.3.0

- Implemented support to HL7 2.7, 2.8, 2.8.1 and 2.8.2 versions
- Changed structure of Message Profiles

New in 1.2.0

- Implemented support to Python 3.4, 3.5, 3.6

New in 1.1.0

- Implemented support to Z-Elements
- Implemented support to Message Profile
- Implemented a minimalistic MLLPServer

2.5 External Links

Michael Sarfati wrote an interesting 2-parts article on his blog about HL7apy with useful examples and tutorials. You can read it here. Thanks a lot Michael for your contribution.

CHAPTER 3

License

HL7apy is released under the MIT License (MIT)

```
Copyright (c) 2012-2018, CRS4

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Python Module Index

h

Index