

Bots Manual

Bots open source edi translator

BOTS version 2.0

Date: 2010-06-22

Copyright by Henk-Jan Ebberts (GPL licence)

Bots is made possible by EbbertsConsult (www.EbbertsConsult.com)

Table of Contents

1.Introduction.....	3
2.Configuration.....	4
3.Routes.....	5
3.1.Attributes in a route.....	5
3.2.Using more than one outchannel incoming edi files (different destinations).....	6
3.3.Using partner-specific tochannels in a route.....	7
3.4.Advanced: routing scripts.....	7
4.Channels.....	9
4.1.Configure a channel: Bots-monitor->Configuration->Channels.....	9
4.2.Communication scripts.....	9
4.3.Writing/reading directly from database/database connector.....	10
4.4.Partners and email details.....	10
5.Translations.....	11
5.1.Translation recepies.....	11
5.2.What attributes determine a translation?.....	13
5.3.How does a translation work?.....	13
6.Mapping scripts.....	14
6.1.Functions in a mapping script.....	14
6.2.Mpath.....	18
6.3.How does a mapping work.....	18
6.4.Loose ends.....	19
7.Message grammars.....	20
7.1.Syntax parameters.....	20
7.2.Structure: sequence of records in a message.....	22
3.5.Recorddefs: definition of records; describes the fields in each record.....	23
3.6.nextmessage: split an EDI file in separate messages.....	24
3.7.nextmessageblock: split an CSV file into messages.....	24
8.User maintained code lists.....	25
9.Technical overview of bots.....	26

1. Introduction

Most information is nowadays on the bots web site: <http://bots.sourceforge.net>.

This manual contains detailed information about the configuration of bots and is more like a reference manual.

Some screenshots are of the 1.6 version; information in new version is not very different.

2. Configuration.

You have to configure Bots in order to do something useful for you. Bots is highly configurable and adaptable to your wishes.

So, how to configure Bots? There are more ways to do this:

1. Use plugins.
This is a quick and easy way to install predefined configurations of Bots. Plugins can be shared by users and communities. Download plugins from <http://bots.sourceforge.net>
See website <http://bots.sourceforge.net/en/plugins.shtml>
2. Use a plugins 'close' to want you want, and adapt.
Understanding Bots configuration is best done this way.
3. Do your own configuration.
Use bots-monitor for configuration. Advanced configuration (routes, grammars, mapping scripts) requires knowledge of EDI, EDI standards and data mapping. A lot of chapters in this manual deal with configuration. A few tips:
 - There is a tutorial on the Bots website.
 - There are plugins available on our website. These provide good examples. They provide good grammars for standard messages, etc.
 - There are grammars for all edifact and X12 messages on the bots web site.
 - Check a grammar using the command line tool 'bots-grammar'.
 - Please, please use an editor with 'syntax highlighting'. All configuration files are written in python and syntax highlighting is much easier.
 - Use python to 'compile' a configuration file; this catches most of your errors. This is often easiest from within an editor. A very good open source editor that does all this and more is 'scite' (<http://www.scintilla.org/SciTE.html>).
 - Share your plugins etc. with other people. It is easy to make a plugin. We will post them on our web site.
 - Share your experiences with us. Bots is constantly being improved, and we value your input.
4. Hire us (www.EbbersConsult.com - the makers of Bots) to help you with configuring Bots: we are the EDI experts.

3. Routes

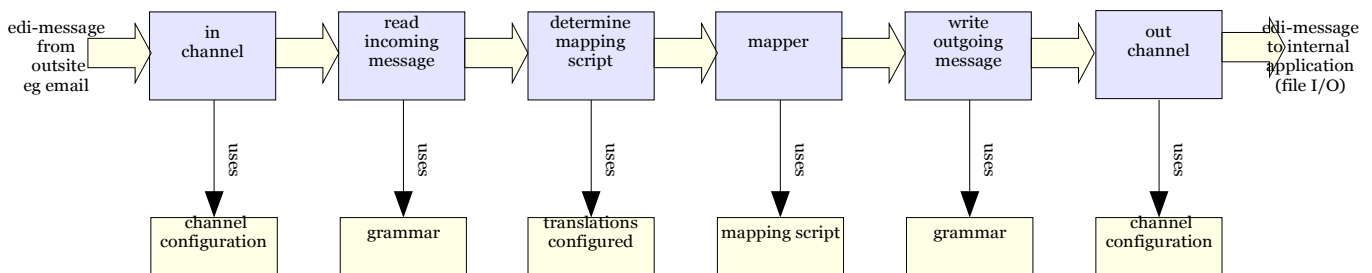
'Routes' are the most important concept in configuring Bots. All actions Bots performs are controlled by routes: no route no action. If you run bots-engine (without parameters), it processes all active routes.

Routes are independent; a EDI message in one route is not used by other route.

Configure a route in: *bots-monitor->Configuration->Routes*.

On the bots website you can download plugin 'my_first_plugin'. Documentation about this plugin: <http://bots.sourceforge.net/en/myfirstplugin.shtml>

This picture show what happens in a route:



In this picture the route controls:

- gets the EDI messages from the in-channel.
- sets the editype and messagetype.
- starts translation, using editype and messagetype to determine the mapping script.
- writes the translated message.
- sends the translated EDI messages via the out-channel.

So to get a route working, at least the following must be configured:

- the route itself (in bots-monitor)
- a in-channel (in bots-monitor)
- a out-channel (in bots-monitor)
- a translation (in bots-monitor)
- the mapping script (script, py-file)
- the grammar for the incoming message (script, py-file)
- the grammar for the outgoing message (script, py-file)

3.1. Attributes in a route

3.1.1. Active

Only active routes are used when bots-engine runs.

3.1.2. Idroute

3.1.3. Seq

(advanced) for composite routes; determines the sequence for running routes with the same Idroute.

3.1.4. Fromchannel.

The communication channel Bots uses to receive EDI messages, e.g. FTP, infile, POP3.

3.1.5. Fromeditype and frommessagetype.

The editype and messagetype of the incoming EDI message.

Bots uses this to determine the right translation/mapping script.

Note: normally one route only has one incoming editype and messagetype.

3.1.6. Alt

(advanced) extra attribute used to determine the translation.

3.1.7. Translate

(advanced) For composite routes: indicates if a translation is done in this partial route. A normal route always does a translation.

3.1.8. Tochannel

Communication channel used as destination for outbound EDI messages, e.g. FTP, outfile, SMTP.

3.1.9. Defer

Set files ready for this communication channel, but communicate these in a route that is run later (and where communication is not deferred. This way all communication for a certain outchannel can be done in one 'session'.

3.1.10. TestIndicator

Use only test edi files/production files/both for the outchannel.

3.1.11. Toeditype and Tomessagetype

Use only this editype/messagetype for the outchannel.

3.1.12. Frompartner_tochannel and Topartner_tochannel

Use only edi files of this frompartner/topartner for the outchannel.

3.2. Using more than one outchannel incoming edi files (different destinations)

Use field 'seq' (sequence number). Several routes can have the same routeID. Bots runs first the route with the lowest seq, then the next lowest seq, etc. This way, you can use different tochannels for different editypes/messagetypes/frompartners/topartners/testindicators. (basically this works as a filter).

Example: receive edifact messages DESADV and INVOIC from one inchannel.

Set it up like this:

Route-id	Seq	fromchannel	fromedi-type	frommessage-type	translate	tochannel	toeditype	tomessage-type
Myroute	1	Mypop3	edifact	edifact	yes			
Myroute	2				no	Desadv	Fixed	Desadv
Myroute	3				no	Invoic	Fixed	Invoice

Bots takes the following actions with these routes:

1. The route with seq=1 runs first. It receives edifact messages (desadv and invoices) from channel 'Mypop3' and translates these messages.
This route has no output.
2. The route with seq=2 runs. It takes the translated fixed desadv messages and outputs these to channel 'Desadv-file'.
3. Then, the route with seq=3 runs. It takes the translated fixed invoice messages and outputs these to channel 'Invoic-file'.

3.3. Using partner-specific tochannels in a route

Set it up like this:

Route-id	Seq	fromchannel	fromedi-type	frommessage-type	translate	tochannel	topartner
Myroute	1	Mypop3	fixed	orders	yes		
Myroute	2				no	ftppartnerX	partnerX
Myroute	3				no	std-email	

In the above example, messages (orders in fixed file format) are collected from the in channel 'Mypop3' (seq = 1). Then (seq = 2), all messages for 'partnerX' are sent to that partner's FTP channel, 'ftppartnerX'. Finally (seq = 3), all messages for partners other than partnerX are sent to the 'std-email' channel.

A partner can belong to a partner groups. So you can setup a specific outchannel for all partner belonging to a partner group.

3.4. Advanced: routing scripts

When the standard routing of Bots does not fit your needs, use routing scripts.

Routing scripts are python programs. In a routing script, you can program/instruct Bots to run a route.

Details

1. use bots-monitor to add a new route; just enter a routeID and seq
2. make a routing script with the same name as the routeID
3. place the routing script in *bots/usersys/routescripts/<routeid>.py*

You can use 1 route script to run the whole route (your script contains a `main()` function that is called); but you can also use access point for your own route, eg after in-communication, before translation etc.

3.4.1. Functions provided by Bots for use in a routing script

1. `communication.router(route, fromchannel, tochannel, editype, messagetype)`
 - gets messages from 'fromchannel', translates them from editype/messagetype, and sends the results to 'tochannel'.

- `router()` is easy to use, but has limitations:
 1. all input for 'fromchannel' has to be of the same editype/messagetype
 2. all output goes to the same tochannel.
- to set up in order to get this command to work:
 1. setup the specified 'fromchannel'
 2. if 'fromchannel' is e.g. POP3, Bots has to know the right emailaddresses/partners;
 3. set up to right translations (in db-translate), message grammar(s) and mapping script(s).
 4. setup the specified 'outchannel'
 5. if 'outchannel' is e.g. SMTP: Bots has to know right emailaddresses/partners;
- example of usage:


```
communication.router(route='myroute',fromchannel='mypop3',tochannel='myoutfile',
                     editype='edifact',messagetype='edifact')
```
- 2. `communication.run(channel, route)`
 - does a communication session with the specified channel
 - for both in-communication and out-communication
 - to set up in order to get this command to work:
 1. setup the specified 'channel' in bots-monitor
 2. if 'channel' is e.g. POP3 or SMTP, Bots has to know the right emailaddresses/partners
 - example of usage:


```
communication.run(route='myroute',channel='mypop3')
```
- 3. `transform.addinfo(change, where)`
 - change/add information for an EDI file.
 - 'where' specifies what EDI files to change/update; 'where' is a dict.
 - 'change' specifies what information to set for the selected EDI files; 'set' is a dict.
 - `addinfo()` resembles an SQL UPDATE query
 - routes are specified in 'where' argument (a dictionary)
 - example of usage (set translated and merged invoices ready for communication via SMTP):


```
transform.addinfo(change={'tochannel':'mysmtp','status':FILEOUT},
                  where={'route':'myroute','status':MERGED,'messagetype':'INVOICD96AUNEAN008'})
```
- 4. `transform.translate(route)`
 - translates all EDI files with status TRANSLATE in the specified route
 - to get this command to work:
 1. set up right translations in bots-monitor
 2. provide message grammar(s)
 3. provide mapping script(s)
 - example of usage:


```
transform.translate(route='myroute')
```
- 5. `transform.mergemessages(route)`
 - merges all translated EDI messages for the specified route into one envelope.
 - merge information is coming from the outmessage grammar for each translation.

- example of usage:
transform.mergemessages(route='myroute')

4. Channels

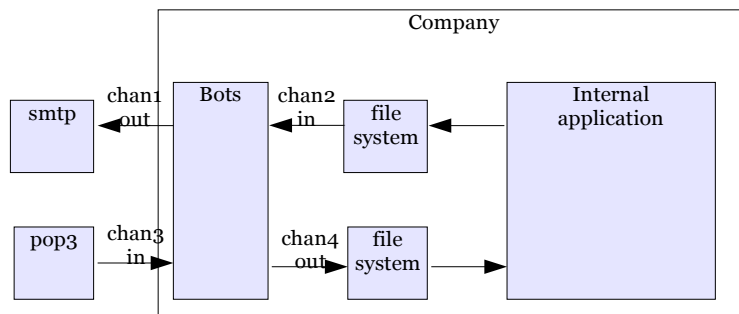
4.1. Configure a channel: *Bots-monitor->Configuration->Channels*.

Examples of channels:

- receive email from a POP3-mailbox at an EDI network provider (incoming)
- send email to a SMTP-mailbox at an EDI network provider (outgoing)
- pick up in-house invoices from a directory on your computer (incoming)
- put the translated orders in a file queue. Import these orders in your application (outgoing).

A channel can be for external communications (e.g. POP3, SMTP) or communications from/to the file system.

Scheme of communication channels for typical Bots use:



Remark: 'incoming' means 'incoming to Bots'; 'outgoing' means 'going out Bots'.

One channel is either for incoming or for outgoing EDI messages, never for both. If the same FTP server account is used both for incoming and outgoing EDI messages, set up 2 channels in Bots.

Note: each 'type' of channel (e.g. POP3, FTP) uses different parameters. E.g.:

- POP3 needs a host, port, user name, password, etc.
- file communication needs a path and a file name.

Note: Bots only communicates as a 'client'. If you want to use a server for communication (FTP, AS2), use a separate server. We are unlikely to build this into Bots; there are other (open source) packages that provide this.

4.2. Communication scripts

Bots channels have a communication type 'communication script'.

This make it possible to write your own communication script.

Communication script should have the same name as the channelID.

Communication script should be in `bots/usersys/communicationscripts`

Use cases:

- communication methode not provided by Bots
- additional requirements eg use partner name in output file.
- call external program to write edi message to your ERP system.

4.3. Writing/reading directly from database/database connector

Bots channels have a communication type 'database' .

This make it possible to write your own database script.

database script should have the same name as the channelID.

database script should be in *bots/usersys/dbconnectors*

4.4. Partners and email details

For email channels, you need to configure partners with their email addresses.

Bots uses this:

1. for incoming messages, to determine whether an email is from a valid sender. Email from unknown partners are errors (think of spam).
2. for outgoing message, to determine the destination address.

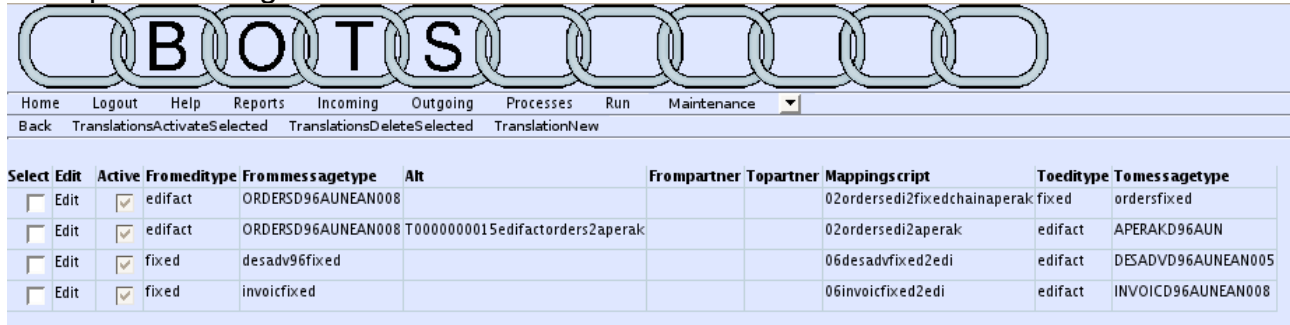
Configuring a partner for email:

1. Configure a partner: *Bots-monitor->Configuration->Partners*.
2. One partner can have a different email address per channel; to do this use email details of partner: *Bots-monitor->Configuration->Email-details*.

5. Translations

A translation determines what message-format to convert to what other message-format. Configure a translation: *Bots-monitor->Configuration->Translations*.

Examples of configured translations in bots-monitor:



Select	Edit	Active	From editype	From messagetype	Alt	From partner	To partner	Mappingscript	To editype	To messagetype
<input type="checkbox"/>	Edit	<input checked="" type="checkbox"/>	edifact	ORDERSD96AUNEAN008				02ordersedi2fixedchainaperak	fixed	ordersfixed
<input type="checkbox"/>	Edit	<input checked="" type="checkbox"/>	edifact	ORDERSD96AUNEAN008	T0000000015edifactorders2aperak			02ordersedi2aperak	edifact	APERAKD96AUN
<input type="checkbox"/>	Edit	<input checked="" type="checkbox"/>	fixed	desadv96fixed				06desadvfixed2edi	edifact	DESADV96AUNEAN005
<input type="checkbox"/>	Edit	<input checked="" type="checkbox"/>	fixed	invoicfixed				06invoicfixed2edi	edifact	INVOICD96AUNEAN008

Route 1 reads as:

translate edifact/ORDERSD96AUNEAN008 to fixed/myinhouseorder using mapping script ordersedifact2myinhouse.py

To do a translation, bots-engine has to know the editype and messagetype of the EDI message. This is configured in the route.

After reading the EDI message, bots-engine looks in the translation table to find the right translation. If found, bots-engine converts the EDI message using the right mapping script, to editype and to messagetype of the translation.

In order to set up a complete translation you need:

1. Translations as shown in the screen shot above.
2. Message grammar for 'from'-message
3. Message grammar for 'to'-message
4. Mapping script(s) for converting from-message to to-message.

5.1. Translation recepies

1. Edi-types with standard envelopes (e.g. edifact).
Example in plugin dutchic_d96a.zip:
 - a grammar for the envelope (with nextmessage and SUBTRANSLATION).
`usersys/grammars/edifact/edifact3.py`
 - each messagetype has its own grammar:
`usersys/grammars/edifact/DESADV96AUNEAN008.py`
`usersys/grammars/edifact/INVOICD96AUNEAN008.py`
 - add translations in db-translate: e.g. (both for envelope and messagetypes).
2. Partner-dependent translation.
E.g. you receive *edifact/ORDERSD96AUNEAN008* for several partners. Partner 'retailer-abroad' uses the message in a different way.
 - one grammar for message:
`usersys/grammars/edifact/ORDERSD96AUNEAN008.py`
 - 2 translations in translate-db; one for this specific partner; one for the others:
translate edifact/ORDERSD96AUNEAN008 to fixed/myinhouseorder with mapping script ordersedifact2myinhouse.py

- *translate edifact/ORDERSD96AUNEAN008 to fixed/myinhouseorder*
with mapping script ordersedifactabroad2myinhouse.py for partner retailer-abroad
- Bots uses QUERIES in grammar to fetch the partner-id in the message.
Note: A partner can belong to a partner groups. So you can have a partner-dependent translation for all partners in a partnergroup.
- 3. Different translations for one editype/messagetype from different routes
 - you should have one grammar for message:
 - *usersys/grammars/edifact/ORDERSD96AUNEAN008.py*
 - add 2 translations in translate-db:
 - *translate edifact/ORDERSD96AUNEAN008 to fixed/myinhouseorder*
with mapping script ordersedifact2myinhouse.py
 - *translate edifact/ORDERSD96AUNEAN008 to fixed/myinhouseorder*
with mapping script ordersedifactmyinhousealt.py for alt 'different order'
 - indicate 'alt' in route for last translation.
- 4. Translate inhouse-messages to more than one version of a editype/messagetype.
Information about the version is in inhouse-message.
 - one grammar for inhouse message:
 - *usersys/grammars/fixed/myinhouseorder.py*
 - 2 translations
 - *translate fixed/ORDERSD963AUNEAN007 to edifact/ ORDERSD93AUNEAN007*
with mapping script ordersfixed2edifact93.py
 - *translate fixed/ORDERSD96AUNEAN008 to edifact/ ORDERSD96AUNEAN008*
with mapping script ordersfixed2edifact96.py for alt 'other translation'
 - in grammar myinhouseorder.py: use QUERIES to extract 'alt'.
- 5. Translate message and send a confirmation back
("chained translation": 1 message in , 2 (or more) messages out).
Example plugin:
Here the first translation returns 'alt'; this is used to determine the 2nd (chained) translation, etc.
Incoming: edifact/ORDERSD96AUNEAN008.
Outgoing: fixed/myinhouseorder + edifact/APERAKD96AUN. (send APERAK back to sender).
 - 3 grammars:
 - usersys/grammars/edifact/ORDERSD96AUNEAN008.py*
 - usersys/grammars/fixed/myinhouseorder.py*
 - usersys/grammars/edifact/APERAKD96AUN.py*
 - 2 mapping scripts:
 - usersys/scripts/ordersedifact2myinhouse.py*
 - usersys/scripts/ordersedifact2aperak.py*
 - If mapping script ordersedifact2aperak returns 'altaperak' (==alt-value in aperak translation) the translation where alt='altaperak' translation is done.
 - add 2 translations to db-translates:
 - translate edifact/ORDERSD96AUNEAN008 to fixed/myinhouseorder*
with mapping script ordersedifact2myinhouse.py
 - translate edifact/ORDERSD96AUNEAN008 to edifact/APERAKD96AUN*
with mapping script ordersedifact2aperak.py; alt is 'altaperak'
- 6. Partner-dependent syntax.
Sometimes a partner wants or forces you to use a special syntax; e.g. an older version of a standard, certain separators etc.
This is especially true for X12.
This can easy be achieved by placing a partner-dependent syntax in
usersys/partners/<editype>/<parttnerid>.py.

This is like the syntax section of a grammar file; see documentation on grammars. The values in the partner dependent syntax overrule the default values (for this partner).

5.2. What attributes determine a translation?

Bots uses 5 attributes to find the correct translation:

1. fromeditype. Normally you will set this in a route
2. frommessagetype. Normally you will set this in a route
3. alt. Set this:
 - in a route
 - by using QUERIES in the grammar
 - returned by another mapping script
4. frompartner. Set this:
 - by using QUERIES in the grammar
 - from the email address
5. topartner
 - by using QUERIES in the grammar
 - from the email address

5.3. How does a translation work?

When a EDI file is read into the translator, Bots has to know the editype and messagetype in advance. This is indicated in the route.

Editype/messagetype is used to lex and parse the EDI file.

(If a editype has standardised envelopes (edifact, X12), it is enough to tell that the editype is e.g. 'edifact' and the messagetype is 'edifact'; bots figures out the right messagetype of the messages in the envelope itself.)

Next Bots determines the mapping script to use and the target editype/messagetype for the messages. To determine this Bots looks in 'translations'. Bots uses 5 attributes to find the correct translation:

- fromeditype
- frommessagetype
- alt
- frompartner
- topartner

Bots finds the most specific translation. Example situation: 2 translations:

- fromeditype: edifact, frommessagetype: 850004010
- fromeditype: edifact, frommessagetype: 850004010, frompartner=RETAILERX

Now if Bots receives an 850 message from RETAILERX, the 2nd translation is used. For other partners the first translation is used.

Note: a partner can belong to a partner group. You can specify the partner group in the translations. If Bots translates messages of a partner belonging to a message group, Bots uses the translation specific for this partner group.

6. Mapping scripts

Each translation uses a mapping script. A mapping script contains detailed instructions about how to put content from an incoming message in an outgoing message. Mapping scripts are python programs.

The best way to learn this is to look at the examples (eg in the plugins).

Use an editor with syntax highlighting and the ability to 'compile'/check syntax.

Environment for mapping scripts

- Place of mapping scripts: in `usersys/mappings/<name of mapping script>.py`
- Bots-engine starts function 'main' of a mapping script.
- 'main' receives 2 parameters: inn (incoming message) and out (outgoing message).
- Mapping scripts are python scripts: use python's power; normal python rules apply.
- Incoming message object (inn) has a dict 'ta_info'. It contains information about the message as specified in the grammar (queries, SUBTRANSLATION). Use ta_info to access eg envelope data like the topartner and frompartner. Do not change ta_info.
- Outgoing message object (out) also has a dict ta_info. Contains:
 - A reference number; you can use this as a message number.
 - Set the 'topartner'. Especially useful if you use partner-dependent syntax for outgoing messages.
- All data about incoming/outgoing messages are strings.
- If using calculations in a mapping script, use type Decimal, not 'float' or 'integer'.
- Advice: set up your grammar to receive one message at a time in the mapping script. This is the easiest way to make mapping scripts: one message in->one message out. This is configured using 'nextmessage' in the grammar.
- Errors in a mapping script are caught by Bots (and displayed in bots-monitor).
- Raise an error in a mapping script if encountering an error situation.

6.1. Functions in a mapping script

('mpath' identifies data in a message; more information about mpath in next chapter)

3.4.2. Get-functions: retrieve data from message.

1. `get (mpath)`
 - Get 1 field from the incoming message; mpath specifies which field to get.
 - Returns: string, or, if field not found, None
 - Example to get the message date from an edifact INVOICD96AUNEAN008:
`inn.get({'BOTSID':'UNH'},{'BOTSID':'DTM','C507.2005':'137','C507.2380':None})`
Explanation: get field C507.2380 from DTM-record if field C507.2005 is '137', DTM-record nested under UNH-record.
The field to retrieve is specified as None.
2. `getnozero (mpath)`
 - Like `get ()`, but: returns a numeric **string** not equal to '0', otherwise None.
3. `getloop (mpath)`
 - For looping over repeated records or record groups.
 - Typical use: loop over article lines in an order.
 - Returns an object usable with `get ()`; see example.
 - If mpath is not found or not correct: no looping, gives no explicit warning.

- Example to loop over lines in edifact order:

```
for lin in inn.getloop({'BOTSID':'UNH'},{'BOTSID':'LIN'}):
    linenumber = lin.get({'BOTSID':'LIN','1082':None})
    articlenumber = lin.get({'BOTSID':'LIN','C212.7140':None})
    quantity = lin.get({'BOTSID':'LIN'},{'BOTSID':'QTY','C186.6063':'21','C186.6060':None})
```

3.4.3. Put-functions: place data in message.

1. `put (mpath)`
 - Places the field(s)/record(s) as specified in mpath in the outmessage.
 - Returns: if successful, True, otherwise False.
 - If mpath contains None-values (typically because a `get()` gave no result) nothing is placed in the outmessage, and `put()` returns False.
 - Example to put a message date in a edifact INVOICD96AUNEAN008:

```
out.put({'BOTSID':'UNH'},{'BOTSID':'DTM','C507.2005':'137','C507.2380':'20070521'})
```

Explanation: put date '20070521' in field C507.2380 and code '137' in field C507.2005 of DTM-record; DTM-record is nested under UNH-record.
2. `putloop (mpath)`
 - Used to generate repeated records or record groups.
 - Recommended: only use it as in: `line = putloop(<mpath-parameters>);`
line is used as `line.put()`
 - Typical use: generate article lines in an order.
 - Note: do not use to loop over every record, use `put()` with the right selection.
 - example of usage (extended from example at `getloop()`; loop over lines in edifact-order and write them to fixed in-house):

```
for lin in inn.getloop({'BOTSID':'UNH'},{'BOTSID':'LIN'}):
    lou = out.putloop({'BOTSID':'HEA'},{'BOTSID':'LIN'})
    lou.put({'BOTSID':'LIN','REGEL':lin.get({'BOTSID':'LIN','1082':None}}))
    lou.put({'BOTSID':'LIN','ARTIKEL':lin.get({'BOTSID':'LIN','C212.7140':None}}))
    lou.put({'BOTSID':'LIN','BESTELDAANTAL':lin.get({'BOTSID':'LIN'},
                                                    {'BOTSID':'QTY','C186.6063':'21','C186.6060':None}}))
```
3. `getcount()`
 - returns the number of records in the tree or node.
 - typically used for UNT-count of segments.
 - example of usage:

```
out.getcount()
```

(returns the numbers of records in outmessage.)
4. `getcountoccurrences (mpath)`
 - returns the number of records selected by mpath.
 - typically used to count number of LIN segments.
 - example of usage:

```
out.getcountoccurrences({'BOTSID':'UNH'},{'BOTSID':'LIN'})
```

(returns the numbers of LIN-records.)
5. `getcountsum (mpath)`
 - counts the totals value as selected by mpath.
 - typically used to count total number of ordered articles.
 - example of usage:

```
out.getcountsum({'BOTSID':'UNH'},{'BOTSID':'LIN'},
{'BOTSID':'QTY','C186.6063':'12','C186.6060':None})
```

(returns total number of ordered articles.)
6. `sort (mpath)`
 - Returns nothing (None)

- Sorts the incoming message. Specify what to sort with what key by using 'mpath'.

- Sorts alphabetically.

- example of usage:

```
inn.sort({'BOTSID':'UNH'},{'BOTSID':'LIN','C212.7140':None})
```

(sorts the incoming article lines on EAN article number.)

- Note: sort only sorts just below the node form which is called. In the example above, the root of inn is the UNH-segment.

7. delete(mpath)

- Delete(s) the record(s) as specified in mpath in the outmessage (and the records 'under' that record).
- After deletion, searching stops (if more than one records exists for this mpath, only the first one is deleted)
- Returns: if successful, True, otherwise False.

- Example to delete a message date in a edifact INVOICD96AUNEAN008:

```
out.delete({'BOTSID':'UNH'},{'BOTSID':'DTM','C507.2005':'137'})
```

Explanation: delete DTM record where field C507.2005 = '137' ; DTM-record is nested under UNH-record.

- Example to delete all ALC segments in edifact message:

```
while message.delete({'BOTSID':'UNH'},{'BOTSID':'ALC'}):
    pass
```

8. change(where=(mpath), change=mpath)

- Used to change an existing record. 'where' identifies the record, 'change' are the values that will be changed (or added is values do not exist) in this record.
- Only one record is changed. This is always the last record of the where-mpath
- example of usage:

```
inn.change(where=({'BOTSID':'UNH'},{'BOTSID':'NAD','3035':'DP'}),change=({'3035':'ST'})
```

Explanation: change qualifier 'DP' in NAD record to 'ST'

3.4.4. Codeconversion-functions.

1. transform.codeconversion(filename, value)

- Converts code 'value' using code list in *usersys/codeconversions/<filename>.py*
- Codelist is a python dict.
- Returns the converted code; if not found raises exception
`botslib.CodeConversionError`

- example of usage:

```
transform.codeconversion(myodelistfile,'value')
```

(returns the code found via lookup of 'value' in 'myodelistfile.py').

2. transform.rcodeconversion(filename, value)

- as codeconversion, but conversion is from right to left.
- If the same value occurs more than once on the right-hand side, one of the left-hand side values is returned (undetermined).
- example of usage: see `transform.codeconversion()`.

3. transform.codetconversion(codelist, value, field)

- Converts code 'value' using a user-maintained code list.
- Convert from left to <field>
- Fields indicates the field in the user maintained codelist you want to have returned. Default (if not used) the 'rightcode' is returned. See in *Bots monitor* for other values.
- Returns the converted code; if not found raises exception
`botslib.CodeConversionError`

- Example of usage:

transform.codetconversion(myodelist,'value')

(returns the code found via lookup of 'value' in dbtable 'myodelist').

4. *transform.rcodetconversion(codelist, value, field)*

- as *transform.codetconversion()*, but conversion is from right to <field>.

All the above codeconversion have a 'safe' variant (eg *safecodetconversion()*).

They do the the same as above, but when the conversion is not possible it does not raise an error but just returns the (original) value.

3.4.5. Persist-functions: store data for use in other messages/transactions

Store data for use in other translations. This might be useful eg for storing data from an incoming order, and use the data later for DESADV and/or INVOIC.

- You can store and retrieve 'any' python data (python pickle is used);
- storage size is limited to 1024 positions.
- parameter 'maxdayspersist' in bots.ini controls

1. *transform.persist_add(domain, key, value)*

- if add the value is not possible eg because domain-value exists already, *botlib.PersistError* is raised.

1. *transform.persist_update(domain, key, value)*

- if domain-value does not exist: gives no error.

1. *transform.persist_lookup(domain, key)*

- if domain-value does not exist: returns None

1. *transform.persist_delete(domain, key)*

- if domain-value does not exist: no error.

3.4.6. Miscellaneous.

1. *transform.useoneof(*arg)*

- Use when different mapping can occur for the same data.

- Example of usage:

transform.useoneof(lin.get({'BOTSID':'LIN'},{'BOTSID':'IMD','C960.4294':None},{'BOTSID':'LIN'},{'BOTSID':'IMD','7081':'DESC','C960.4294':None}))

- *useoneof* returns the first arg of *arg that 'exists' (=evaluates as True).

1. *transform.inn2out(inn,out)*

- Use the incoming message as the outgoing message. (verbatim copy)
- Is useful to translate the message to another editype. Examples:
 - edifact to flat file. This is what a lot of translators do.
 - x12 to xml. x12 data is translated to xml syntax, semantics are of course still x12
- Another use: read a edi message, adapt, and write (to same editype/message type including changes).

1. *transform.unique(domain)*

- Returns counter/unique number.
- For each 'domain' separate counters are used.
- Counter start at '1' (at first time you use counter). Maximum of counter is at least (2**31)-2; if reached is reset to 1.
- Example of usage:

transform.unique('my article line counter')

(returns a number unique for the domain).

2. *transform.eancheck(EAN_number)*

- Returns True if this is a EAN number (checks checkdigit), False if not.
- Synonyms for EAN number: GTIN, ILN, UPC, EAN, UAC, JAN
- Can be used for UPC-A, UPC-E, EAN8, EAN13, ITF-14, SSCC/EAN-128 etc.
- When not a string with digits, raises `botlib.EanError`
- example of usage:

```
transform.eancheck('8712345678906')
```

(returns 'True' for this EAN number).

3. `transform.calceancheckdigit(EAN_number)`

- Returns the checkdigit-string for a EAN number (without checkdigit).
- example of usage:

```
transform.calceancheckdigit('871234567890')
```

(returns '6' for EAN number '871234567890').

4. `transform.addeancheckdigit(EAN_number)`

- Returns EAN number including check digit (adds checkdigit).
- example of usage:

```
transform.addeancheckdigit('8712345678906')
```

(returns '8712345678906' for EAN number '871234567890').

6.2. Mpath

3.4.7.

(Lot of text to explain something that's easy to understand by example. Look at the examples!! We think the use of mpaths is quite intuitive if you know EDI).

- Mpath consists of one or more python 'dicts' (dictionaries), separated by commas:

```
dict1, dict2, dict3
```

- A python dict consists of: opening brace, (one or more) name-value pairs separated by commas, closing brace:

```
{'name1':'value1', 'name2':'value2', 'name3':'value3'}
```

- So an mpath might look like this:
- First dict is for 'root' level of records, 2nd dict is for records nested under root level, 3rd dict is for records nested under records of 2nd level, etc.
- 'BOTSID' is the record identifier/segment tag/XML tag. This is required.
- In mpath used in `get()`: value None indicates that the value of this field should be returned.
- In mpath used in `get()`: all other name-values are interpreted as: if field==value.
- In mpath used in `put()`: all name-values are written, except if one of the values is None, in which case none of mpath's contents are written.

6.3. How does a mapping work

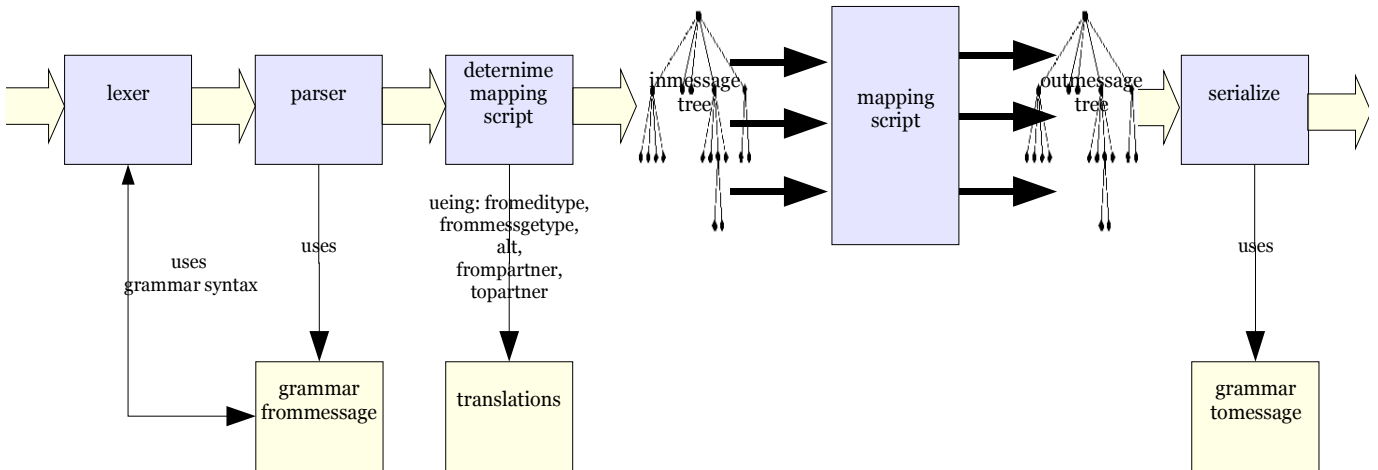
Bots reads an EDI file into memory. The message is lexed and parsed.

The message is then transformed into a tree structure, similar to the use of DOM in XML.

`get(mpath)` is used to search this tree. This way the information in the message is easy accessible. There is no looping over the segments in a message like a lot of translators do; which is quite similar to a SAX-parser in XML.

When you 'write' to the outmessage with `put (mpath)` a new tree is build. When the mapping script is done, the tree is serialized to an EDI message.

Schema:



Using `put (mpath)` to build a tree is surprisingly intuitive. You write whole segments at once in the right place in the EDI message.

Because a message is represented as a tree, it is simple to sort e.g. the article lines in a incoming message by article number.

Theoretically, a message could be too large to read into memory. But as it is, EDI messages are small and computer memory is cheap.

6.4. Loose ends

1. If multiple messages in an EDI file are not split up (a grammar without nextmessage):
 1. Using `inn.get()` gives an error: that "root" of incoming message is empty, etc.
 2. You start the mapping script with `getloop()`; and do `get()` with results of `getloop()`
 3. Examples are provided.
 4. It's best to split up messages.
2. For messages consisting of only one record type (typically CSV):
 1. (recommended) use `nextmessageblock()` to separate the messages
 2. in-message: you must use `getloop()` to get the records
 3. out-message: you must do a `write()` for every record.
 4. Examples provided (translation from as well as to CSV)

7. Message grammars

1. A grammar is a description of an EDI message.
2. All grammar files are in: `usersys/grammars/<editype>/<grammar name>.py`
3. A grammar file has several sections.
4. A section is either:
 - the section itself.
 - import from another file: `from <filename> import <section>`
e.g.: `from D96Areconds import recorddefs`
Purpose: sections in separate files for better maintenance; e.g. all D.96a edifact messages use the same segments..
5. This chapter contains details about all the possible sections in a grammar.
6. Use an editor with syntax highlighting and possibility to 'compile' (checks syntax).
7. Bots has a tool to check a grammar (bots-grammar).

7.1. Syntax parameters

- Is a python dict.
- Syntax parameters have default values, dependent upon the editype. If you want to see all the defaults used: in `bots/grammars.py` all the defaults are set in the classes of the edi-types.
- For outgoing messages it is possible to specify a partner dependent syntax; this is especially useful for x12.

Syntax parameters:

<code>add_crfterrecord_sep</code>	<i>In:</i> N/A <i>Out:</i> put extra <CR/LF> after a record/segment separator. Value: string, typically '\n' or '\r\n'
<code>acceptspaceinnumfield</code>	<i>In:</i> for fixed formats; do not raise error for numeric field having no contents (but assume value is 0) <i>Out:</i> N/A
<code>charset</code>	<i>In:</i> charset to use; can be overridden by charset-declaration in content. <i>Out:</i> charset to use in output. Bots is quite strict in this. Note: a list of charsets supported by bots is available via python: http://docs.python.org/lib/lib.html - search for codecs, that's how this is called in python.
<code>checkcharsetin</code>	<i>In:</i> what to do for chars not in charset. Possible values 'strict' (gives error) or 'ignore' (skip the characters not in the charset) or 'botsreplace' (replace with char as set in bots.ini; default is space) <i>Out:</i> N/A
<code>checkcharsetout</code>	<i>In:</i> N/A <i>Out:</i> what to do for chars not in charset. Possible values 'strict' (gives error), 'ignore' (skip the characters not in the charset) or 'botsreplace' (replace with char as set in bots.ini; default is space)
<code>checkfixedrecordtooshort</code>	<i>In</i> (fixed): warn if record too short. Possible values: True/False, default: False <i>Out:</i> N/A
<code>checkunknownentities</code>	<i>In/Out:</i> for XML, JSON: skip unknown attributes/elements (instead of raising an error)

contenttype	<i>Out:</i> content type of translated file; used as mime-envelope of email
decimaal	<i>In/Out:</i> decimal point; default is '.'. For edifact: this is read from UNA-string if present.
envelope	<i>In:</i> N/A <i>Out:</i> envelope to use; if nothing specified: no envelope - files are just copied/appended.
escape	<i>In/Out:</i> escape character used. Defaults: edifact: '?'. <i>Out:</i> escape character used. Defaults: edifact: '?'. <i>In/Out:</i> field separator. Defaults: edifact: '+'; CSV: ',' x12: '*')
field_sep	<i>In/Out:</i> field separator. Defaults: edifact: '+'; CSV: ',' x12: '*')
flattenxml	<i>In/Out:</i> write fields of a record as XML-elements (instead of attributes). This is useful for eg straight fixed to xml mappings.
forcequote	<i>In:</i> N/A <i>Out:</i> for CSV: Possible values: 0: (default) quote only if necessary; 1: always use quote
startrecordID	<i>In (fixed):</i> start position of record ID; value: number, default 0. <i>Out:</i> N/A
endrecordID	<i>In(fixed):</i> end position of record ID; value: number, default 3. <i>Out:</i> N/A
merge	<i>In:</i> N/A <i>Out:</i> merge translated messages to one file (for same sender, receiver, messagetype, channel, etc). Related: envelope. Possible values: True; False
noBOTSID	<i>In/Out (CSV):</i> use if records contain no real record ID/tag
output	<i>In:</i> N/A <i>Out:</i> (template) values: 'xhtml-strict'
quote_char	<i>In/Out (CSV):</i> char used as quote symbol
record_sep	<i>In/Out:</i> char used as record separator. Defaults: edifact: "'" (single quote); fixed: '\n'; x12: '~'
reserve	<i>In/Out (edifact,x12):</i> repetition separator; not yet in use; generated in UNA
sfield_sep	<i>Out (edifact,x12):</i> subfield separator. Defaults: edifact: ':', x12: '>'
skip_char	<i>In:</i> char(s) to skip, not interpreted when reading file. Typically '\n' in edifact (another solution here is to use 'charset = 'unoa-strict' and 'checkcharset' = 'ignore'. But then all non-UNOA chars are ignored.) <i>Out:</i> N/A
skip_firstline	<i>In (CSV):</i> skip first line (often contains field names). Possible values: True/False, default: False
template	<i>In:</i> N/A (template): XML/XHTML template to use for HTML-output.
triad	<i>In:</i> triad (thousands) symbol used (e.g. ',' in '1,000,048.35'). If specified, this symbol is skipped when parsing numbers. By default, it is left out (numbers are expected to come without thousands separators). <i>Out:</i> N/A
version	<i>In (edifact, x12)</i> not used, is read from envelope fields. <i>Out (edifact,x12):</i> version of standard generate. Value: string, typically: '3' or '004010'
Specific field in envelope	X12: ISA01 ISA02 ISA03

```

ISA04
ISA05
ISA07
ISA11
version (ISA12)
ISA14
ISA15
functionalgroup (GS01)
GS07
Edifact:
    S001.0001 (charset)
    S001.0002 (version)
    UNB.S002.0007
    UNB.S003.0007

```

7.2. Structure: sequence of records in a message

Is required (except for template grammars).

Is a list of records.

Each record is a python dict.

Each dict has the following keys:

1. ID: tag of record.
The record tag must be unique in the list of records.
If an edifact message structure contains 'UNS'-records, you should 'nest' all segments after UNS under UNS.
2. MIN: minimum number of occurrences.
3. MAX: maximum number of occurrences.
4. LEVEL: list of records, nested under this record.
5. QUERIES:
 - Content: dict; where key = parametername, value is mpath for get().
 - Purpose: extract information from incoming EDI messages; information is passed to Bots machinery.
 - Use of QUERIES information:
 1. For updating db-ta; this information can be used in further routing of message.
 2. Information is passed to mapping script; accessible with `inn.ta_info[parameter]`.
 3. This is the only way to access envelope information in case of standard envelope/SUBTRANSLATION.
 4. editype, messagetype, frompartner, topartner, alt are used to find the right translation.
 - Advice: use in combination with section 'nextmessage'
6. SUBTRANSLATION:
 - Content is a dict (mpath), tuple or list
 - Purpose: start a translation of a message within a standard envelope.
 - Usage: info in SUBTRANSLATION is used to retrieve 'messagetype' from message; this messagetype is used to start the translation.
 - Subtranslation starts at the nesting level it is given.

- Advice: use together with section 'nextmessage', otherwise your mapping script has to handle several messagetypes in one script.

3.5. Recorddefs: definition of records; describes the fields in each record.

Is required, except for template grammars.

Is a dictionary; key is record ID/tag, value is list of fields.

The first field in the list is **always** field 'BOTSID'.

- In CSV: some CSV files do not have a real record ID/record type. Use `grammar.syntax['noBOTSID']=True`; Bots uses name of record for BOTSID. See examples.
- For fixed records, Bots has to know where record ID is. Default: first three positions in record, can be parameterized in the syntax of the grammar file.
- edifact: record ID is edifact tag (DTM, BGM, etc).
- X12: record ID is the segment identifier
- XML: record ID is XML tag (without angle brackets).

Each field is a list of:

1. field ID; it is strongly recommended to make this unique (Bots does not check for this; rather, when it encounters multiple fields with the same ID, it uses the first one).
2. M/C: mandatory or conditional.
3. max length; for N-field it is possible to specify max number of decimals. This is used in checking incoming numbers, and to format outgoing numbers.
4. format: specific per editype (edifact, x12, etc); see their specific rules. For csv, fixed records etc no standard formats exists, so Bots uses these formats
 - AN or A: alphanumeric.
 - N: numeric field with fixed numbers of decimals. Length includes zeroes, decimal point, minus sign, etc. Values are rounded if too many decimals outgoing, incoming gives an error. Exponential notation is not allowed. For outgoing Bots tries to convert the exponent. Negative is allowed.
 - R: numeric field with floating decimal point. Exponential notation is allowed. . Length includes zeroes, decimal point, minus sign, exponential sign etc.
 - I: implicit decimal. Bots converts this to explicit decimals (in mapping script). For outgoing: Bots converts this to the right implicit notation of the number. Is rounded. Can be negative, no exponential notation is allowed. For outgoing Bots tries to convert the exponent.
 - D/DT: date. Can be 8 or 6 positions. 8 positions: CCYYMMDD, 6 positions: YYMMDD. Bots checks for valid dates both in- en outgoing.
 - T/TM: time. Can be 4 or 6 positions. 4 positions: HHMM, 6 positions: HHMMSS. Bots checks for valid times both in- en outgoing.

For edifact/x12, composites are possible. A composite has:

1. field ID; it is strongly recommended to make this unique (Bots does not check for this; rather, when it encounters multiple composites with the same ID, it uses the first one).
2. M/C (mandatory/conditional).
3. a list of subfields.

Implementation details of field formats in grammar:

- A(N): charset is checked using charset/unicode when reading EDI file.
- max length of field is tested (field is tested as it is, incl. all spaces, zeroes, decimal separator, etc).
- numerical:
 - '-' is accepted both at beginning and end. Bots outputs only leading '-'
 - '+' is accepted at beginning. Bots does not output '+'
 - thousands separators are removed if specified in syntax-parameter 'triad' (see above).
Bots does not output thousands separators
 - default decimal separator is '.';
change default with syntax-parameter 'decimaal'; Bots uses this to convert '.' to ',' so if set to ',' Bots accepts both ',' and '.'
 - leading zeroes are removed
 - trailing zeroes are kept
 - '.45' is converted to '0.45' (incoming en outgoing). The missing zero is not counted in validating max length
 - '4.' is converted to '4'. The decimal point *is* used in validating max length
 - if a numeric field is not a valid number, it results in an error message

3.6. nextmessage: split an EDI file in separate messages

'nextmessage' is used for splitting the messages in an EDI file; this way a mapping script receives one message at a time. Nextmessage is an mpath used by get().

Example 1: translation of fixed file with orders. Each orders starts with (header) record 'HEA'; order lines are in 'LIN' records. To split up the file in separate order messages you would use:

```
nextmessage = ({'BOTSID':'HEA'},)
```

Example 2: in edifact envelope (UNB-UNZ) the messages are split up using:

```
nextmessage = ({'BOTSID':'UNB'},{'BOTSID':'UNH'})
```

In edifact/x12 a envelope can contain more than one messagetype. This requires different translations for these messages; use 'SUBTRANSLATION' (see earlier chapter).

3.7. nextmessageblock: split an CSV file into messages

'nextmessageblock' is used for splitting the messages in an EDI file with a single type of record, typically of editype CSV. Nextmessageblock is an mpath used by get().

Bots-engine uses this mpath to retrieve a value from each record; all subsequent records with the same value are regarded as belonging to the same message, and passed as one message to the mapping script. When Bots encounters a record with a different value, Bots assumes this belongs to the next message.

Example: translation of CSV file with orders; only one record type; each record contains order header data (e.g. order number) and the information of one order line (article number, quantity). To split up the file into separate order messages you would use:

```
nextmessageblock = ({'BOTSID':'HEA','ORDERNUMMER':None})
```

8. User maintained code lists

Use cases:

user maintained code list can be accessed from a mapping script

e.g. convert EAN numbers to internal article numbers; convert partnerID's.

9. Technical overview of bots

Some people find this very clear . Others don't.

