python-hl7 Documentation

Release 0.3.0

John Paulett

August 18, 2014

Contents

python-hl7 is a simple library for parsing messages of Health Level 7 (HL7) version 2.x into Python objects. python-hl7 includes a simple client that can send HL7 messages to a Minimal Lower Level Protocol (MLLP) server (*mllp_send*).

HL7 is a communication protocol and message format for health care data. It is the de-facto standard for transmitting data between clinical information systems and between clinical devices. The version 2.x series, which is often is a pipe delimited format is currently the most widely accepted version of HL7 (there is an alternative XML-based format).

python-hl7 currently only parses HL7 version 2.x messages into an easy to access data structure. The library could eventually also contain the ability to create HL7 v2.x messages.

python-hl7 parses HL7 into a series of wrapped `hl7.Container` objects. The there are specific subclasses of `hl7.Container` depending on the part of the HL7 message. The `hl7.Container` message itself is a subclass of a Python list, thus we can easily access the HL7 message as an n-dimensional list. Specifically, the subclasses of `hl7.Container`, in order, are `hl7.Message`, `hl7.Segment`, `hl7.Field`, `hl7.Repetition`. and `hl7.Component`.

Warning: *0.3.0* breaks backwards compatibility by correcting the indexing of the MSH segment and the introducing improved parsing down to the repetition and sub-component level.

CHAPTER 1

Result Tree

HL7 Messages have a limited number of levels. The top level is a Message. A Message is comprised of a number of Fields (`hl7.Field`). Fields can repeat (`hl7.Repetition`). The content of a field is either a primitive data type (such as a string) or a composite data type comprised of one or more Components (`hl7.Component`). Components are in turn comprised of Sub-Components (primitive data types).

The result of parsing is accessed as a tree using python list conventions:

```
Message[segment][field][repetition][component][sub-component]
```

The result can also be accessed using HL7 1-based indexing conventions by treating each element as a callable:

```
Message(segment)(field)(repetition)(component)(sub-component)
```

CHAPTER 2

Usage

As an example, let's create a HL7 message:

```
>>> message = 'MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL-3456|P|2.4\r'
>>> message += 'PID|||555-44-4444||EVERYWOMAN^EVE^E^^^^L|JONES|196203520|F|||153 FERNWOOD DR.^^STATES
>>> message += 'OBR|1|845439^GHH OE|1045813^GHH LAB|1554-5^GLUCOSE|||200202150730|||||||555-55-5555
>>> message += 'OBX|1|SN|1554-5^GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl|70_105|H|||F'
```

We call the `hl7.parse()` command with string message:

```
>>> import hl7
>>> h = hl7.parse(message)
```

We get a `hl7.Message` object, wrapping a series of `hl7.Segment` objects:

```
>>> type(h)
<class 'hl7.Message'>
```

We can always get the HL7 message back:

```
>>> unicode(h) == message
True
```

Interestingly, `hl7.Message` can be accessed as a list:

```
>>> isinstance(h, list)
True
```

There were 4 segments (MSH, PID, OBR, OBX):

```
>>> len(h)
4
```

We can extract the `hl7.Segment` from the `hl7.Message` instance:

```
>>> h[3]
[[u'OBX'], [u'1'], [u'SN'], [[[u'1554-5'], [u'GLUCOSE'], [u'POST 12H CFST:MCNC:PT:SER/PLAS:QN']]], [u
>>> h[3] is h(4)
True
```

Note that since the first element of the segment is the segment name, segments are effectively 1-based in python as well (because the HL7 spec does not count the segment name as part of the segment itself):

```
>>> h[3][0]
[u'OBX']
>>> h[3][1]
```

```
[u'1']
>>> h[3][2]
[u'SN']
>>> h(4)(2)
[u'SN']
```

We can easily reconstitute this segment as HL7, using the appropriate separators:

```
>>> unicode(h[3])
u'OBX|1|SN|1554-5^GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl|70_105|H|||F'
```

We can extract individual elements of the message:

```
>>> h[3][3][0][1][0]
u'GLUCOSE'
>>> h[3][3][0][1][0] is h(4)(3)(1)(2)(1)
True
>>> h[3][5][0][1][0]
u'182'
>>> h[3][5][0][1][0] is h(4)(5)(1)(2)(1)
True
```

We can look up segments by the segment identifier, either via `hl7.Message.segments()` or via the traditional dictionary syntax:

```
>>> h.segments('OBX')[0][3][0][1][0]
u'GLUCOSE'
>>> h['OBX'][0][3][0][1][0]
u'GLUCOSE'
>>> h['OBX'][0][3][0][1][0] is h['OBX'](1)(3)(1)(2)(1)
True
```

Since many many types of segments only have a single instance in a message (e.g. PID or MSH), `hl7.Message.segment()` provides a convienance wrapper around `hl7.Message.segments()` that returns the first matching `hl7.Segment`:

```
>>> h.segment('PID')[3][0]
u'555-44-4444'
>>> h.segment('PID')[3][0] is h.segment('PID')(3)(1)
True
```

The result of parsing contains up to 5 levels. The last level is a non-container type.

```
>>> type(h)
<class 'hl7.Message'>

>>> type(h[3])
<class 'hl7.Segment'>

>>> type(h[3][3])
<class 'hl7.Field'>

>>> type(h[3][3][0])
<class 'hl7.Repetition'>

>>> type(h[3][3][0][1])
<class 'hl7.Component'>

>>> type(h[3][3][0][1][0])
<type 'unicode'>
```

The parser only generates the levels which are present in the message.

```
>>> type(h[3][1])
<class 'hl7.Field'>

>>> type(h[3][1][0])
<type 'unicode'>
```

MLLP network client - `mllp_send`

python-hl7 features a simple network client, `mllp_send`, which reads HL7 messages from a file or `sys.stdin` and posts them to an MLLP server. `mllp_send` is a command-line wrapper around `hl7.client.MLLPClient`. `mllp_send` is a useful tool for testing HL7 interfaces or resending logged messages:

```
mllp_send --file sample.hl7 --port 6661 mirth.example.com
```

See *mllp_send - MLLP network client* for examples and usage instructions.

For receiving HL7 messages using the Minimal Lower Level Protocol (MLLP), take a look at the related twisted-hl7 package. If do not want to use twisted and are looking to re-write some of twisted-hl7's functionality, please reach out to us. It is likely that some of the MLLP parsing and formatting can be moved into python-hl7, which twisted-hl7 and other libraries can depend upon.

Python 2 vs Python 3 and Unicode vs Byte strings

python-hl7 supports both Python 2.6+ and Python 3.3+. The library primarily deals in unicode (the `str` type in Python 3).

Passing a byte string to `hl7.parse()`, requires setting the `encoding` parameter, if using anything other than UTF-8. `hl7.parse()` will always return a datastructure containing unicode.

`hl7.Message` can be forced back into a string using `unicode(message)` in Python 2 and `str(message)` in Python 3.

mllp_send - MLLP network client assumes the stream is already in the correct encoding.

`hl7.client.MLLPClient`, if given a unicode string or `hl7.Message` instance, will use its `encoding` method to encode the unicode data to a byte string.

Contents

5.1 python-hl7 API

hl7.**parse**(*line*, *encoding=u'utf-8'*)

Returns a instance of the hl7.Message that allows indexed access to the data elements.

Note: HL7 usually contains only ASCII, but can use other character sets (HL7 Standards Document, Section 1.7.1), however as of v2.8, UTF-8 is the preferred character set [1].

python-hl7 works on Python unicode strings. `hl7.parse()` will accept unicode string or will attempt to convert bytestrings into unicode strings using the optional `encoding` parameter. `encoding` defaults to UTF-8, so no work is needed for bytestrings in UTF-8, but for other character sets like 'cp1252' or 'latin1', `encoding` must be set appropriately.

```
>>> h = hl7.parse(message)
```

To decode a non-UTF-8 byte string:

```
hl7.parse(message, encoding='latin1')
```

Return type hl7.Message

hl7.**ishl7**(*line*)

Determines whether a *line* looks like an HL7 message. This method only does a cursory check and does not fully validate the message.

Return type bool

hl7.**isfile**(*line*)

Files are wrapped in FHS / FTS FHS = file header segment FTS = file trailer segment

hl7.**split_file**(*hl7file*)

Given a file, split out the messages. Does not do any validation on the message. Throws away batch and file segments.

5.1.1 Data Types

class hl7.**Sequence**

Base class for sequences that can be indexed using 1-based index

[1] http://wiki.hl7.org/index.php?title=Character_Set_used_in_v2_messages

__call__(*index, value=<object object at 0x7f912644b620>*)
Support list access using HL7 compatible 1-based indices. Can be used to get and set values.

```
>>> s = hl7.Sequence([1, 2, 3, 4])
>>> s(1) == s[0]
True
>>> s(2, "new")
>>> s
[1, 'new', 3, 4]
```

class hl7.**Container**(*separator, sequence=[], esc=u'\', separators=u'r|~^&'*)
Abstract root class for the parts of the HL7 message.

__unicode__()
Join a the child containers into a single string, separated by the self.separator. This method acts recursively, calling the children's __unicode__ method. Thus `unicode()` is the approriate method for turning the python-hl7 representation of HL7 into a standard string.

```
>>> unicode(h) == message
True
```

Note: For Python 2.x use `unicode()`, but for Python 3.x, use `str()`

class hl7.**Accessor**

static __new__(*segment, segment_num=1, field_num=None, repeat_num=None, component_num=None, subcomponent_num=None*)
Create a new instance of Accessor for *segment*. Index numbers start from 1.

_asdict()
Return a new OrderedDict which maps field names to their values

classmethod _make(*iterable, new=<built-in method __new__ of type object at 0x90aa40>, len=<built-in function len>*)
Make a new Accessor object from a sequence or iterable

_replace(*_self, **kwds*)
Return a new Accessor object replacing specified fields with new values

component_num
Alias for field number 4

field_num
Alias for field number 2

key
Return the string accessor key that represents this instance

classmethod parse_key(*key*)
Create an Accessor by parsing an accessor key.

The key is defined as:

SEG[n]-Fn-Rn-Cn-Sn
F Field
R Repeat
C Component
S Sub-Component

Indexing is from 1 for compatibility with HL7 spec numbering.

Example:

PID.F1.R1.C2.S2 or PID.1.1.2.2

PID (default to first PID segment, counting from 1)
F1 (first after segment id, HL7 Spec numbering)
R1 (repeat counting from 1)
C2 (component 2 counting from 1)
S2 (component 2 counting from 1)

repeat_num
Alias for field number 3

segment
Alias for field number 0

segment_num
Alias for field number 1

subcomponent_num
Alias for field number 5

class `hl7.`**`Message`**(*separator*, *sequence=[ ]*, *esc=u'\'*, *separators=u'r|~^&'*)
Representation of an HL7 message. It contains a list of `hl7.Segment` instances.

`__getitem__`(*key*)
Index, segment-based or accessor lookup.

If key is an integer, `__getitem__` acts list a list, returning the `hl7.Segment` held at that index:

```
>>> h[1]
[[u'PID'], ...]
```

If the key is a string of length 3, `__getitem__` acts like a dictionary, returning all segments whose *segment_id* is *key* (alias of `hl7.Message.segments()`).

```
>>> h['OBX']
[[[u'OBX'], [u'1'], ...]]
```

If the key is a string of length greater than 3, the key is parsed into an `hl7.Accessor` and passed to `hl7.Message.extract_field()`.

If the key is an `hl7.Accessor`, it is passed to `hl7.Message.extract_field()`.

`__setitem__`(*key*, *value*)
Index or accessor assignment.

If key is an integer, `__setitem__` acts list a list, setting the `hl7.Segment` held at that index:

```
>>> h[1] = hl7.Segment("|", [hl7.Field("^", [u'PID'], [u''])])
```

If the key is a string of length greater than 3, the key is parsed into an `hl7.Accessor` and passed to `hl7.Message.assign_field()`.

```
>>> h["PID.2"] = "NEW"
```

If the key is an `hl7.Accessor`, it is passed to `hl7.Message.assign_field()`.

assign_field(*value*, *segment*, *segment_num=1*, *field_num=None*, *repeat_num=None*, *component_num=None*, *subcomponent_num=None*)

Assign a value into a message using the tree based assignment notation. The segment must exist.

Extract a field using a future proofed approach, based on rules in: http://wiki.medical-objects.com.au/index.php/Hl7v2_parsing

escape(*field*, *app_map=None*)

See: http://www.hl7standards.com/blog/2006/11/02/hl7-escape-sequences/

To process this correctly, the full set of separators (MSH.1/MSH.2) needs to be known.

Pass through the message. Replace recognised characters with their escaped version. Return an ascii encoded string.

Functionality:

- Replace separator characters (2.10.4)
- replace application defined characters (2.10.7)
- Replace non-ascii values with hex versions using HL7 conventions.

Incomplete:

- replace highlight characters (2.10.3)
- How to handle the rich text substitutions.
- Merge contiguous hex values

extract_field(*segment*, *segment_num=1*, *field_num=1*, *repeat_num=1*, *component_num=1*, *subcomponent_num=1*)

Extract a field using a future proofed approach, based on rules in: http://wiki.medical-objects.com.au/index.php/Hl7v2_parsing

'PID|Field1|Component1^Component2|Component1^Sub-Component1&Sub-Component2^Component3|Repeat1~Repeat2',

PID.F3.R1.C2.S2 = 'Sub-Component2'
PID.F4.R2.C1 = 'Repeat1'

Compatibility Rules:

If the parse tree is deeper than the specified path continue following the first child branch until a leaf of the tree is encountered and return that value (which could be blank).

Example:

PID.F3.R1.C2 = 'Sub-Component1' (assume .SC1)

If the parse tree terminates before the full path is satisfied check each of the subsequent paths and if every one is specified at position 1 then the leaf value reached can be returned as the result.

PID.F4.R1.C1.SC1 = 'Repeat1' (ignore .SC1)

segment(*segment_id*)

Gets the first segment with the *segment_id* from the parsed *message*.

```
>>> h.segment('PID')
[[u'PID'], ...]
```

Return type `hl7.Segment`

segments (*segment_id*)
Returns the requested segments from the parsed *message* that are identified by the *segment_id* (e.g. OBR, MSH, ORC, OBX).

```
>>> h.segments('OBX')
[[[u'OBX'], [u'1'], ...]]
```

Return type list of `hl7.Segment`

unescape (*field*, *app_map=None*)
See: http://www.hl7standards.com/blog/2006/11/02/hl7-escape-sequences/

To process this correctly, the full set of separators (MSH.1/MSH.2) needs to be known.

This will convert the identifiable sequences. If the application provides mapping, these are also used. Items which cannot be mapped are removed

For example, the App Map count provide N, H, Zxxx values

Chapter 2: Section 2.10

At the moment, this functionality can:

- replace the parsing characters (2.10.4)
- replace highlight characters (2.10.3)
- replace hex characters. (2.10.5)
- replace rich text characters (2.10.6)
- replace application defined characters (2.10.7)

It cannot:

- switch code pages / ISO IR character sets

class `hl7.`**Segment** (*separator*, *sequence=[ ]*, *esc=u'\'*, *separators=u'r|~^&'*)
Second level of an HL7 message, which represents an HL7 Segment. Traditionally this is a line of a message that ends with a carriage return and is separated by pipes. It contains a list of `hl7.Field` instances.

class `hl7.`**Field** (*separator*, *sequence=[ ]*, *esc=u'\'*, *separators=u'r|~^&'*)
Third level of an HL7 message, that traditionally is surrounded by pipes and separated by carets. It contains a list of strings or `hl7.Repetition` instances.

class `hl7.`**Repetition** (*separator*, *sequence=[ ]*, *esc=u'\'*, *separators=u'r|~^&'*)
Fourth level of an HL7 message. A field can repeat. It contains a list of strings or `hl7.Component` instances.

class `hl7.`**Component** (*separator*, *sequence=[ ]*, *esc=u'\'*, *separators=u'r|~^&'*)
Fifth level of an HL7 message. A component is a composite datatypes. It contains a list of string sub-components.

5.1.2 MLLP Network Client

class `hl7.client.`**MLLPClient** (*host*, *port*, *encoding='utf-8'*)
A basic, blocking, HL7 MLLP client based upon `socket`.

MLLPClient implements two methods for sending data to the server.

- `MLLPClient.send()` for raw data that already is wrapped in the appropriate MLLP container (e.g. *<SB>message<EB><CR>*).
- `MLLPClient.send_message()` will wrap the message in the MLLP container

Can be used by the `with` statement to ensure `MLLPClient.close()` is called:

```
with MLLPClient(host, port) as client:
    client.send_message('MSH|...')
```

MLLPClient takes an optional `encoding` parameter, defaults to UTF-8, for encoding unicode messages [2].

close()
Release the socket connection

send(*data*)
Low-level, direct access to the socket.send (data must be already wrapped in an MLLP container). Blocks until the server returns.

send_message(*message*)
Wraps a byte string, unicode string, or `hl7.Message` in a MLLP container and send the message to the server

If message is a byte string, we assume it is already encoded properly. If message is unicode or `hl7.Message`, it will be encoded according to `hl7.client.MLLPClient.encoding`

5.2 `mllp_send` - MLLP network client

python-hl7 features a simple network client, `mllp_send`, which reads HL7 messages from a file or `sys.stdin` and posts them to an MLLP server. `mllp_send` is a command-line wrapper around `hl7.client.MLLPClient`. `mllp_send` is a useful tool for testing HL7 interfaces or resending logged messages:

```
$ mllp_send --file sample.hl7 --port 6661 mirth.example.com
MSH|^~\&|LIS|Example|Hospital|Mirth|20111207105244||ACK^A01|A234244|P|2.3.1|
MSA|AA|234242|Message Received Successfully|
```

5.2.1 Usage

```
Usage: mllp_send [options] <server>

Options:
  -h, --help            show this help message and exit
  --version             print current version and exit
  -p PORT, --port=PORT  port to connect to
  -f FILE, --file=FILE  read from FILE instead of stdin
  -q, --quiet           do not print status messages to stdout
  --loose               allow file to be a HL7-like object (\r\n instead of
                        \r). Requires that messages start with "MSH|^~\&|".
                        Requires --file option (no stdin)
```

5.2.2 Input Format

By default, `mllp_send` expects the `FILE` or stdin input to be a properly formatted HL7 message (carriage returns separating segments) wrapped in a MLLP stream (`<SB>message1<EB><CR><SB>message2<EB><CR>...`).

However, it is common, especially if the file has been manually edited in certain text editors, that the ASCII control characters will be lost and the carriage returns will be replaced with the platform's default line endings. In this case,

[2] http://wiki.hl7.org/index.php?title=Character_Set_used_in_v2_messages

mllp_send provides the --loose option, which attempts to take something that "looks like HL7" and convert it into a proper HL7 message..

5.2.3 Additional Resources

- http://python-hl7.readthedocs.org

5.3 python-hl7 - Message Accessor

reproduced from: http://wiki.medical-objects.com.au/index.php/Hl7v2_parsing

Warning: Indexes in this API are from 1, not 0. This is to align with the HL7 documentation.

Example HL7 Fragment:

```
>>> message = 'MSH|^~\&|\r'
>>> message += 'PID|Field1|Component1^Component2|Component1^Sub-Component1&Sub-Component2^Component3

>>> import hl7
>>> h = hl7.parse(message)
```

The resulting parse tree with values in quotes:

Segment = "PID"
- F1
  - R1 = "Field1"
- F2
  - R1
    - C1 = "Component1"
    - C2 = "Component2"
- F3
  - R1
    - C1 = "Component1"
    - C2
      - S1 = "Sub-Component1"
      - S2 = "Sub-Component2"
    - C3 = "Component3"
- F4
  - R1 = "Repeat1"
  - R2 = "Repeat2"

Legend

F Field
R Repeat
C Component
S Sub-Component

A tree has leaf values and nodes. Only the leaves of the tree can have a value. All data items in the message will be in a leaf node.

After parsing, the data items in the message are in position in the parse tree, but they remain in their escaped form. To extract a value from the tree you start at the root of the Segment and specify the details of which field value you want to extract. The minimum specification is the field number and repeat number. If you are after a component or sub-component value you also have to specify these values.

If for instance if you want to read the value "Sub-Component2" from the example HL7 you need to specify: Field 3, Repeat 1, Component 2, Sub-Component 2 (PID.F1.R1.C2.S2). Reading values from a tree structure in this manner is the only safe way to read data from a message.

```
>>> h['PID.F1.R1']
u'Field1'

>>> h['PID.F2.R1.C1']
u'Component1'
```

You can also access values using `hl7.Accessor`, or by directly calling `hl7.Message.extract_field()`. The following are all equivalent:

```
>>> h['PID.F2.R1.C1']
u'Component1'

>>> h[hl7.Accessor('PID', 1, 2, 1, 1)]
u'Component1'

>>> h.extract_field('PID', 1, 2, 1, 1)
u'Component1'
```

All values should be accessed in this manner. Even if a field is marked as being non-repeating a repeat of "1" should be specified as later version messages could have a repeating value.

To enable backward and forward compatibility there are rules for reading values when the tree does not match the specification (eg PID.F1.R1.C2.S2) The common example of this is expanding a HL7 "IS" Value into a Codeded Value ("CE"). Systems reading a "IS" value would read the Identifier field of a message with a "CE" value and systems expecting a "CE" value would see a Coded Value with only the identifier specified. A common Australian example of this is the OBX Units field, which was an "IS" value previously and became a "CE" Value in later versions.

> Old Version: "|mmol/l|" New Version: "|mmol/l^^ISO+|"

Systems expecting a simple "IS" value would read "OBX.F6.R1" and this would yield a value in the tree for an old message but with a message with a Coded Value that tree node would not have a value, but would have 3 child Components with the "mmol/l" value in the first subcomponent. To resolve this issue where the tree is deeper than the specified path the first node of every child node is traversed until a leaf node is found and that value is returned.

```
>>> h['PID.F3.R1.C2']
u'Sub-Component1'
```

This is a general rule for reading values: **If the parse tree is deeper than the specified path continue following the first child branch until a leaf of the tree is encountered and return that value (which could be blank).**

Systems expecting a Coded Value ("CE"), but reading a message with a simple "IS" value in it have the opposite problem. They have a deeper specification but have reached a leaf node and cannot follow the path any further. Reading a "CE" value requires multiple reads for each sub-component but for the "Identifier" in this example the specification would be "OBX.F6.R1.C1". The tree would stop at R1 so C1 would not exist. In this case the unsatisfied path elements (C1 in this case) can be examined and if every one is position 1 then they can be ignored and the leaf of the tree that was reached returned. If any of the unsatisfied paths are not in position 1 then this cannot be done and the result is a blank string.

This is the second Rule for reading values: **If the parse tree terminates before the full path is satisfied check each of the subsequent paths and if every one is specified at position 1 then the leaf value reached can be returned as the result.**

```
>>> h['PID.F1.R1.C1.S1']
u'Field1'
```

This is a general rule for reading values: **If the parse tree is deeper than the specified path continue following the first child branch until a leaf of the tree is encountered and return that value (which could be blank).**

In the second example every value that makes up the Coded Value, other than the identifier has a component position greater than one and when reading a message with a simple "IS" value in it, every value other than the identifier would return a blank string.

Following these rules will result in excellent backward and forward compatibility. It is important to allow the reading of values that do not exist in the parse tree by simply returning a blank string. The two rules detailed above, along with the full tree specification for all values being read from a message will eliminate many of the errors seen when handling earlier and later message versions.

```
>>> h['PID.F10.R1']
u''
```

At this point the desired value has either been located, or is absent, in which case a blank string is returned.

5.3.1 Assignments

The accessors also support item assignments. However, the Message object must exist and the separators must be validly assigned.

Create a response message.

```
>>> SEP = '|^~\&'
>>> CR_SEP = '\r'
>>> MSH = hl7.Segment(SEP[0], [hl7.Field(SEP[1], ['MSH'])])
>>> MSA = hl7.Segment(SEP[0], [hl7.Field(SEP[1], ['MSA'])])
>>> response = hl7.Message(CR_SEP, [MSH, MSA])
>>> response['MSH.F1.R1'] = SEP[0]
>>> response['MSH.F2.R1'] = SEP[1:]

>>> unicode(response)
u'MSH|^~\\&|\rMSA'
```

Assign values into the message. You can only assign a string into the message (i.e. a leaf of the tree).

```
>>> response['MSH.F9.R1.C1'] = 'ORU'
>>> response['MSH.F9.R1.C2'] = 'R01'
>>> response['MSH.F9.R1.C3'] = ''
>>> response['MSH.F12.R1'] = '2.4'
>>> response['MSA.F1.R1'] = 'AA'
>>> response['MSA.F3.R1'] = 'Application Message'

>>> unicode(response)
u'MSH|^~\\&|||||||ORU^R01^|||2.4\rMSA|AA||Application Message'
```

You can also assign values using `hl7.Accessor`, or by directly calling `hl7.Message.assign_field()`. The following are all equivalent:

```
>>> response['MSA.F1.R1'] = 'AA'
>>> response[hl7.Accessor('MSA', 1, 1, 1)] = 'AA'
>>> response.assign_field('AA', 'MSA', 1, 1, 1)
```

5.3.2 Escaping Content

HL7 messages are transported using the 7bit ascii character set. Only characters between ascii 32 and 127 are used. Characters which cannot be transported using this range of values must be 'escaped', that is replaced by a sequence of characters for transmission.

The stores values internally in the escaped format. When the message is composed using 'unicode', the escaped value must be returned.

```
>>> message = 'MSH|^~\&|\r'
>>> message += 'PID|Field1|\F\|\r\r'
>>> h = hl7.parse(message)

>>> unicode(h['PID'][0][2])
u'\\F\\'

>>> h.unescape(unicode(h['PID'][0][2]))
u'|'
```

When the accessor is used to reference the field, the field is automatically unescaped.

```
>>> h['PID.F2.R1']
u'|'
```

The escape/unescape mechanism support replacing separator characters with their escaped version and replacing non-ascii characters with hexadecimal versions.

The escape method returns a 'str' object. The unescape method returns a unicode object.

```
>>> h.unescape('\\F\\')
u'|'

>>> h.unescape('\\R\\')
u'~'

>>> h.unescape('\\S\\')
u'^'

>>> h.unescape('\\T\\')
u'&'

>>> h.unescape('\\X202020\\')
u'   '

>>> h.escape('|~^&')
u'\\F\\\\R\\\\S\\\\T\\'

>>> h.escape('áéíóú')
u'\\Xc3\\\\Xa1\\\\Xc3\\\\Xa9\\\\Xc3\\\\Xad\\\\Xc3\\\\Xb3\\\\Xc3\\\\Xba\\'
```

Presentation Characters

HL7 defines a protocol for encoding presentation characters, These include hightlighting, and rich text functionality. The API does not currently allow for easy access to the escape/unescape logic. You must overwrite the message class escape and unescape methods, after parsing the message.

5.4 Contributing

The source code is available at http://github.com/johnpaulett/python-hl7

Please fork and issue pull requests. Generally any changes, bug fixes, or new features should be accompanied by corresponding tests in our test suite.

5.4.1 Testing

The test suite is located in `tests/` and can be run several ways.

It is recommended to run the full tox suite so that all supported Python versions are tested and the documentation is built and tested. We provide a `Makefile` to create a virtualenv, install tox, and run tox:

```
$ make tests
  py27: commands succeeded
  py26: commands succeeded
  docs: commands succeeded
  congratulations :)
```

To run the test suite with a specific python interpreter:

```
python setup.py test
```

To documentation is built by tox, but you can manually build via:

```
$ make docs
...
Doctest summary
===============
   23 tests
    0 failures in tests
    0 failures in setup code
...
```

It is also recommended to run the flake8 checks for PEP8 and PyFlake violations. Commits should be free of warnings:

```
$ make lint
```

5.5 Change Log

5.5.1 0.3.0 - 2014-08-18

> **Warning:** *0.3.0* breaks backwards compatibility by correcting the indexing of the MSH segment and the introducing improved parsing down to the repetition and sub-component level.

- Changed the numbering of fields in the MSH segment. **This breaks older code.**
- Parse all the elements of the message (i.e. down to sub-component). **The inclusion of repetitions will break older code.**
- Implemented a basic escaping mechanism

- New constant 'NULL' which maps to '""'
- New `hl7.isfile()` and `hl7.split_file()` functions to identify file (FHS/FTS) wrapped messages
- New mechanism to address message parts via a *symbolic accessor name*
- Message (and Message.segments), Field, Repetition and Component can be accessed using 1-based indices by using them as a callable.
- Added Python 3 support. Python 2.6, 2.7, and 3.3 are officially supported.
- `hl7.parse()` can now decode byte strings, using the `encoding` parameter. `hl7.client.MLLPClient` can now encode unicode input using the `encoding` parameter. To support Python 3, unicode is now the primary string type used inside the library. bytestrings are only allowed at the edge of the library now, with `hl7.parse` and sending via `hl7.client.MLLPClient`. Refer to *Python 2 vs Python 3 and Unicode vs Byte strings*.
- Testing via tox and travis CI added. See *Contributing*.

A massive thanks to Kevin Gill and Emilien Klein for the initial code submissions to add the improved parsing, and to Andrew Wason for rebasing the initial pull request and providing assistance in the transition.

5.5.2 0.2.5 - 2012-03-14

- Do not senselessly try to convert to unicode in mllp_send. Allows files to contain other encodings.

5.5.3 0.2.4 - 2012-02-21

- `mllp_send --version` prints version number
- `mllp_send --loose` algorithm modified to allow multiple messages per file. The algorithm now splits messages based upon the presumed start of a message, which must start with `MSH|^~\&|`

5.5.4 0.2.3 - 2012-01-17

- `mllp_send --loose` accepts & converts Unix newlines in addition to Windows newlines

5.5.5 0.2.2 - 2011-12-17

- *mllp_send* now takes the `--loose` options, which allows sending HL7 messages that may not exactly meet the standard (Windows newlines separating segments instead of carriage returns).

5.5.6 0.2.1 - 2011-08-30

- Added MLLP client (`hl7.client.MLLPClient`) and command line tool, *mllp_send*.

5.5.7 0.2.0 - 2011-06-12

- Converted `hl7.segment` and `hl7.segments` into methods on `hl7.Message`.
- Support dict-syntax for getting Segments from a Message (e.g. `message['OBX']`)
- Use unicode throughout python-hl7 since the HL7 spec allows non-ASCII characters. It is up to the caller of `hl7.parse()` to convert non-ASCII messages into unicode.

- Refactored from single hl7.py file into the hl7 module.
- Added Sphinx documentation. Moved project to github.

5.5.8 0.1.1 - 2009-06-27

- Apply Python 3 trove classifier

5.5.9 0.1.0 - 2009-03-13

- Support message-defined separation characters
- Message, Segment, Field classes

5.5.10 0.0.3 - 2009-01-09

- Initial release

5.6 Authors

- John Paulett (john -at- paulett.org)
- Andrew Wason
- Kevin Gill
- Emilien Klein

5.7 License

Install

python-hl7 is available on PyPi via `pip` or `easy_install`:

```
pip install -U hl7
```

For recent versions of Debian and Ubuntu, the *python-hl7* package is available:

```
sudo apt-get install python-hl7
```

CHAPTER 7

Links

- Documentation: http://python-hl7.readthedocs.org
- Source Code: http://github.com/johnpaulett/python-hl7
- PyPi: http://pypi.python.org/pypi/hl7

HL7 References:

- Health Level 7 - Wikipedia
- nule.org's Introduction to HL7
- hl7.org
- OpenMRS's HL7 documentation
- Transport Specification: MLLP
- HL7v2 Parsing
- HL7 Book