BOTS

Bots Manual

Bots open source edi software

BOTS version 1.3.0
Date: 2008-06-30

EbbersConsult

Table of Contents

1.Introduction....................3
2.Installation....................7
3.Using Bots....................11
4.Introduction to configuration....................14
5.Configuration using plugins....................15
6.Configuration overview....................17
7.Routes....................19
8.Translations....................23
9.Channels (for communication)....................26
10.Partner related....................27
11.User maintained code lists....................28
12.Mapping scripts....................29
13.Message grammars....................35
14.Addendum: technical schema for bots....................40

1. Introduction

Bots is open source software for EDI.
EDI (Electronic Data Interchange) is the exchange of electronic business data between companies. EDI is also known as business-to-business commerce, b2b commerce, electronic commerce, xml exchange etc.

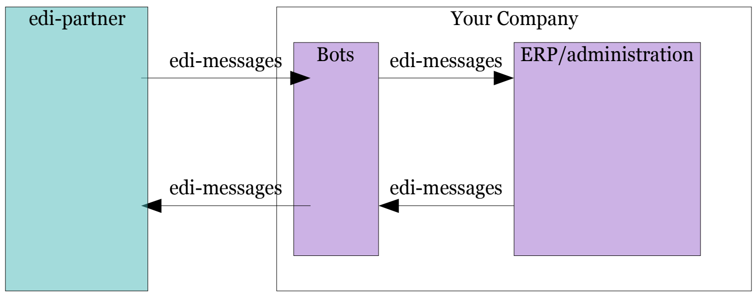
In EDI data is exchanged between two computers, without human intervention; this is different from e.g. the Internet, where humans enter or read the data. In EDI only data is exchanged (e.g., an order), your administration remains independent of the administrations of trading partners: it is not possible to peek into the administration systems of your trading partners (and neither can they peek into yours!)
Link for more information about EDI on wikipedia.

Examples of EDI:
- a retailer orders every day by EDI. The supplier sends invoices to this retailer by EDI.
- automotive: just-in-time deliveries by the suppliers.

What can Bots do for you and your company?
- Bots communicates electronic documents from and to your trading partners.
- Bots translates the EDI messages of your trading partners to a format suitable for your administration (and vice versa).
- Bots monitors and manages the flow of incoming and outgoing messages, alerting you when errors occur.

Your existing administration must be able to, for example, import orders or export invoices.

Place of Bots in your company:



1.1. Bots is open source software

Open Source software is free software. Bots is free both as in 'freedom of speech' and as in 'free beer'. The user's rights are of chief concern: the right to use, copy, distribute, study, change and improve upon the software. Mozilla Firefox, OpenOffice, and Linux are examples of successful Open Source projects.

Bots uses the GNU General Public Licence (GPL).
This means: use Bots as you like for free: use it, change it, distribute it. The only 'limitation'

is: if you distribute Bots (or a modified version of Bots, or software based on Bots), you have to do this under the same GPL licence. We consider this is no limitation, but an assurance to protect the user's rights by guaranteeing that Bots remains open and free. You are explicitly not forbidden to distribute Bots, or software based on Bots, for a fee.

But how about support?
Commercial support is available from the same company that designed and built Bots: EbbersConsult. EbbersConsult is glad to help you:
- Instal and set up Bots.
- Connect your system to the EDI systems of your trading partners.
- Introduce EDI in your organisation.
- Customise Bots for your organisation.
- Helpdesk support.

So, what's the catch?
There is none. Download Bots yourself, install it, try it, configure it, customise it, distribute it, etc. Users can exchange configurations, grammars, modifications, etcetera, so you don't pay for customisations that are already available. You are explicitly encouraged to add your own contributions on the website.

If the Open Source software licence poses a problem for you - e.g. if you wish to distribute a modified version of Bots, but do not wish your modifications to become Open Source software themselves (and, hence, to be visible to anybody) - you can obtain Bots under a different licence from EbbersConsult.

More information about Open Source software:
- Wikipedia on Open Source
- the Open Source Initiative
- the complete text of the GNU General Public Licence

1.2. Features

Headlines:
- (Secure) communications.
- Any-to-any-format conversions.
- Manage and overview EDI data flows.

EDI formats handled:
- EDIFACT.
- X12.
- XML.
- Fixed records/flat file/ASCII.
- CSV-format (excel).
- SAP idoc.
- JSON
- Readable/printable format ('EDI-fax').
- Code conversion, e.g. EAN/UCC-numbers to internal article number.

Bots translates any of these formats to any other of these formats. This makes Bots very flexible.

Communication
- Email (POP3 and SMTP).
- Secure email (POP3 and SMTP over SSL/TLS).
- FTP.
- File I/O.
- VAN's: supported are now
  - IPMail (Btinfonet)
  - InterCommit.
- X400: not direct. Advise: use VAN's like IPMail or InterCommit for transparant connection to X400. Better and less expensive.
- AS/2: not direct. Advise: use VAN's like IPMail or InterCommit. Better and less expensive.

Manage EDI:
- View: what has been received, what has been sent
- Resend
- Re-receive
- Receive (email) reports in case of errors.

Flexible deployment:
- Bots is web-based You can use bots-monitor from any workstation.
- Communication and translation can be started by the user or be scheduled.
- Get a report by email if errors occur in communication or translation.

Plugins for distributable configuration:
- Easy installation of standard EDI scenarios.
- Distribution and sharing of configurations by e.g. Bots users, buyers, suppliers of ERP software or EDI communities.

Technical info:
- Highly configurable; advanced configuration using scripting.
- Query EDI messages in mapping scripts to retrieve and place content. Think of X-path or SQL, but tailored for EDI.
- Incoming messages are converted to a tree structure (think DOM for XML). Outward messages are also built as a tree structure and serialized to an EDI message.
- Programming language: Python.
- Python is used for configuration (mapping scripts, grammars, route scripts etc.) rather than a self-invented toy language. This enables you to use existing tools for e.g. syntax checking and highlighting.
- Fully integrated Unicode (character sets).
- Platform-independent; known to work on Windows and Linux. Bots should work on any platform that runs Python, but as it uses some external modules, you should check that those work if you use a different platform. Some modules have C extensions. A list of these external modules is available on the Documentation section of the Bots web site.

- Scalable: different parts of Bots can run on different computers; Bots supports databases suited for heavy loads.
- Supported databases: SQLite (default), MySQL, PostgreSQL.

1.3. Thanks

Bots is made possible by the people who made the following software:

- Python
- SQLAlchemy
- PyProtocols
- TurboCheetah
- Cheetah
- TurboKid
- Kid
- CherryPy
- ConfigObj
- DecoratorTools
- elementtree
- RuleDispatch
- FormEncode
- PasteScript
- PasteDeploy
- Paste
- Turbogears
- PyProtocols
- Setuptools
- pysqlite
- mysql-Python
- psycopg2
- SQLite
- MySQL
- PostgreSQL

Thanks!

2. Installation

By default Bots uses the SQLite database.
For usage of another database, see specific chapter.

Bots uses a python runtime and a lot of libraries. The libraries are downloaded at install time. To ensure a good installation the libraries are downloaded from one place.

2.1. Windows installer

1. Install Python
   Check if Python is already installed.
   Bots needs Python 2.4 or higher; recommended is Python 2.5.
   Download Python installer from http://www.Python.org
2. Install Bots
   Download Bots installer from http://bots.sourceforge.net
   During the installation the libraries Bots needs are installed.
   Some libraries like sqlite have their own 'windows installer screens', just install them; you'll probably see a lot of 'DOS'-windows, most blank, of with some text. This is OK. You'll be notified the installation went OK. If not: contact us via the mailing list.

2.2. Package manager (e.g. Linux, Unix, etc)

Package manager should handle requirements. For now, there is no *.deb or *.rpm for Bots; so we are using Python eggs to install it.
Install with your package manager:

1. Python2.4 or higher; recommended is Python 2.5.
   Often this is already installed.
2. Pysqlite.
   Pysqlite has a automatic dependency on SQLite.
   Pysqlite is not needed if you use Python>=2.5;
   Pysqlite is not needed if you prefer another database; see specific chapter.
3. TurboGears>= 1.02

Then download the bots python egg for your Python version. Then (command line):
easy_install bots-1.3.0-py2.5.egg
(adapt command to the egg you've downloaded)

2.3. Manual installation

Python
For download and instructions http://www.Python.org
Bots needs Python 2.4 or higher; recommended is Python 2.5.

1. (py)SQLite
   Needed for Python 2.4; not needed for Python 2.5 and higher.
   For download and instructions: http://www.pysqlite.org/
2. TurboGears>=1.0.4
   You have to install all dependencies, including SQLAlchemy>4.0.
3. Bots
   Download the Bots zip file. Place code and data in the 'right' place.
   Bots uses Python; normally Bots is installed in 'site-packages' of Python.
   Bots has some scripts (e.g. place in '/usr/bin'); look in zip-file for details.

In the zip-file is a SQLite database included. Use that; or roll your own. Roll your own database this way (that's how I make a new database for Bots):

1. Create the required tables for Bots:
   (command line) *tg-admin sql create*
   (tg-admin is a script/executable from TurboGears; under Windows the script/executable would be in something like: 'c:\python25\Scripts'.
2. Create a user in the database; use the database view/editor of your choice.
   User should be initialised in table tg_user.
   user_id, user_name and password are required.
3. Initialise the database.
   This has to be done in order to initialise some required records.
   Download from the bots web site pluging initbotsdatabase.zip; install it using the user interface (bots monitor); on the web site are detailed instructions for installing a plugin.

2.4. Using a different database than SQLite

(This chapter assumes you have sufficient knowledge of your database system.)
First install Bots in the normal way; see previous chapters.

1. Both the database and a Python interface module for the database have to be installed. Information about databases supported by Bots:

Database	Python DB API
MySQL	MySQL for Python http://sourceforge.net/projects/mysql-Python
PostgreSQL	Psycopg2 http://initd.org/projects/psycopg2 Older version use mx-datetime; you'll have to use more recent versions. mx is/was a dependency for PyGreSQL-package for Debian/Ubuntu; but in fact mx is not needed and gives runtime errors. Bots uses Python's datetime, not mx-datetime. For me, a work-around was to remove/rename mx...

2. Create a database for Bots; use utf8 as charset.
   Create a db-user.
   In the examples the database is called 'botsdb'.
   In the examples a username 'bots' with password 'botsbots' is used.
   The db-user has to have sufficient rights in your database; the user has to be able to create tables. (Bots also requires a user password; this is completely separate from and has nothing to do with the db-user).
3. Change Bots db-settings:
   In the installation directory of Bots is a text file called 'botstg.cfg'.
   Parameter 'sqlalchemy.dburi' determines which database is used.
   Examples of configurations of other databases are provided in botstg.cfg.
   Change settings to your preferences and save 'botstg.cfg'.
4. Create the required tables for Bots:
   (command line) *tg-admin --config=<location botstg.cfg> sql create*
   (tg-admin is a script/executable from turbogears; in Windows the script/executable would be in something like: 'c:\python25\Scripts'. Be aware that Python's installation doesn't automatically add this directory to the Windows path.)
5. Create a user in the database; use the database view/editor of your choice.
   User should be initialised in table tg_user.
   user_id, user_name and password are required.
6. Initialise the database.
   This has to be done in order to initialise some required records.
   Download from the bots web site pluging initbotsdatabase.zip; install it using the user interface (bots monitor); on the web site are detailed instructions for installing a plugin.

Now Bots should be using the database you configured. When starting bots-engine or bots-web server Bots tells you which database it uses.

2.5. What is installed by Bots?

- (program/script) bots-engine (does the actual EDI communication and translation)
- (program/script) bots-webserver
- (program/script) bots-plugout (for generating plugins)
- (program/script) bots-grammar (for checking a grammar)
- (program/script) bots-unlockdb (for unlocking the database if locked. This should only be needed when Bots crashes, or when power breaks down, or when developing.)
- bots directory: program code, data, configuration, etc. Directories always present:
  - bots root directory. Installed as e.g. (Windows) *C:\python24\Lib\sitepackages\bots* or (*NIX) */usr/lib/python2.4/site-packages/bots-1.1.0-py2.4.egg/bots*.
    Contains program files (*.py, *.pyc etc).
  - botssys
    - data: internal storage of EDI files.
    - sqlitedb: database file(s) of SQLite.
    - infile (default place for plugins to install test files)
    - outfile (default place for plugings to put translated edi files)
    - infile: plugins places input/test edi files here.
    - outfile: plugins places output/translated edi files here.

- config: configuration files.
- static: static files for bots-webserver (CSS, html, images, JavaScript)
- templates: template for bots-webserver
- tests: not used.
- install*: python libraries used during installation.
- usersys: user configuration files for edi (bots-engine configuration)
  - charsets: e.g. edifact uses its own charsets.
  - code conversions: for conversions of codes.
  - grammar: grammars for EDI messages; contains directories per editype
  - mappings: mappings scripts; contains directories per editype.
  - routescripts.
  - partners: partner-specific syntax for outgoing messages; contains directories per editype.

2.5.1. Changing the place for bots storage and configuration on file system

(*nix usually does not store program files, data and configuration in one directory).
Use symbolic links. In Windows, before Vista, this is not possible because Windows has no symbolic links. Most of the time Windows users do not seem to care.
Input and output directories for edi data can be at any place; change this in the channel.

2.6. Examples of Bots setups:

1. Simple installation: Bots is installed on one workstation; the user of this workstation is the only user of Bots.
2. Server installation: Bots is installed on your server. Users from different workstations on your LAN can use the web-based interface (bots-monitor).
3. Extended server installation: bots-engine is scheduled on one computer; bots-webserver runs on another computer; the database on yet another computer (your usual database server). Use bots-monitor from anywhere on your LAN.

3. Using Bots

Bots has the following main components:

- Bots-monitor: the user interface, using your web browser.
- Bots-engine: does the actual EDI communication and translation.

The use of bots-monitor and bots-engine is quite simple; it should be usable by 'everyone'.

3.1. Using bots-monitor

Bots does not support Internet Explorer 6 well (the screen is very 'small'). Install Internet Explorer 7/Firefox/Opera etc. instead. All of these are free. Also note that Microsoft itself advises to use Internet Explorer 7.

Start:

1. First start bots-webserver:
   - installed with Windows installer:
     use the 'shortcut' to Bots-webserver in your 'Programs' menu.
   - other (command line):
     bots-webserver
2. Start your internet browser (e.g. Internet Explorer, Firefox, Opera).
   When bots-webserver runs on the same computer, type in the address:
   http://localhost:8080
   When bots-webserver runs on a different computer, use its IP address or DNS name, e.g.
   http://192.168.10.10:8080
3. Tips:
   - add 'bots' to your favourites/make a bookmark.
   - bots-monitor has a help function; often 'pop-up tips' are available.

What can you do with bots-monitor:

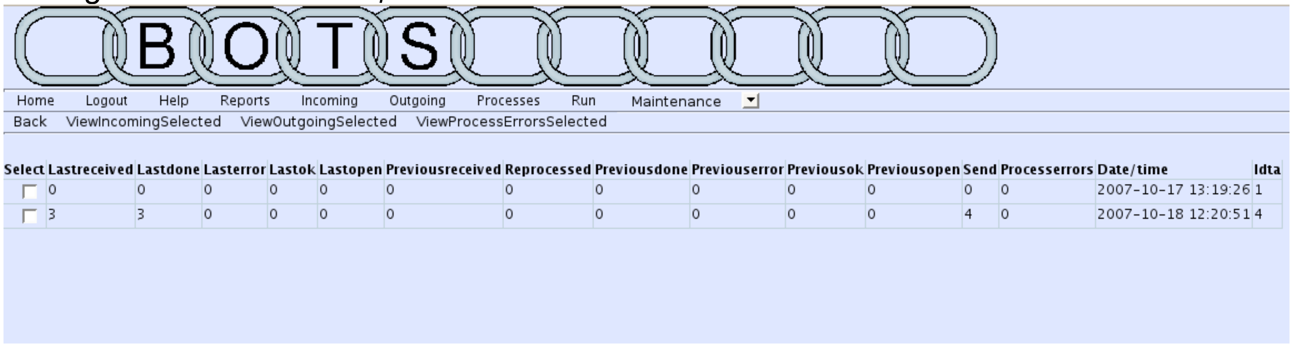
1. Run bots-engine (which performs the actual communications and translations)
2. View the results of (previous) runs of bots-engine.
3. View what files have been received, and the status of processing.
4. View the resulting outgoing messages.
5. Detailed view of what has happened with incoming messages.
6. Re-send or re-receive messages.
7. Configuration.

Bots-monitor requires a user name and password.
Default user name is 'bots' with password 'botsbots'.
Change/add users via *bots-monitor->Maintenance->Users*
The web interface of Bots is not secured: it does not use HTTPS, passwords are sent unencrypted. Do not use bots-monitor over a public network (such as the Internet).

3.2. How to view results in bots-monitor - basics

3.2.1. Reports

When you have run bots-engine, view the results in bots-monitor.
First: go to: *bots-monitor->Reports*. Screenshot:


BOTS
Home Logout Help Reports Incoming Outgoing Processes Run Maintenance
Back ViewIncomingSelected ViewOutgoingSelected ViewProcessErrorsSelected

Select	Lastreceived	Lastdone	Lasterror	Lastok	Lastopen	Previousreceived	Reprocessed	Previousdone	Previouserror	Previousok	Previousopen	Send	Processerrors	Date/time	Idta
☐	0	0	0	0	0	0	0	0	0	0	0	0	0	2007-10-17 13:19:26	1
☐	3	3	0	0	0	0	0	0	0	0	0	4	0	2007-10-18 12:20:51	4

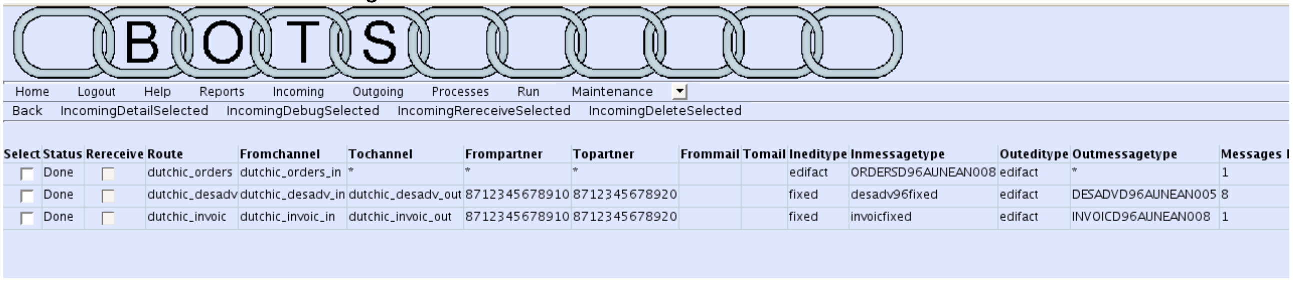
Here you see the results of 2 runs of bots-engine; each run is a row in this list.
The last run is at the bottom of the list; you can see the when it was run.

For the last run:
- 3 edi files were received ('last received'=3)
- 3 edi file were processed completely ('last done'=3)

For the first run: 'last received'=0, so nothing was received and nothing was done.

3.2.2. Incoming

View the incoming EDI files for the last run in more detail:
Go to: *bots-monitor->Incoming*. Screenshot:


BOTS
Home Logout Help Reports Incoming Outgoing Processes Run Maintenance
Back IncomingDetailSelected IncomingDebugSelected IncomingRereceiveSelected IncomingDeleteSelected

Select	Status	Rereceive	Route	Fromchannel	Tochannel	Frompartner	Topartner	Frommail	Tomail	Ineditype	Inmessagetype	Outeditype	Outmessagetype	Messages
☐	Done	☐	dutchic_orders	dutchic_orders_in	*	*	*			edifact	ORDERSD96AUNEAN008	edifact	*	1
☐	Done	☐	dutchic_desadv	dutchic_desadv_in	dutchic_desadv_out	8712345678910	8712345678920			fixed	desadv96fixed	edifact	DESADVD96AUNEAN005	8
☐	Done	☐	dutchic_invoic	dutchic_invoic_in	dutchic_invoic_out	8712345678910	8712345678920			fixed	invoicfixed	edifact	INVOICD96AUNEAN008	1

For each incoming EDI file you can see what the results are, the route, etc.

And naturally, you can see the outgoing files in *bots-monitor->Outgoing*

3.3. Using bots-engine

Can be started in 4 ways - choose your favourite:
1. When installed with Windows installer:
   use the 'shortcut' to bots-engine in menu 'Programs'.
2. Start from bots-monitor:
   bots-monitor->Run
3. Command line:
   bots-engine

4. Schedule bots-engine in your scheduler.

Bots-engine does the communications and translations.
Bots-engine has no user interface (is a batch process).
To view the results of bots-engine, use bots-monitor.
After performing its actions bots-engine stops.
If you start bots-engine using the Windows shortcut the window closes quite fast, so you can not read the output from bots-engine. This is not a problem: use bots-monitor to view the results. If you do want to view the output start bots-engine from the command line/DOS prompt.
You can get error reports by mail; this is especially useful if you schedule bots-engine:

- In file bots/config/bots.ini: set 'sendreportiferror' in 'settings' to 'True'.
- In bots-monitor->maintenance->channels, change the channel for reports ('botsreport') to your situation (e.g. modify address for SMTP).
- In bots-monitor->maintenance->emaildetails, set the right email address for 'botsreportsender' and 'botsreportreceiver'.

Command line parameters:
names of routes can be used as command line parameters. Eg:

```
bots-engine  routeincomingorders  routeoutgoinginvoices
```

Bots will run these routes. Bots does not check if the routes are activated (in the bots-monitor).
By using routes as command line parameters it is easier to call Bots from another program or script.

4. Introduction to 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 3 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.
   See chapter about plugins.

2. Do your own configuration.
   Use bots-monitor for some easy configuration. 'Users' use bots-monitor for tasks like adding EDI-partners or changing user names/passwords, but rarely do advanced 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 are tutorials on the Bots website.
   - Share your grammars etc. with other people. It is easy to make a plugin - see chapter about plugins. Let us know and we'll post them on our web site. We know - the EDI world is not used to sharing. But give it a try.
   - There are plugins available on our website. These provide good examples (we strive to make them production quality, and some are). They provide good grammars for standard messages, etc.
   - Check a grammar using the command line tool 'bots-grammar'.
   - Convert a SEF file to a bots grammar. There is a (free) utility for this bots/sef2bots.py in the distribution. Mind: not all SEF files are adequate - some are demo quality, others contain hard-to-find errors.
   - If we make a grammar ourselves, we separate the definitions of records/segments from the 'structure' of the message. E.g. all edifact D96A messages use the same segment definitions.
   - Please, please use an editor with 'syntax highlighting'. All configuration files are written in Python and syntax highlighting is soooo 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 experiences with us. Bots is constantly being improved, and we value your input.

3. Hire us (www.EbbersConsult.nl - the makers of Bots) to help you with configuring Bots: we are the EDI experts.

5. Configuration using plugins.

5.1. Plugins

Plugins are predefined configurations ready for installation in your copy of Bots.

For example, one plugin contains the necessary information to receive edifact orders (using Dutch EANCOM conventions) and print them (a.k.a. as edi-fax).
Download plugins from http://bots.sourceforge.net

Often a user needs to make small adaptations in bots-monitor to a loaded plugin, e.g. enter user name/password for an email account.

Remark: plugins contain source code. This is a security risk. At the moment, plugins are not verified e.g. using digital signatures. Use only plugins from trusted sources.

5.2. Using bots-plugin

Usage (command line):

bots-plugin <pluginfile>

(On Windows, the bots-plugin script is in a directory called something like 'c:\python25\Scripts'.) This installs the plugin in Bots. By default, it looks for <pluginfile> in the bots/plugins directory. If you saved the file elsewhere, you have to specify the full pathname.

During installation, Bots checks if parts of the plugin are already installed. If so, you are asked whether to overwrite the previously installed parts. Each part has 'owner' and 'version' information.

Installing plugins is a purely local process - nothing is downloaded.

5.3. Make your own plugins: plugout

Use (command line) bots-plugout.
(On Windows, the bots-plugout script is in a directory called something like 'c:\python25\Scripts'.)

Usage: bots-plugout <indexfilename> {<dbtable> | <botsfileobject>}

<botsfileobject> has format: <kindoffile>::<pathname>

<kindoffile> is e.g. "grammars" or "mappings".
<pathname>: directory or file; may contain wildcards.
Writes entries suited for use in a Bots plugin to <indexfilename>.
Does not overwrite <indexfilename>, but appends to it.
Example: plugout target.py mapping::edifact

(write plugin for all mappings in usersys/mappings/edifact)

<dbtable> is the name of a table in bots database.
e.g. "routes", "translation", "channel".
using "*" writes entries for all configuration tables to file.
Example: plugout indexfile.py routes translate channel
(write plugin indexfile for all data in dbtables routes, translate and channel)

Using the above options, you ought to be able to get a good index file for the plugout.

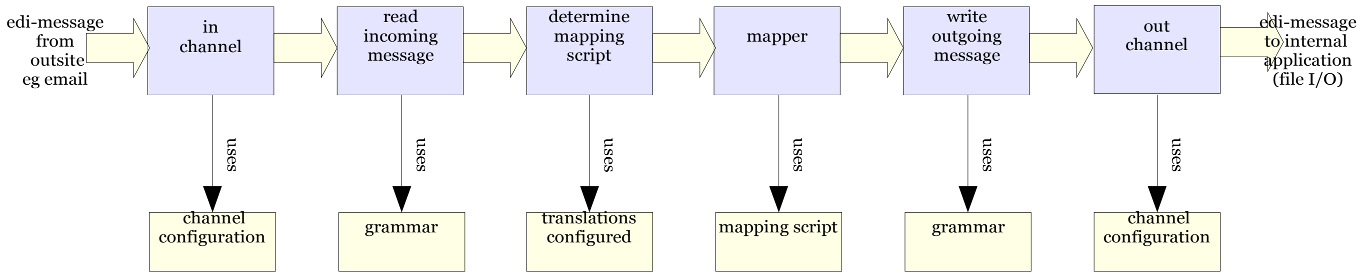
Further steps:
1. Manually deleting to get only the intended configuration in the plugin.
2. Get the configuration files for the plugin. Normally these are files in bots/usersys. We ourselves make a copy of usersys and delete the files we do not want to be in the plugin.
3. Make the plugin by putting the index file plus configuration files (the copy of usersys) in a zip-file.

Details about plugins:
- is a zip file; open it and look at the contents.
- contains a index file with the same name as the plugin (e.g.. plugin "myplugin.zip" contains index file "myplugin.py").
- the index file contains the list 'plugins'. The plugin list consists of Python dictionaries. Each dictionary contains if the database entires needed for the plugin.
- For plugins and /examples: see the bots website.

6. Configuration overview

'Routes' are the most important concept in configuring Bots.
All actions Bots performs are controlled by routes: no routemeans no action.

This picture show what happens in a route:

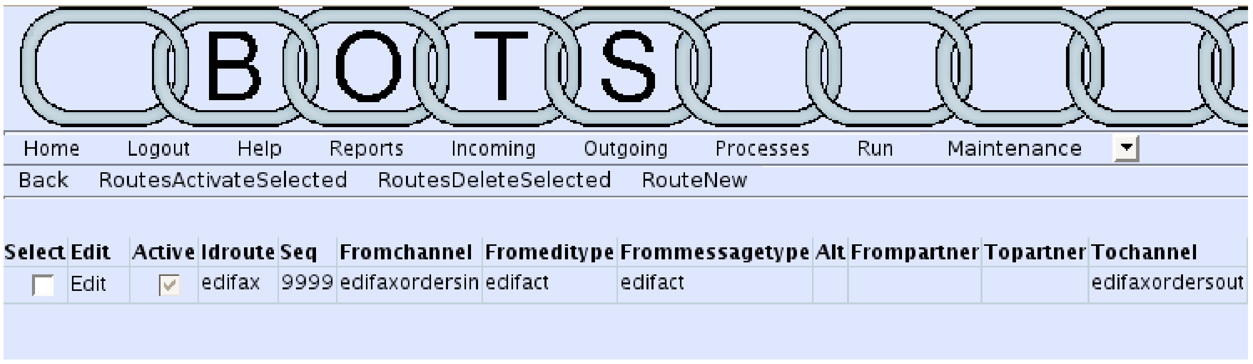


In this picture the route controls:

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

6.1. Configuration of route in bots-monitor

Picture of *bots-monitor->maintenance->routes*:


BOTS
Home Logout Help Reports Incoming Outgoing Processes Run Maintenance
Back RoutesActivateSelected RoutesDeleteSelected RouteNew

Select	Edit	Active	Idroute	Seq	Fromchannel	Fromeditype	Frommessagetype	Alt	Frompartner	Topartner	Tochannel
☐	Edit	✓	edifax	9999	edifaxordersin	edifact	edifact				edifaxordersout

The route is active.
The name of the route is: 'edifax'.
The route gets messages from channel 'edifactordersin'.
Editype is 'edifact', message-type is 'edifact'; this is used to determine the translation.
Translated EDI messages are sent to channel 'edifaxordersout'.

To get this route working, the following must be configured:

- in-channel 'edifactordersin'
- out channel 'edifaxordersout'
- translation for the relevant messagetype and editype, including which mapping script to use

- the mapping script itself
- the grammar for the incoming message
- the grammar for the outgoing message.

6.2. Naming conventions, concepts, descriptions

Concepts used in this overview:

editype	e.g. edifact, x12, csv, SAP idoc, flat file, inhouse, fixed record
messagetype	e.g. ORDERSD96AUNEAN008, 850004010, 'my flat file orders'
channel	controls communication in or out.
translation	convert messages to another format using a mapping script
grammar	a description of the structure, records and fields in an EDI message
mapping script	detailed instructions how to do the translation.

7. Routes.

Doing configuration starts with setting up a route.
Configure a route in: *bots-monitor->Maintenance->Routes*.
Routes use other components: channels, translations, partners.
If you run bots-engine, it processes all active routes.
Routes are independent; EDI messages in one route are not used by other routes.

7.1. Basics

To configure a route, go to *bots-monitor->maintenance->routes.* This chapter leads you through the most important parts of a route. The chapter on 'advanced routes' contains recipes for more complex routes.

7.1.1. Idroute

The name you give the route.

7.1.2. Active

Only active routes are used by bots-engine.

7.1.3. Fromchannel.

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

7.1.4. 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.

7.1.5. Tochannel

The communication channel Bots uses as destination for the translated outbound EDI messages, e.g. FTP, outfile, SMTP, etc.

7.1.6. TestIndicator

Indicates if Bots should process only test edi files, only production files or both to the indicated outchannel.

7.2. Routes - advanced recipes

7.2.1. Receiving more than one editype/edimessage in one channel

1. If the message has a standardized envelope: see chapter “recipes for advanced translations”.
2. See chapter 'Advanced: routing scripts'

7.2.2. Using more than one outchannel/destination

Use field 'seq' (sequence, a number). Several routes can have the same routeID, but a different 'seq'. 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.
Example: receive edifact messages DESADV and INVOIC from one inchannel.
Set it up like this:

Route-id	Seq	fromchannel	fromeditype	frommessagetype	tochannel	toeditype	tomessagetype
Myroute	1	Mypop3	edifact	edifact			
Myroute	2				Desadv-file	Fixed	Desadv
Myroute	3				Invoic-file	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'.

7.2.3. Using partner-specific tochannels in a route

Set it up like this:

Route-id	Seq	fromchannel	fromeditype	frommessagetype	tochannel	topartner
Myroute	1	Mypop3	fixed	orders		
Myroute	2				ftppartnerX	partnerX
Myroute	3				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.

Additionally, partners can be grouped, allowing you to send to/receive from multiple partners in one go.

7.3. 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*
4. bots-engine calls the script's `main()` function.

7.3.1. 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 emailadresses/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 emailadresses/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 emailadresses/partners
   - example of usage:

     communication.run(route='myroute',channel='mypop3')

3. `transform.addinfo(set, where)`
   - change/add information for an EDI file.
   - 'where' specifies what EDI files to change/update; 'where' is a dict.
   - 'set' 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(set={'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:

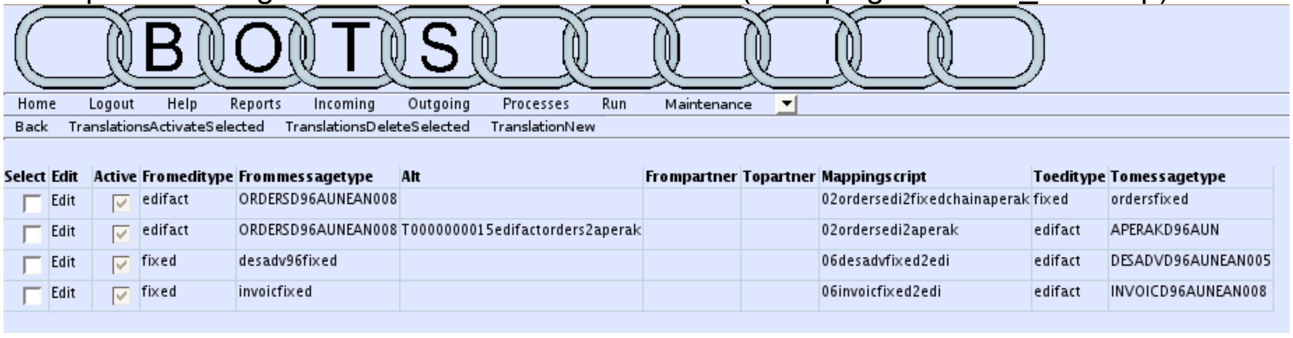
  transform.mergemessages(route='myroute')

7.3.2. *Advice for route scripts*

1. Routes are independent: EDI messages from one route are not used in another route. You can 'overrule' this in route scripts. This is not recommended; let us know if and why you need this.

8. Translations

A translation determines what message-format to convert to what other message-format.
Configure a translation: *Bots-monitor->Maintenance->Translations*.
Examples of configured translations in bots-monitor (from plugin dutchic_d96a.zip):



Select	Edit	Active	Fromeditype	Frommessagetype	Alt	Frompartner	Topartner	Mappingscript	Toeditype	Tomessagetype
☐	Edit	✓	edifact	ORDERSD96AUNEAN008				02ordersedi2fixedchainaperak	fixed	ordersfixed
☐	Edit	✓	edifact	ORDERSD96AUNEAN008	T0000000015edifactorders2aperak			02ordersedi2aperak	edifact	APERAKD96AUN
☐	Edit	✓	fixed	desadv96fixed				06desadvfixed2edi	edifact	DESADVD96AUNEAN005
☐	Edit	✓	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, toeditype and tomessagetype 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.

8.1. Recipes for advanced translations

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/DESADVD96AUNEAN008.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.

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 a grammar file, but it only contains the syntax section.

8.2. Which are the attributes that determine a translation set?

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

8.3. 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 done 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.

Apart from this, 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.

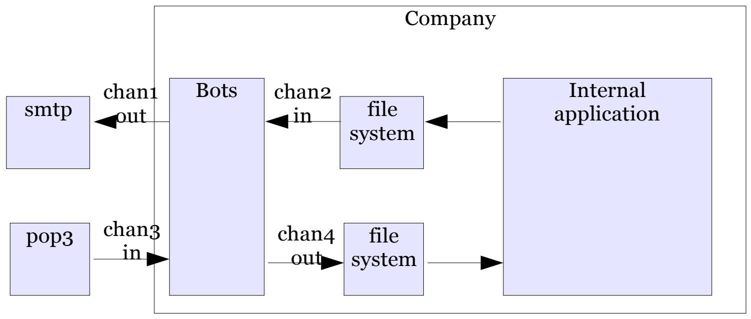
9. Channels (for communication)

Configure a channel: *Bots-monitor->Maintenance->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.
- infile 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, as there are other (open source) packages that do this.

10. Partner related

10.1. 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->Maintenance->Partners.*
2. Configure email details of partner: *Bots-monitor->Maintenance->Email-details.*
   Fill in the partner-id, channel-id and an email address.

One partner can have more than one email address.

10.2. Partner groups

A partner can belong to (one or more) partner groups. This is useful in translations and routing.

10.3. 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>/<partnerid>.py.
This is like a grammar file, but it only contains the syntax section.

11. User maintained code lists

Initialise these in bots/config/bots.ini
Maintenance: *bots-monitor->maintenance->user maintained code lists*.
Use: in mapping scripts, to convert e.g. EAN numbers to internal article numbers.

12. 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.
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 programs: use the power of Python. Normal Python rules apply.
- Incoming message object (inn) has a dict called 'ta_info'. It contains the 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.

12.1. Functions in a mapping script

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

12.1.1. <u>Get-functions: retrieve data from message.</u>

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})
```

12.1.2. 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 this to loop over each and every record, instead 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 is called. In the example above, the root of inn is the UNH-segment.

12.1.3. 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(mycodelistfile,'value')
     (returns the code found via lookup of 'value' in 'mycodelistfile.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 (undetemined).
   - 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(mycodelist,'value')
     (returns the code found via lookup of 'value' in dbtable 'mycodelist').
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, bute when the conversion is not possible is does not raise an error but just returns the (original) value.

12.1.4. 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, botslib.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 exits: no error.

12.1.5. Miscellaneous.

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 is to read a message, adapt somthing (eg code translation) and output it again with the chagnes.
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 is 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 botslib.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'*).

12.2. Mpath.

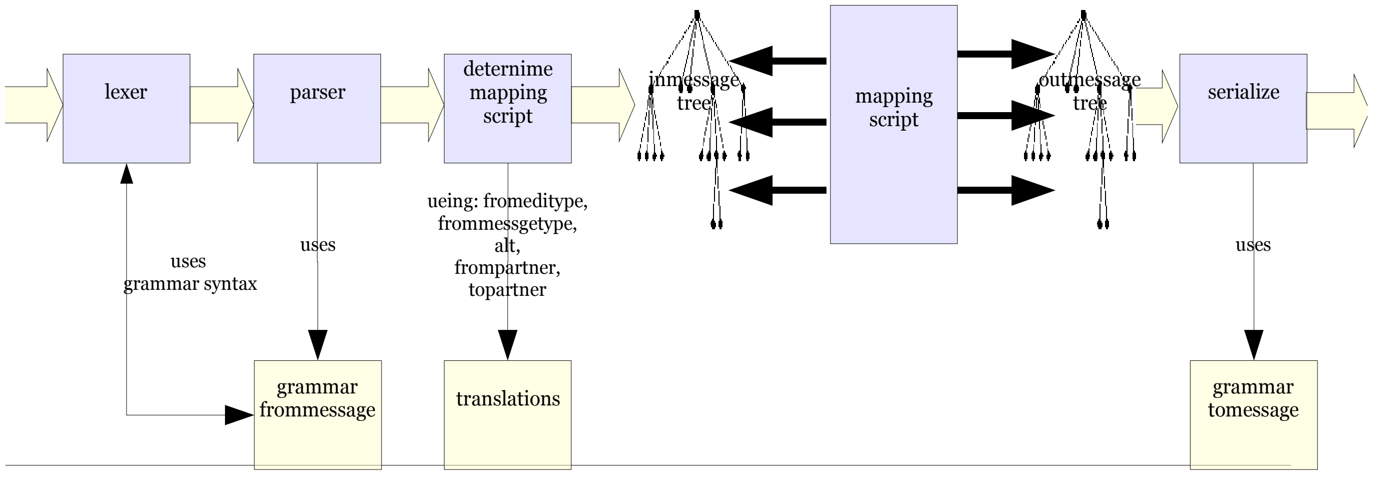
(Lot of text to explain something that's easy to understand by example. Look at the examples!! We think the use of mpaths is quite intuitive if you know EDI).

- 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 (reserved) name for a record identifier/segment tag/XML tag. This is required.
- 'BOTSCONTENT' is the (reserved) name for XML content. This is required.
  Example for XML:
  <plop>krrrr</plop>
  is represented in mpath:
  { 'BOTSID' : 'plop', 'BOTSCONTENT' : 'krrrr' }
- 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.

12.3. How does a mapping script work.

Bots reads an EDI file into memory. The message is lexed and parsed.
The message is then transformed into a tree structure, simular 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 simular 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.

12.4. 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)

13. 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 D96Arecords 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).

13.1. Syntax: 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:

add_crlfafterrecord_sep	*In*: N/A *Out*: put **extra** <CR/LF> after a record/segment separator. Value: string, typically '\n' or '\r\n'
acceptspaceinnumfield	*In*: for fixed formats; do not raise error for numeric field having no contents (but assume value is 0) *Out*: N/A
charset	*In*: charset to use; can be overridden by charset-declaration in content. *Out*: 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.
checkcharsetin	*In*: what to do for chars not in charset. Possible values 'strict' (gives error) or 'ignore' (skip the characters not in the charset). *Out*: N/A
checkcharsetout	*In*: N/A *Out*: what to do for chars not in charset. Possible values 'strict' (gives error) or 'ignore' (skip the characters not in the charset)
checkfixedrecordtooshort	*In* (fixed): warn if record too short. Possible values: True/False, default: False *Out*: N/A
checkunknownentities	*In/Out*: for XML, JSON: skip unknown attributes/elements (instead of raising an error)
contenttype	*Out*: content type of translated file; used as mime-envelope of email
decimaal	*In/Out:* decimal point; default is '.'. For edifact: this is read from UNA-string if present.

envelope	*In*: N/A *Out*: envelope to use; if nothing specified: no envelope - files are just copied/appended.
escape	*In/Out*: escape character used. Defaults: edifact: '?'.
field_sep	*In/Out*: field separator. Defaults: edifact: '+'; CSV: ',' x12: '*')
flattenxml	*In/Out*: write fields of a record as XML-elements (instead of attributes). This is useful for eg straight fixed to xml mappings.
forcequote	*In*: N/A *Out*: for CSV: Possible values: 0: (default) quote only if necessary; 1: always use quote
startrecordID	*In (*fixed): start position of record ID; value: number, default 0. *Out*: N/A
endrecordID	*In*(fixed): end position of record ID; value: number, default 3. *Out*: N/A
merge	*In*: N/A *Out*: merge translated messages to one file (for same sender, receiver, messagetype, channel, etc). Related: envelope. Possible values: True; False
noBOTSID	*In/Out* (CSV): use if records contain no real record ID/tag
output	*In*: N/A *Out*: (template) values: 'xhtml-strict'
quote_char	*In/Out* (CSV): char used as quote symbol
record_sep	*In/Out*: char used as record separator. Defaults: edifact: "'" (single quote); fixed: '\n'; x12: '~'
reserve	*In/Out* (edifact,x12): repetition separator; not yet in use; generated in UNA
sfield_sep	*Out* (edifact,x12): subfield separator. Defaults: edifact:':', x12:'>'
skip_char	*In*: 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.) *Out*: N/A
skip_firstline	*In* (CSV): skip first line (often contains field names). Possible values: True/False, default: False
template	*In*: N/A (template): XML/XHTML template to use for HTML-output.
triad	*In*: 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). *Out*: N/A
version	*In* (edifact, x12) not used, is read from envelope fields. *Out* (edifact,x12): version of standard generate. Value: string, typically: '3' or '004010'

13.2. Structure: sequence of records in 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.

13.3. 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 to 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

13.4. 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).

13.5. nextmessageblock: split an CSV file into messages

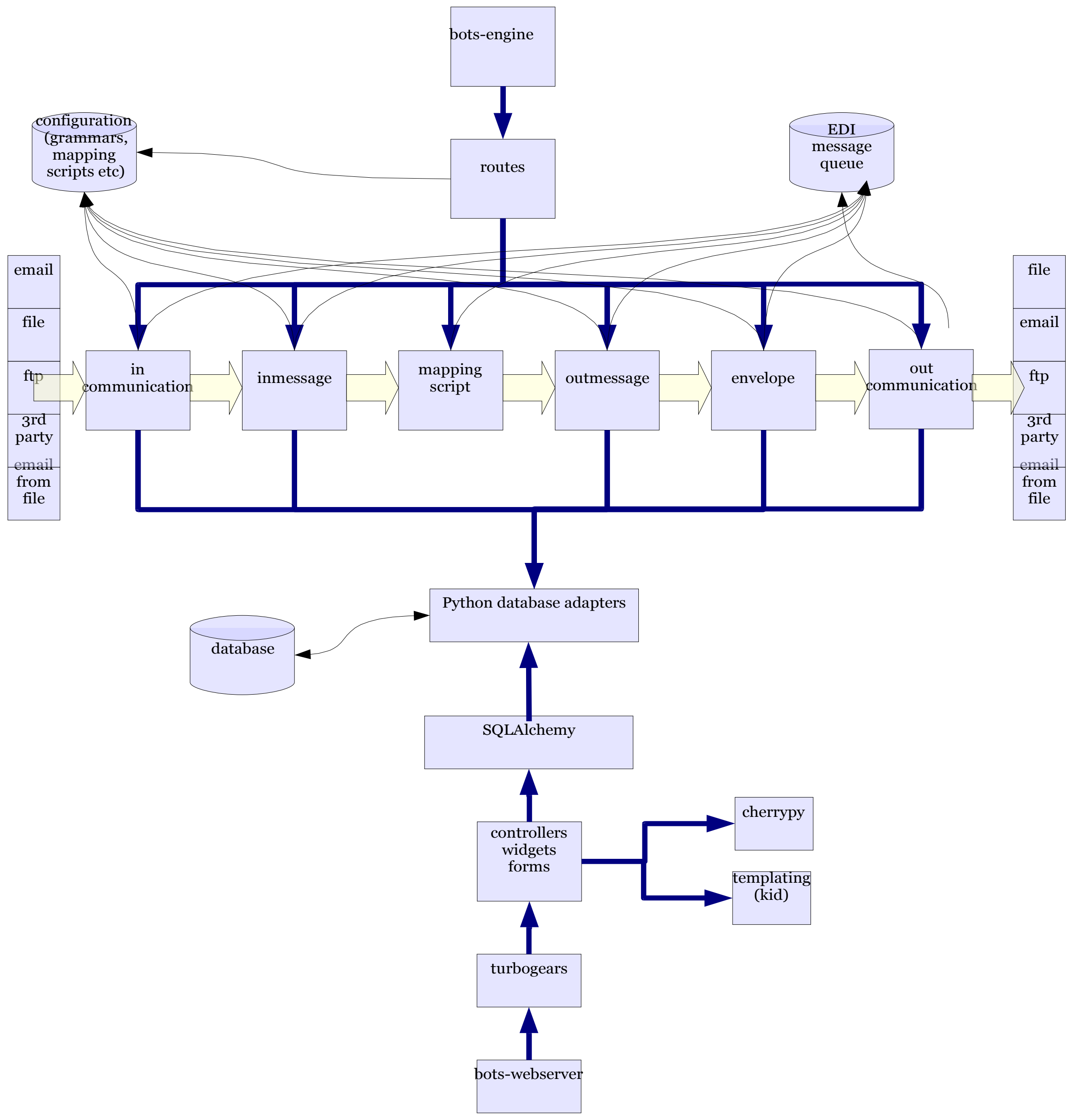
'nextmessageblock' is used for splitting the messages in an EDI file with a single type of record, typically of editype CSV. Nextmessageblock is an mpath used by get().
Bots-engine uses this mpath to retrieve a value from each record; all subsequent records with the same value are regarded as belonging to the same message, and passed as one message to the mapping script. When Bots entcounters records with a different value, Bots assumes this is 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})

14. Addendum: technical schema for bots

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


bots-engine
routes
configuration (grammars, mapping scripts etc)
EDI message queue
email
file
ftp
3rd party
email from file
in communication
inmessage
mapping script
outmessage
envelope
out communication
file
email
ftp
3rd party
email from file
Python database adapters
database
SQLAlchemy
controllers widgets forms
cherrypy
templating (kid)
turbogears
bots-webserver