documentation for bots

Release 3.2.0

hjebbers

2017-06-10

Contents

- **Bots** is fully functional software for EDI (Electronic Data Interchange).
- All major EDI data formats are supported: EDIFACT, X12, TRADACOMS, XML.
- Runs on Windows, Linux, OSX and Unix.
- **Bots** is very stable.
- **Bots** handles high volumes of edi transactions.
- **Bots** is flexible and can be configurated for your specific EDI needs.
- Read about the features of bots and the latest news

CHAPTER 1

First steps

1. *Installation*
2. *Start bots up*
3. Get your first configuration running: *Quick Start Guide*
4. Check out some *plugins*.

CHAPTER 2

Other info on bots

- Sourceforge website.
- Active mailing list.
- Bots is licenced under GNU GENERAL PUBLIC LICENSE Version 3.
- Commercial support by EbbersConsult.

CHAPTER 3

It's hard to get started

Often people experience a steep learning curve when starting with edi. A lot of knowledge is involved:

- edi standards (edifact, x12, tradacoms, EANCOM etc)
- business processes between you and your edi-partner (logistics!), changes in the business processes
- understand what your edi-partner wants/requires
- edi communication methods (x400, VAN's, AS2 etc)
- imports and exports of your ERP system
- specifics of the edi software.
- etc

It is hard to find good information about edi: standards are not always free (eg x12 is not free), decent example messages are hard to get and often if is hard to find good information on Internet. Edi is traditionally 'closed' and sparse with information. Partly this seems to be a 'cultural thing', partly because edi existed before Internet, partly because it is all about business data that is not for the general public.

Don't give up ;-)) I think everybody who started with edi has gone through this.

Install

Bots works on operating systems with python installed. Confirmed is:

- windows (windows10, XP, Vista, windows7, Server 2008, Server 2012, etc)
- apple OS.X
- linux debian (ubuntu, mint, etc)
- linux Red hat (Centos. Fedora)
- OpenSolaris
- FreeBSD

- AIX

Let us know if it runs (or not) on another OS.

Windows

1. **Install Python**
   (a) Check if Python is already installed.
   (b) Use python 2.7; Python >= 3.0 does not work.
   (c) Download Python installer.
   (d) Install python (double-click).
2. **Install bots**
   (a) Download bots installer.
   (b) Install bots (double-click).
   (c) Installation takes some time; be patient. During the installation the libraries bots needs are installed.
   (d) You will be notified if the installation went OK.
   (e) If not: contact is via the mailing list.

Note:

1. Mind your rights. Both Python and Bots need to be installed as admin (windows vista/7/8/10).
2. The windows installer includes all dependencies for standard installation. Some extra dependencies are needed for less used functions (eg. extracting data from excel or pdf files).

---

*nix installation

There is no `*.deb` or `*.rpm` for bots - would be great if you have experience with this and want to give some help. So a standard python source code install is done.

1. **Install Python**
   (a) Check if Python is already installed - most of the time python is already installed on `*nix`. Use python 2.6 or 2.7. (not python >= 3.0).
   (b) If not: use package manager or see python web site.
2. **Install dependencies/libraries**
   - See *list of dependencies*.
   - Easiest is to use your package manager for installing.
3. **Install bots**
   (a) Download bots installer (e.g. bots-3.1.0.tar.gz)
   (b) Unpack (command-line): `tar bots-3.1.0.tar.gz`
   (c) Go to created directory (command-line): `cd bots-3.1.0`
   (d) Install (command-line): `python setup.py install`

(e) Postinstall: depending on what do want: change rights for directories botssys, usersys and config or place these elsewhere and make symbolic links in the bots installation directories.

Note: Place the directories botssys, usersys and config somewhere else (out of /usr), change the owner/rights and make symbolic links in the bots installation to these directories.

Installation from scratch

Installation on amazon EC2, looks like red hat version of linux (Note that versions might not be correct anymore)

```
#install django
$ wget -O django.tar.gz https://www.djangoproject.com/download/1.4.13/
↪tarball/
$ tar -xf django.tar.gz
$ cd Django-1.4.13
$ sudo python setup.py install
$ cd ..
#install cherrypy
$ wget http://download.cherrypy.org/CherryPy/3.2.2/CherryPy-3.2.2.tar.gz
$ tar -xf CherryPy-3.2.2.tar.gz
$ cd CherryPy-3.2.2
$ sudo python setup.py install
$ cd ..
#install Genshi
$ wget http://ftp.edgewall.com/pub/genshi/Genshi-0.7.tar.gz
$ tar -xf Genshi-0.7.tar.gz
$ cd Genshi-0.7
$ sudo python setup.py install
$ cd ..
#install bots
$ wget -O bots-3.1.0.tar.gz http://sourceforge.net/projects/bots/files/bots
↪%20open%20source%20edi%20software/3.1.0/bots-3.1.0.tar.gz/download
$ tar -xf bots-3.1.0.tar.gz
$ cd bots-3.1.0
$ sudo python setup.py install
$ cd ..
#set rigths for bots directory to non-root:
$ sudo chown -R myusername /usr/lib/python2.6/site-packages/bots

#start up bots-webserver:
$ bots-webserver.py
```

Installation from scratch (bots2.2)

Installation on vanilla CentOS6.2 (logged in as root) (Note that versions might not be correct anymore):

```
#install django
wget http://www.djangoproject.com/download/1.3.1/tarball/
tar -xf Django-1.3.1.tar.gz
cd Django-1.3.1
python setup.py install
cd ..
#install cherrypy
wget http://download.cherrypy.org/CherryPy/3.2.2/CherryPy-3.2.2.tar.gz
tar -xf CherryPy-3.2.2.tar.gz
cd CherryPy-3.2.2
python setup.py install
cd ..
```

```
#install Genshi
wget http://ftp.edgewall.com/pub/genshi/Genshi-0.6.tar.gz
tar -xf Genshi-0.6.tar.gz
cd Genshi-0.6
python setup.py install
cd ..
#install bots
wget http://sourceforge.net/projects/bots/files/bots%20open%20source%20edi
↪%20software/2.2.1/bots-2.2.1.tar.gz/download
tar -xf bots-2.2.1.tar.gz
cd bots-2.2.1
python setup.py install
cd ..

#start up bots-webserver:
bots-webserver.py
```

Dependencies

Always:

- Needs: python 2.6/2.7. Python >= 3.0 does not work.
- Needs: django >= 1.4.0, django <= 1.7.0
- Needs: cherrypy > 3.1.0

Optional:

- Genshi (when using templates/mapping to HTML).
- SFTP needs paramiko and pycrypto. Newer versions of paramiko also need ecdsa.
- Cdecimals speeds up bots. See website
- bots-dirmonitor needs:
  - pyinotify on `*nix`
  - Python for Windows extensions (pywin) for windows
- xlrd (when using incoming editype 'excel').
- mysql-Python >= 1.2.2, MySQL (when using database MySQL).
- psycopg2, PostgreSQL (when using database PostgreSQL).

Install FAQ

I try to install bots at Windows 7/10, but.....

- Probably a rights problem - you'll have to have administrator rights in order to do a proper install.
- Right click the installer program, and choose 'Run as Administrator'.
- sometimes the shortcut is not installed in the menu, and you will have to make this manually.

Does bots have edifact and x12 messages installed out-of-the-box? No. But this can be downloaded on the sourceforge site either as part of a working configuration (plugin) of separate (grammars).

Bots is not working on linux - rights problems. Did you start bots-webserver and/or bots-engine with sufficient rights - e.g. as root. Change the owner/rights of the files in botssys, usersys and config; run bots-webserver/bots-engine without root rights.

Error during windows installation

```
close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr
```

- seems to happen when UAC is turned off.
- Actually bots just seems to be installed OK, and works OK.....
- Fixed this in version 3.2

Get Bots Running

Main components of bots

1. **Bots-monitor:**
   - The user interface or the GUI.
   - This is a web interface and runs in a web browser like Firefox, Chrome, or Internet Explorer.
   - Bots uses web technology for the interface - but bots does NOT communicate to the internet for this. All is on your local computer.
   - Bots-monitor can be accessed from all workstations in your LAN.

   Warning:

   out-of-the-box bots-monitor uses plain HTTP and is not secure. Advised is either:
   - do not use bots-monitor over a public network (such as Internet)
   - secure the connection using *HTTPS/SSL*.

2. **Bots-webserver:**
   - Program that serves web pages to bots-monitor.
   - The bots-webserver has to run in order to use bots-monitor.
3. **Bots-engine:**
   - This program does the actual edi communication and translation.
   - Bots-engine does the communications and translations (of eg edifact or x12).
   - 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.

Start bots-monitor (using bots-webserver)

1. **Start bots-webserver (several options):**
    - When bots is installed using with Windows installer use the 'shortcut' to Bots-webserver in your 'Programs' menu.
    - (*nix) Command line: `bots-webserver.py`
    - (Windows, python 2.7) go to command line and: `c:\python27\python c:\python27\Scripts\bots-webserver.py`
2. Bots-webserver should stay running (and not disappear). If not, see *Start-up FAQ*.
3. **View using your Internet browser**
    - When bots-webserver runs on the same computer, use address: http://localhost:8080
    - use Firefox, Chrome, Opera or Internet Explorer. Bots does NOT support Internet Explorer 6. Issues have been reported with IE8, but for some IE8 does work.
    - When accessing bots-monitor over your LAN (bots-webserver runs on another computer) the IP address or DNS name of that computer, e.g.: `http://192.168.10.10:8080`.
4. Default login: user name `bots`, password `botsbots`.
5. Tip: add `bots` to your favorites/bookmarks.

Start bots-engine

There are several ways to start bots-engine:

1. (windows, *nix) Start from bots-monitor: `bots-monitor->Run->Run (only new)`
2. (*nix) Command line: `bots-engine.py`
3. (Windows, python 2.7) go to command line and: `c:\python27\python c:\python27\Scripts\bots-engine.py`

The results of what bots-engine has done can be viewed in the bots-monitor.

Note: if you did not configure of bots to do something, the bots-engine will run but will not do much. To get bots to do something see *Quick Start Guide*.

Start FAQ

When starting bots-webserver the window disappears after a few seconds? Start the bots-webserver from the command line; you will be able to see what goes wrong. (Windows, python 2.7) go to command line and: `c:\python27\python c:\python27\Scripts\bots-webserver.py` For the most common cause for the problem see the next question.

Bots-webserver gives error: IOError: Port 8080 not free on 'x.x.x.x' (or similar).

Another program already uses this 'port'.

Adapt the port bots uses: in configuration file bots/config/bots.ini look for 'port'.

Change port to eg 8090

Start bots-webserver again.

In your browser you will have to indicate another port eg: http://localhost:8090

Can I run multiple instances of bots-engine in parallel?

No, this is not possible.
Instead bots >= 3.0 has better control of running the engine: *jobqueue server*.

Quick Start Guide

- Purpose: get your first edi configuration running.
- This is done by installing plugin `my_first_plugin`; this plugin provides a working configuration.
- When run, this configuration will read and write example edi messages (provided in the plugin) from your system.
- In this configuration incoming edifact orders are translated to a fixed file format.

Install plugin ''my_first_plugin''

1. **Assumed is:**
    - You've installed bots, see *Install*
    - You've managed to get bots-monitor running, see *get bots running*.
2. Now download and install plugin `my_first_plugin`.
3. Instructions for installing a plugin are *here*.
4. The plugin can be downloaded from the bots sourceforge site.

Activate the route

1. go to the routes screen: `bots-monitor->Configuration->Routes`
2. note that route `myfirstroute` is not active now (indicated by red icon)
3. select the tick-box in front of the route `myfirstroute`
4. select action `activate/de-activate`
5. click on the 'Go'-button behind the selected action.
6. note that route `myfirstroute` is active now (indicated by green icon)

Run the translation

1. Run the translation: `bots-monitor->Run->Run (Only New).`
2. You will get notified that the `bots-engine` is started.
3. Bots-engine is the part of bots that does the translations and communications; it runs in the background. Bots-engine will be finished in approximately one second.

View the results

Now let's view the results of the translation:

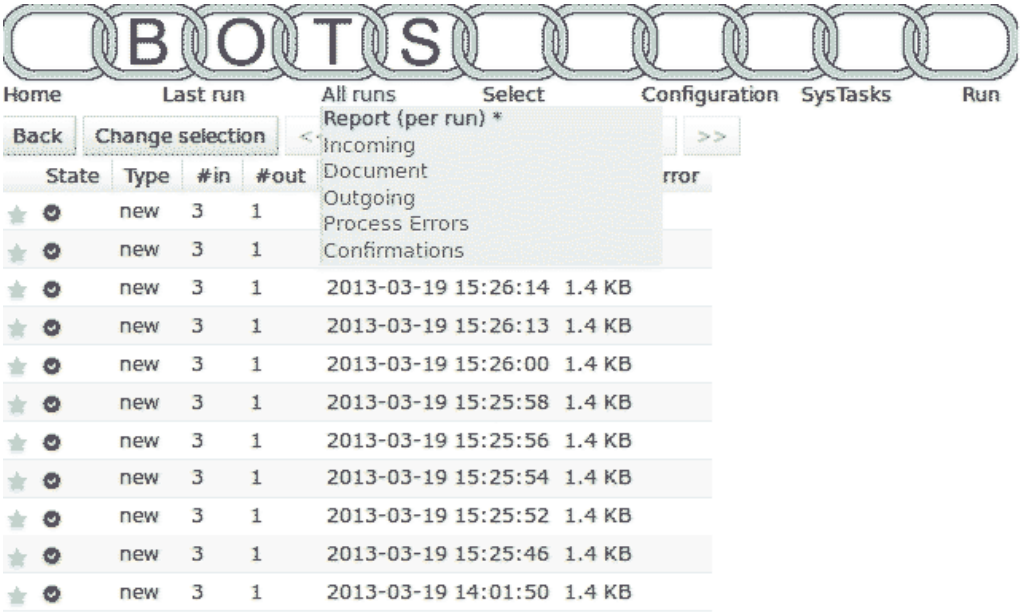
1. First look at the results of the run: `bots-monitor->All runs->Reports (per run).` Each run of bots is represented by a line; the last run is on top.
2. View the incoming files via `bots-monitor->Last run->Incoming`. Click on the incoming file to see its contents.
3. View the outgoing files (the results of the translation) go to or `bots-monitor->Last run->outgoing`. Again: click on the file name to see its contents.

4. Note: this configuration reads the incoming files but does not delete them. So you can run it over and over again.

View Results

View runs

For each run of bots-engine you can see the results in `bots-monitor->All runs->Reports (per run):`

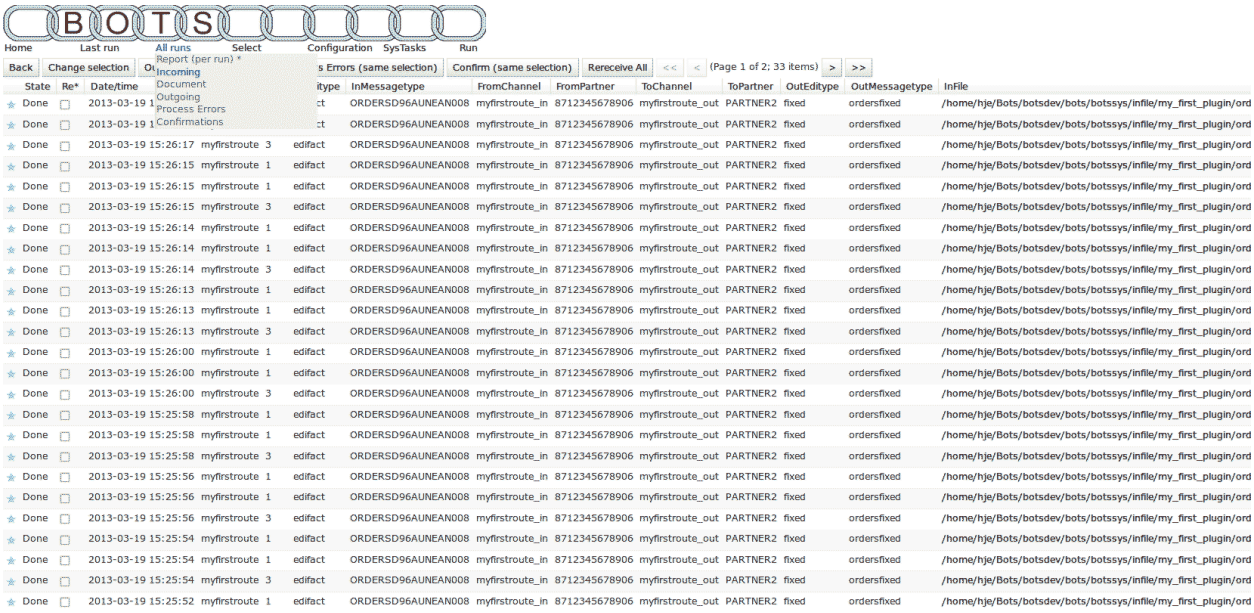


If you go to the star in front of a line, a drop-down menu appears where you can zoom in on the results of the run.

Incoming

Incoming files of the last run can be viewed via `bots-monitor->Last run->Incoming`:

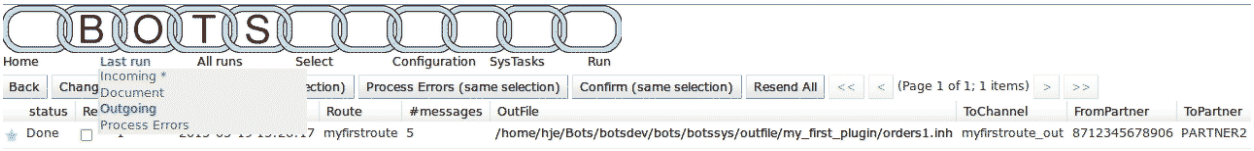
View all incoming files via `bots-monitor->All runs->Incoming`:



By using the 'Change selection' button you can change the selection criteria for the view. By clicking a filename you can view the contents of that file.

Outgoing

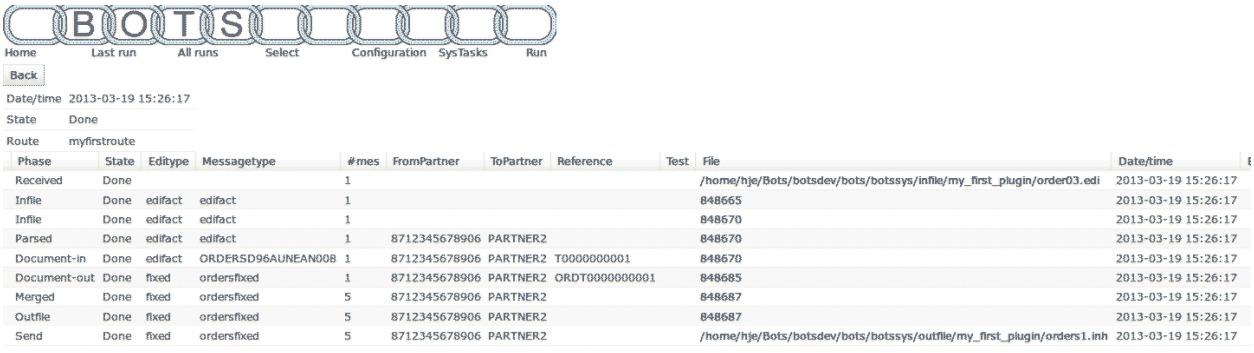
The outgoing files of the last run can be viewed via `bots-monitor->Last run->Outgoing`:



By clicking a filename you can view the contents of that file.

Detail Screen

When you are in the `incoming` screen and go to the star in front of a line, a drop-down menu appears where you can zoom in on the details of the processing of the incoming file:



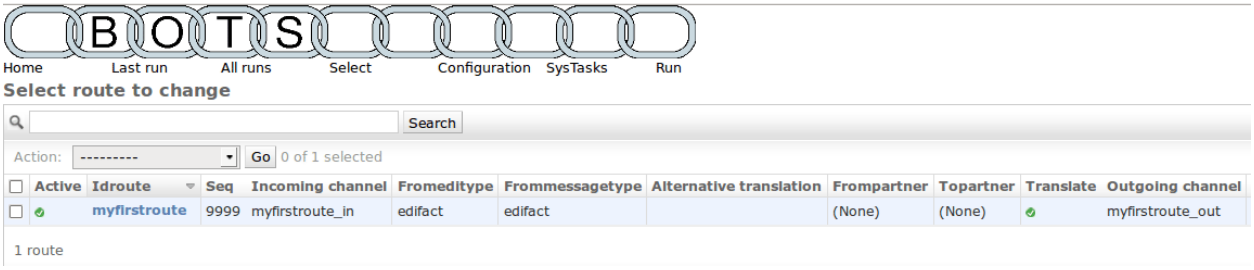
Here the detailed steps in processing an incoming file are shown.

Walk through setup

Lets take a look at the setup in `my_first_plugin`.

Look at the configured route

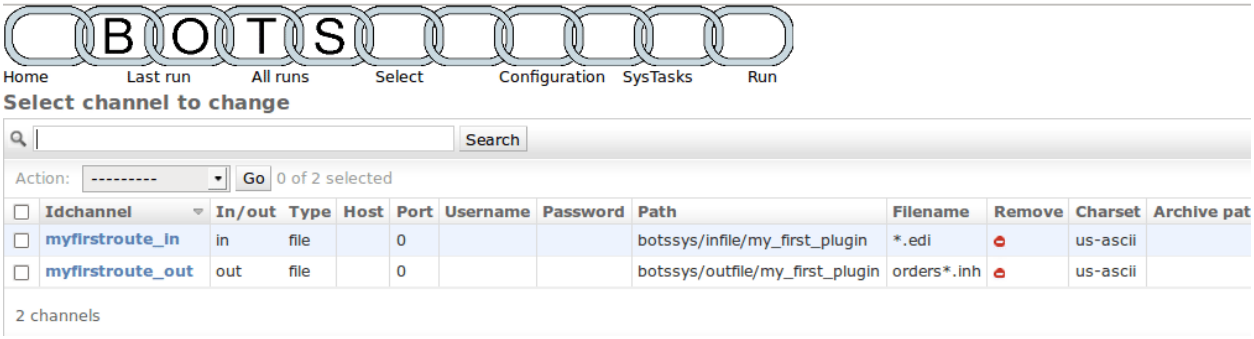
To view the route configured: `bots-monitor->Configuration->Routes`. This will look like:



- A route tells bots what to do: where to get the edi-files, what type of files these are (this determines the translation done), and where to put the translated edi-files..
- One route is configured, called `myfirstfoute`.
- The route uses communication channel `myfirstroute_in` to get incoming edi-files.
- These are edifact format: fromeditype=edifact, frommessagetype=edifact. Bots will figure out the exact messagetype (like ORDERSD96AUN) by itself.
- The translated edi-files (fixed format) go to communication channel `myfirstroute_out`.

View the communication channels

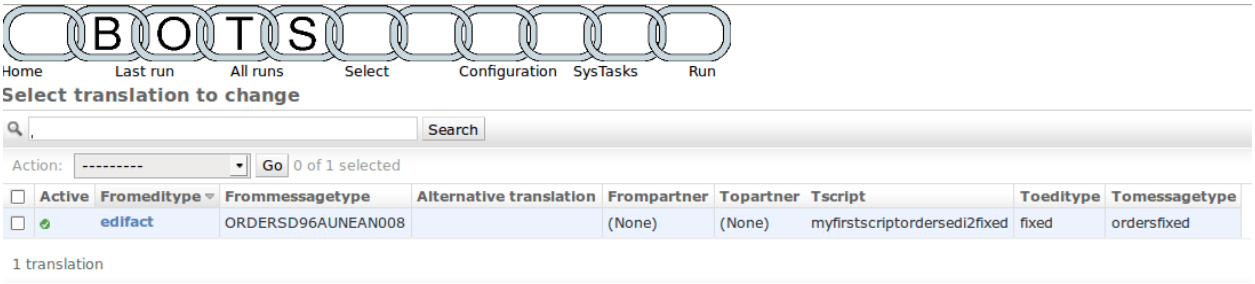
To view the communication channels configured: `bots-monitor->Configuration->Channels`. This will look like:



- A communications channel communicates edi-files in or out of bots.
- There are different types of channels, eg: file, ftp, smtp, pop3, etc.
- In this plugin 2 routes are configured. Both are type `file`: all reading and writing is to file system.
- There is one `in`-channel and one `out`-channel.
- Channels for file-system require a path and a filename.

The translations in this configuration

To view the translations configured: `bots-monitor->Configuration->Translations`. This will look like:



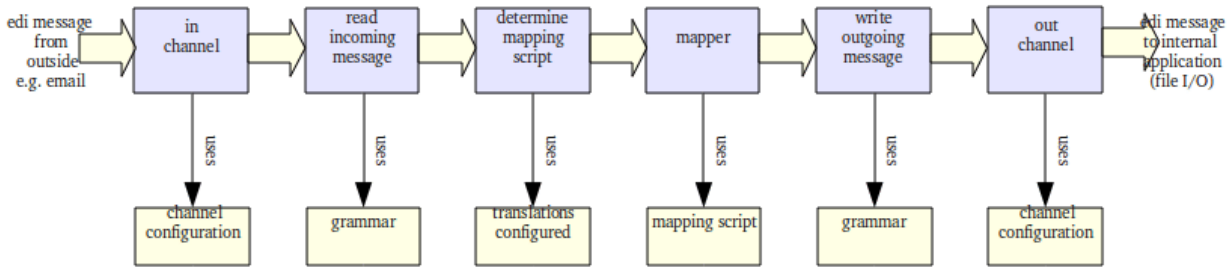
- There is one *translation* configured.
- This translation translates edi messages of editype `edifact` and messagetype `ORDERSD96AUNEAN008` using mappingscript `myfirstscriptordersedi2fixed` to edi messages of editype `fixed` and messagetype `ordersfixed`.
- Each messagetype has a *grammar* which describes the message: records, fields, formats. The grammar is a file; you can find it in: `C:\Python27\Lib\site-packages\bots\usersys\grammars\fixed`
- The *mapping script* does the actual translation; basically it gets data from the incoming message and puts the data in the outgoing message.
- The mapping script is a file; you can find it in: `C:\Python27\Lib\site-packages\bots\usersys\mappings\edif`

What happened

How does this ‘‘run‘‘ thing work?

1. When you run bots-engine, each active route is run.
2. Per route bots will perform all configured actions (read, translate, write):
3. In bots-monitor you can view the results of the run(s), the incoming edi-files, the outgoing edi-files, etc.

Detailed explanation



All actions in the picture take place in the route `myfirstroute`. The route contains:

- an in-channel - fetches the incoming orders.
- in the route is indicated that it should translate
- an out-channel - transports the in-house orders to your file system.

When the incoming files have been read via the in-channel, bots starts to translate:

- bots parses the incoming file, using the information in the route: editype=edifact, messagetype=edifact.
- as edifact is a standard bots can find out itself that the incoming edifact file contains order messages of messagetype ORDERSD96AUNEAN008

- bots looks in the translation table (see `bots-monitor->Configuration->translations`) to find out what to do: what mapping script to use, to what editype and messagetype should be translated. In this case the mapping script `myfirstscriptordersedi2fixed` translates to editype **fixed**, messagetype **ordersfixed**.
- the mapping script is the heart of the translation. In the mapping script the data from the incoming message is fetched and place into the outgoing message.

A complete translation in bots needs:

- Configure of the translation (`bots-monitor->Configuration->Translations`).
- A grammar for the incoming message. A grammar describes an edi-message: the records, sequence of the records, fields in the records, field lengths etc.
- A mapping script. The mapping script gets data from the incoming message and puts it in the outgoing message. A mapping script is a Python script. You do not need to be proficient in Python to do this; only the basics of Python are used. And Python is a relatively easy computer language. There are a lot of good examples of mapping scripts in the plugins.
- A grammar for the outgoing message.

Guide For Bots-Monitor

Note: See also The *Quick Start Guide* (includes screen-shots of bots-monitor).

Menu in bots-monitor

1. **Home: start page and general information.**
2. **Last run: view results of the last run:**
    - incoming: view incoming files and results
    - document: view status of business document. An edi file can contain multiple documents (eg orders). In this view are the results of the separate business documents. See *setting document views*
    - outgoing: view outgoing files and results
    - process errors: view process errors; mostly these will be communication errors.
3. **All runs: view results of all runs:**
    - reports: view of all runs and results
    - incoming: view incoming files and results
    - document: view status of business document. An edi file can contain multiple documents (eg orders). In this view are the results of the separate business documents. See *setting document views*
    - outgoing: view outgoing files and results
    - process errors: view process errors; mostly these will be communication errors.
    - confirmations: view the results of confirmations you wanted and confirmation you gave. See *setup confirmations*
4. **Select: use criteria like date/time to view results you want to see like eg editype edifact or x12.**
    - Note: select screens can also be used using 'select' button in other views.
5. **Configuration: configuration of the edi setup.**

6. **System tasks (administrators only): read plugins, create plugins, maintain users, etc.**
7. **Run: manually start a run of bots-engine:**
    - Run (only new): receive, translate, and send new edi messages.
    - Run userindicated rerecieves: receive previously received edi-files again from archive. User has to mark edi-files as 're-receive' via incoming view.
    - Run userindicated resends: resend previously send edi-files again. User has to mark edi-files as 're-send' via outgoing view.

User interface tips

- View screens often have a star at the beginning of each line; moving over the star will show possible actions.
- In view screens, you can see the contents of an edi file if you click on the file name.
- When viewing the contents of an edi file, you can go backwards and forwards to see the processing steps of the file.
- Might be handy to use tabbed browsing.
- Bots uses user rights (viewers, administrators and superuser). See *setting user rights*

How to Re-Receive

- Incoming files can be `re-received`.
- When re-receiving the (previously received) file is taken from the archive, and re-processed.
- As the file from the archive is used, no (outside) communication is used.

Steps

- In the incoming screen, go to the start (in front of each line); a small menu will pop-up. Choose 'rereceive' from this menu. The file will be marked as 'rereveive'. Mark all files that you want to re-receive.
- Go to main `menu->Run->Run user-indicated re-receives`.
- The engine will now start, and do all the re-receives you marked.

Re-receive many files

- As the files need to be marked one by one, this can be troublesome when you need to re-receive many files.
- In the screen for incoming there is a button called **Rereceive all**. When you use this button all files in the selection are marked as re-receive. This might take some while.
- Warning: all files in the selection are marked, not only the ones visible on your page.

Note: Tip: first make the right selection via the button 'Change Selection'. Additional individual files can be marked/de-marked later.

Configure bots

- Out of the box bots does nothing. You have to configure bots for your specific edi requirements.
- Check out different ways to start your own *configuration*.
- See *debug* overview for info how to debug while making a configuration.

- Bots also has nice features for *configuration change management* (build test sets for you configuration, easier pushing of changes from test to production).

Configuration explained in short

- `routes` are edi-workflows.
- `channels` do the communication (from file system, ftp, etc).
- each route hs an `inchannel` and an `outchannel`
- Translations rules determine: translate what to what.

Most asked configuration topics

- *composite routes*
- *passthrough route* (without translation)
- *options for outgoing filenames*
- *direct database communication*
- *partner specific translations*
- *code conversion*
- view *business documents* instead of edi-files.
- *confirmations/acknowledgements*
- *merging and enveloping outgoing edi files*
- *partner specific syntax* (especially for x12 and edifact)

How to do configuration

How do I make an edi configuration?

1. **Use *plugins*. This is a quick and easy way to install predefined configurations of bots.**
    - Downloads: bots sourceforge site
    - Description of plugins: *on this wiki page*
    - Plugins can be shared by users and communities. Share your plugin!
2. Use a plugin *close* to what you want, and adapt.
3. **Do your own configuration:**
    - Take a look at the *Quick Start Guide*.
    - Use bots-monitor for configuration of routes, channels, translations and partners.
    - There are grammars for all edifact and x12 messages on the bots sourceforge site.
    - Grammars and mapping require knowledge of edi and edi-standards and some python knowledge. (python is a relatively easy programming, no deep knowledge of python is needed).
    - Use command line tool `bots-xml2botsgrammar.py` to generate a grammar from an xml file.
    - Check a grammar using the command line tool `bots-grammar.py`.
4. A step by step description (next section) of making a configuration.
5. If you are doing your own configuration check out how to *debug*

DIY step by step

This is how I start with a new route and translation...

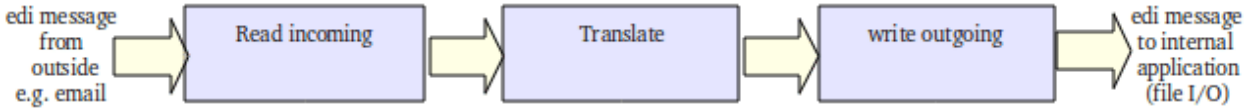
- **Create some *channels* (eg. test_in and test_out).**
  - These are just "file" channels that read from botssysinfile(some-dir) and write to botssysoutfile(some-dir).
  - You should have some sample input documents from your partner, put one or a few of these in the infile location.
  - Make sure the test channel is NOT set to delete them.
- **Get (or create) the incoming *grammar*.**
  - If incoming is edifact or X12 then this part is already done for you, download the grammars from the bots website.
  - For XML there is a conversion tool to make a start.
  - For fixed/csv etc. you need to do it yourself, it is best to start with one of the plugin examples.
  - Creating a fully working correct grammar can be the most difficult part of setting up bots, but is a once-off task.
  - From then on that edi type can be used in any route.
- **Configure a *route* to get files from your inchannel and specify the editype and messagetype corresponding to your gramma**
  - You can use the same editype for outgoing too for now.
  - Don't worry about translation yet, just run bots engine, check errors, and keep adjusting your grammar and testing iteratively until the file is "received" ok.
  - It will then be "stuck" because there's no translation.
  - If you get syntax errors when using a downloaded grammar, most likely the input file actually contains errors. Contact your partner about this, or correct them by editing the file and continue testing.
- As you did for incoming, create your outgoing grammar and configure it in the route. If you are creating your own grammars try at first to keep them simple with no mandatory segments etc. This can be added later.
- **Configure a *translation* and create a *mapping script*.**
  - Start with mapping just the bare minimum mapping one or two fields (eg. order number) and again iteratively modify and test until you have some output.
  - if your output grammar is edifact or X12 for example, you'll need to at least map the mandatory segments to create a valid document.
- When testing a mapping, it's very useful to insert print statements to help with *debugging*. The output will be seen in the webserver console window if you run bots engine from the GUI menu.
- Once you can run the route with no errors and get "something" output you'll feel that sense of achievement and can then go on to add everything else you need into the mapping and grammar, piece by piece.
- If you have many mappings to do, create a module of common functions you create, and import into every mapping.
- Also check out the bots built in mapping functions provided. The code conversion tables are particularly useful.
- The channels are the final part once it's all working and tested, to read and write from the actual systems involved. Create the new channels and change the route to use them.

I would say the learning curve is a little steep at first, but once started you'll be glad you did.

Routes

Definition: A route is a workflow for edi-files

- A route determines where to get the incoming files, what to do with these edi-files (eg translate) and their destination.
- `Routes` are the most important concept in configuring bots.
- Routes are independent: an edi-file in a route stays in that route.
- Route are configured in `bots-monitor->Configuration->Routes`.



To get a route like this working the following must be configured:

- The route itself in `bots-monitor->Configuration->Routes`.
- An in-channel for incoming edi files.
- An out-channel for outgoing edi files.
- The *translation*.

The route above is a simple route; files come from one source, there is one translation, one destination. More options are possible using *composite routes*.

Composite routes

A simple route reads the edi files from one inchannel, translates, and sends the translated files to one outchannel. More flexibility is offered by the use of composite routes. In a composite route there are several entries in the routes screen, each with the same **idroute** but a different **seq** (sequence number within route). Each entry (with different **seq**) is called **route-part**. Best way to use this is to have each route-part do any one of the things:

- Fetch incoming files; you can fetch files from multiple sources
- Translate (typically once per route)
- Send outgoing files to different destinations/partners using *filtering*

It is advised to set up composite routes this way (using at least 3 route-parts).

Use cases

- Send edi files via ftp where each partner has its own ftp-server.
- Confirmations/acknowledgements: acknowledgements for incoming edi-files are routed back to the sender (filter by editype/messagetype).
- Fetch from multiple sources, eg ftp-servers of different partners..
- Route to different internal destinations: invoices to another system than ASN's (filter by messagetype)
- Use a VAN, but one partner uses AS2 (filter by partner)
- Incoming files are translated multiple times, each message-type goes to different destination. Eg: translate orders both to in-house file (import ERP) and an HTML-email (for viewing).

Example plugin

Download the plugin demo_composite_route This plugin has one composite route consisting of:

- 2 input parts
- Translate part
- 3 output parts, using *filtering*

Detailled description here.

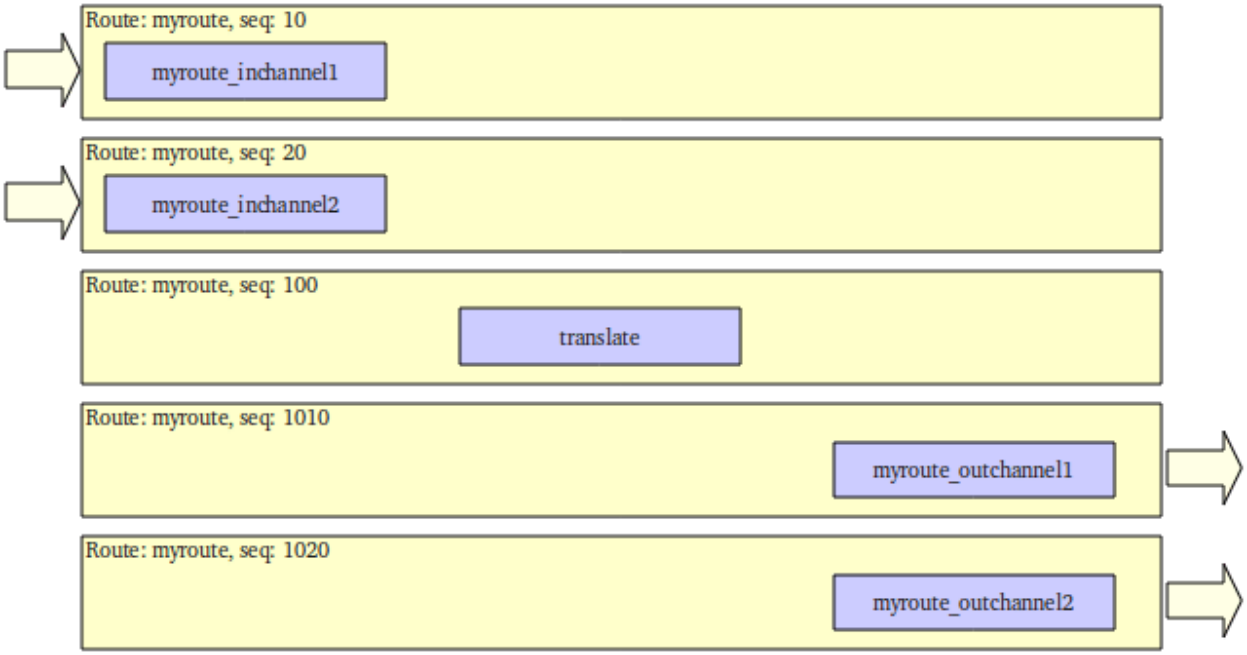
Filtering for different outchannels

- You can filter per outchannel; eg send only asn's through this outchannel.
- In route-screen (`bots-monitor->Configuration->Routes`) the fields used for filtering under **Filtering for outchannel**.
- If eg toeditype=csv, only csv-files will be send over the outchannel.

Note: If filtering is not specified, all outgoing files in the route are send through the outchannel.

Schematic

Schematic overview of a route consisting of 5 parts:



Note:

- If no inchannel in route-part nothing comes in for that route-part.
- If 'translate' in a route-part is off, no translation in that route-part.
- If no outchannel in route-part nothing goes out for that route-part.

PassThrough (no translation)

Sometimes you want to pass an edi-file though bots without translation. In this case bots is only used to manage/register the sending or receiving of edi-files.

For bots>=3.0

In route (`bots-monitor->Configuration->Routes`) use value **Pass-through** for **translate**.

For bots<=2.2

Use a routescript:

1. Configure route the normal way (bots-monitor->Configuration->Routes)
2. Make a routescript with the same name as the routeID
3. Place the routescript in bots/usersys/routescripts/routeid.py

Contents of routescript:

```
from bots.botsconfig import *
import bots.transform as transform

def postincommunication(routedict,*args,**kwargs):
    # postincommunication() is run after fromchannel communication.
    # the status of incoming files is changed to outgoing.
    # bots skips parsing and translation.
    transform.addinfo(change={'status':MERGED},where={'status':FILEIN,'idroute
↪':routedict['idroute']})
```

Route Scripting

When the standard routing of bots does not fit your needs, use routescripts. Routescripts are python programs.

There are 2 types of routescripts:

1. User exits: at certain points in a route, bots calls your user exit. See *overview of exit points*. Most common usage is `pre-processing` an edi file, see *recipes*.
2. Your script takes over the whole route. This is done by using function `main()` in your routescript. See *Example*.

How to set up a route-script

1. Use bots-monitor to add a new route.
2. Make a routescript with the same name as the routeID in `bots/usersys/routescripts/<routeid>.py`

Overview Exit Points

These examples show all of the available exit points for route-scripts

```
def start(routedict,*args,**kwargs):
    # run before anything is done is route.
    print routedict['idroute'],'start'

def preincommunication(routedict,*args,**kwargs):
    # if there is a fromchannel: run before fromchannel communication.
    print routedict['idroute'],'preincommunication'
```

```
def postincommunication(routedict,*args,**kwargs):
    # if there is a fromchannel: run after fromchannel communication.
    print routedict['idroute'],'postincommunication'

def pretranslation(routedict,*args,**kwargs):
    # if translation is done in route: run before translation.
    print routedict['idroute'],'pretranslation'

def posttranslation(routedict,*args,**kwargs):
    # if translation is done in route: run after translation.
    print routedict['idroute'],'posttranslation'

def premerge(routedict,*args,**kwargs):
    # if there is a outchannel: run before merging.
    print routedict['idroute'],'premerge'

def postmerge(routedict,*args,**kwargs):
    # if there is a outchannel: run after merging.
    print routedict['idroute'],'postmerge'

def preoutcommunication(routedict,*args,**kwargs):
    # if there is a outchannel: run before outchannel communication.
    print routedict['idroute'],'preoutcommunication'

def postoutcommunication(routedict,*args,**kwargs):
    # if there is a outchannel: run after outchannel communication.
    # eg. call an external program to process files just sent into ERP system
    print routedict['idroute'],'postoutcommunication'

def end(routedict,*args,**kwargs):
    # after everything is done in route
    print routedict['idroute'],'end'
```

Preprocessing and Postprocessing Recipies

Some recipes for preprocessing edi files. Plugin **demo_preprocessing** at the bots sourceforge site demonstrates preprocessing.

Example 1: Discard input files that are too small

```
import os
import bots.preprocess as preprocess
from bots.botsconfig import *

def postincommunication(routedict,*args,**kwargs):
    ''' function is called after the communication in the route.'''
    preprocess.preprocess(routedict=routedict,function=discard_file)

def discard_file(ta_from,endstatus,*args,**kwargs):
    ''' discard files that are to small (zero files)'''
    ta_from.synall()
    filesize = ta_from.filesize
    if filesize < 100:    #filesize in bytes
        ta_from.update(statust=DONE)                    #statust=DONE: bots␣
↪discards file, gives no errors.
```

```
    else:
        ta_to = ta_from.copyta(status=endstatus)            #make new transaction for␣
↪bots database
        ta_to.update(statust=OK,filename=ta_from.filename) #update outmessage␣
↪transaction (same) filename
```

Example 2: Extract data from PDF file (to csv)

```
# Extract data from PDF file (to csv)
# x_group: group text closer than this as one field (default 10)
# y_group: group lines closer than this as one line (default 5)
# password: if required

import bots.preprocess as preprocess

def postincommunication(routedict,*args,**kwargs):
    preprocess.preprocess(routedict,preprocess.extractpdf,x_group=12,y_group=3,
↪password='secret')
```

Example 3: Manipulate records without BOTSID

```
import bots.preprocess as preprocess
import bots.botslib as botslib
import bots.botsglobal as botsglobal
from bots.botsconfig import *

def postincommunication(routedict,*args,**kwargs):
    preprocess.preprocess(routedict,custom_preprocess)

def custom_preprocess(ta_from,endstatus,*args,**kwargs):
    try:
        # copy ta for preprocessing
        ta_to = ta_from.copyta(status=endstatus)

        # open the files
        infile = botslib.opendata(ta_from.filename,'r')
        tofile = botslib.opendata(str(ta_to.idta),'wb')

        # preprocessing: read infile, write tofile
        # This file has headers and lines, but no field that can be used for BOTSID.
        # Determine the line type from the data, and add HDR or LIN in first column
        # Text heading lines and blank lines are omitted
        for line in infile:
            if '\tAU' in line:
                tofile.write('HDR\t' + line)
            elif ('\tWAIT' in line or
                  '\tFULL' in line or
                  '\tEMPTY' in line):
                tofile.write('LIN\t' + line)

        infile.close()
        tofile.close()

        ta_to.update(statust=OK,filename=str(ta_to.idta)) #update outmessage␣
↪transaction with ta_info;
    except:
        txt=botslib.txtexc()
        botsglobal.logger.error(u'Custom preprocess failed. Error:\n%s',txt)
```

```
        raise botslib.InMessageError(u'Custom preprocess failed. Error:\n$error',
↪error=txt)
```

Example 4: Sort input file

```
import bots.preprocess as preprocess
import bots.botslib as botslib
import bots.botsglobal as botsglobal
from bots.botsconfig import *

def postincommunication(routedict,*args,**kwargs):
    preprocess.preprocess(routedict,sort_file)

def sort_file(ta_from,endstatus,*args,**kwargs):
    try:
        # copy ta for preprocessing
        ta_to = ta_from.copyta(status=endstatus)

        # open the files
        infile = botslib.opendata(ta_from.filename,'r')
        tofile = botslib.opendata(str(ta_to.idta),'wb')

        # sort output
        lines = infile.readlines()
        lines.sort()
        for line in lines:
            tofile.write(line)

        infile.close()
        tofile.close()

        ta_to.update(statust=OK,filename=str(ta_to.idta)) #update outmessage␣
↪transaction with ta_info;
    except:
        txt=botslib.txtexc()
        botsglobal.logger.error(u'Sort preprocess failed. Error:\n%s',txt)
        raise botslib.InMessageError(u'Sort preprocess failed. Error:\n$error',
↪error=txt)
```

Example 5: Postprocessing; Post processing works the same way as pre processing, except it is done before out communication.

```
import bots.preprocess as preprocess
import bots.botslib as botslib
import bots.botsglobal as botsglobal
from bots.botsconfig import *

def preoutcommunication(routedict,*args,**kwargs):
    preprocess.postprocess(routedict,split_lines)

def split_lines(ta_from,endstatus,,*args,**kwargs):
    try:
        # copy ta for postprocessing, open the files
        ta_to = ta_from.copyta(status=endstatus)
        infile = botslib.opendata(ta_from.filename,'r')
        tofile = botslib.opendata(str(ta_to.idta),'wb')

        # split every line at the first separator (space)
```

```
        # output the two parts on separate lines
        for line in infile:
            part = line.partition(' ')
            tofile.write(part[0] + '\n' + part[2])

        # close files and update outmessage transaction with ta_info
        infile.close()
        tofile.close()
        ta_to.update(statust=OK,filename=str(ta_to.idta))

    except:
        txt=botslib.txtexc()
        botsglobal.logger.error(_(u'split_lines postprocess failed. Error:\n%s'),txt)
        raise botslib.OutMessageError(_(u'split_lines postprocess failed. Error:\n
↪$error'),error=txt)
```

Example 6: Preprocessing an encrypted file

```
import bots.preprocess as preprocess
import bots.botslib as botslib
import gnupg

# Preprocessing - Decrypt infile using GPG
# Dependencies: python-gnupg-0.3.0
#   botssys/gnugpghome directory, containing:
#     gpg binary files (gpg.exe and iconv.dll)
#     keys (pubring.gpg, secring.gpg, trustdb.gpg)
#     passphrase.txt

def postincommunication(routedict,*args,**kwargs):
    # preprocess to decrypt, then passthrough (no translation)
    preprocess.preprocess(routedict,decrypt_GPG)
    transform.addinfo(change={'status':MERGED},where={'status':FILEIN,'idroute
↪':routedict['idroute']})

def decrypt_GPG(ta_from,endstatus,*args,**kwargs):

    # copy ta for preprocessing
    ta_to = ta_from.copyta(status=endstatus)

    # gnupghome contains the gpg binary files, public/private keys, and passphrase
    gnupghome = botslib.join(botsglobal.ini.get('directories','botssys'),'gnupghome')
    passphrase = open(botslib.join(gnupghome,'passphrase.txt'),'r').read()
    gpgbinary = botslib.join(gnupghome,'gpg.exe')

    # Here is where we do the actual decryption
    gpg = gnupg.GPG(gnupghome=gnupghome,gpgbinary=gpgbinary)
    with botslib.opendata(ta_from.filename,'rb') as input:
        status = gpg.decrypt_file(input, passphrase=passphrase,output=botslib.
↪abspathdata(str(ta_to.idta)))

    # log the results and finish
    botsglobal.logger.debug(status.stderr)
    if status.ok:
        botsglobal.logger.info(status.status)
        ta_to.update(statust=OK,filename=str(ta_to.idta))
    else:
        botsglobal.logger.error(status.status)
```

```
        ta_to.update(statust=ERROR,filename=str(ta_to.idta))
        raise PreprocessError(status.status + '\n' + status.stderr)

class PreprocessError(botslib.BotsError):
    pass
```

Example 7: Preprocessing to ignore/remove XML namespaces; This example changes the default namespace to a namespace prefix (so it is ignored). It also removes a namespace prefix (ENV). You may need to use either or both of these methods, depending on the content of your XML file.

```
#*******************************************************************************
# preprocess - Remove XML namespaces to simplify grammar and mapping
# Generally Bots does not need to use the xmlns for incoming files
# This example handles both default and prefix namespaces

def postincommunication(routedict):

    def _preprocess(ta_from,endstatus,**argv):

        # copy ta for preprocessing
        ta_to = ta_from.copyta(status=endstatus)

        # open the files
        infile = botslib.opendata(ta_from.filename,'r')
        tofile = botslib.opendata(str(ta_to.idta),'wb')

        for line in infile:
            tofile.write(line.replace('xmlns=','xmlns:NOTUSED=').replace('<ENV:','<').
↪replace('</ENV:','</'))

        # close files and update outmessage transaction
        infile.close()
        tofile.close()
        ta_to.update(statust=OK,filename=str(ta_to.idta))

    preprocess.preprocess(routedict,_preprocess)
```

Take Over Whole Route

The below example takes oves the the whole route such that only the `main()` function defined here will be executed as part of the route.

```
# imports needed for some functions below
from bots.botsconfig import *
import bots.transform as transform
import bots.botsglobal as botsglobal
import bots.botslib as botslib
import bots.cleanup as cleanup
import bots.pluglib as pluglib
import os
import time

def main(routedict,*args,**kwargs):
    # if main is present, it takes over the whole route (nothing is done in route␣
↪except this function).
    # for example, create a daily route that does a backup and cleanup
```

```
    # Prevent this script from accidentally running more than once per day
    if transform.persist_lookup(routedict['idroute'],'run_date') != time.strftime('%Y
↪%m%d'):
        transform.persist_add_update(routedict['idroute'],'run_date',time.strftime('%Y
↪%m%d'))

        # backup directory and subdirectory - one per weekday gives a 7 day rolling␣
↪backup
        backup_dir = botslib.join(botsglobal.ini.get('directories','botssys'), 'backup
↪')
        makedir(backup_dir)
        backup_dir = botslib.join(backup_dir, time.strftime('%w-%a'))
        makedir(backup_dir)

        # Create a bots backup (as a plugin)
        botsglobal.logger.info('Create a bots backup')
        pluglib.plugoutcore({
             'databaseconfiguration': True,
             'umlists': True,
             'fileconfiguration': True,
             'infiles': False,
             'charset': False,
             'databasetransactions': False,
             'data': False,
             'logfiles': True,
             'config': True,
             'database': True,
             'filename': botslib.join(backup_dir,'bots-backup.zip')})

        # Do cleanup, if scheduled daily
        # In bots.ini set whencleanup=daily (other values are always or never)
        if botsglobal.ini.get('settings','whencleanup','always') == 'daily':
            botsglobal.logger.info(u'Cleanup of database and files')
            cleanup.cleanup()

def makedir(dir):
    try: os.makedirs(dir)
    except: pass
```

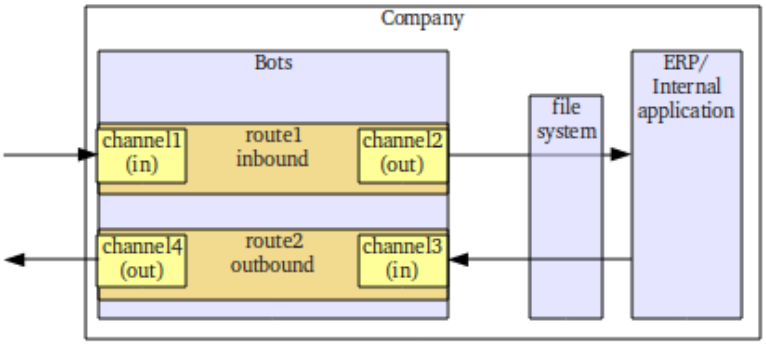
Channels

> Definition: Channels take care of communication with partners and backend systems using file I/O, ftp, email, etc.

A channel is either incoming or outgoing. Examples of channels:

- Receive email from a pop3-mailbox
- Send to a ftp-server
- Pick up in-house invoices from a directory on your computer
- Put orders in a file queue for import in your application.

Example of communication channels for bots:



Notes on naming conventions:

- incoming = incoming to bots
- outgoing = going out bots
- inbound = my organization receives
- outbound = going out of my organization

Note: Bots does client communicates (no server). If you need a communication server (eg for ftp, as2), use a separate server.

Standard Communications

Bots supports a set of communication protocols out of the box such that you only need to configure it in `bots-monitor->Configuration->Channels` to get it working. The set of supported protocols are listed below:

Table 3.1: Supported Communication Types

Protocol	Description
file	Use this to push/pull files with the filesystem, also works with shared file systems.
smtp	Use this to send an email to your trading partner with the EDI file.
smtps	Same as smtp but additionally uses SSL for secure transmission
smtpstarttls	Same as smtp but additionally uses TLS for secure transmission
pop3	Use this to extract emails with EDI files from your email inbox.
pop3s	Same as pop3 but additionally uses SSL for secure transmission
pop3sapop	Same as pop3 but additionally encrypts your password during authentication
http	Use the HTTP protocol to GET or PUT files with your trading partner.
https	Same as http but additionally uses SSL for secure transmission.
imap4	Use this to extract emails with EDI files from your email inbox.
imap4s	Same as imap4 but additionally uses SSL for secure transmission
ftp	Use the FTP protocol to exchange files with your trading partner.
ftps (explicit)	Same as ftp but additionally uses explicit SSL for secure transmission
ftps (Implicit)	Same as ftp but additionally uses implicit SSL for secure transmission
sftp (ssh)	Use the SFTP protocol for exchange files with your trading partner.
xmlrpc	Use the XMLRPC protocol for exchange files with your trading partner.
trash/discard	Use this to discard uneeded messages types from your trading partner.
communicationscript	Use this for custom comms, as described *here*
db	Use this for database comms, as described *here*

Database Communication

Bots can read and write from a database. See plugin `demo_databasecommunication` on the bots sourceforge site.

Discussion: Direct Database Communication or Not

In most edi setups there is no direct database communication, but an intermediate file is used:

- Inbound edi: bots translates edi files to your import format
- Outbound edi: the export files of your application are translated by bots to the wished edi format.

Reasons to set up edi using an intermediate file:

- It is better to do the import/export with the tools you are familiar with, and not to use a new tool (knowledge, maintenance).
- Different edi partners do use different standards (dialects of the standards, different interpretations, different countries, different sectors, edi standards are not that good). Try to use one import/export format; let the edi software handle the differences between partners.

Reasons to use direct database format:

- Import/Exports are very simple/straightforward
- Your system does not (yet) have good functionality for handling the incoming edi data. Example: receive sales reports, but what to do with this? This data is quite simple, just import this in one (new) table. The users can query this table for information.

Inbound edi (write to database)

- Translation rule should be to edi-type `db`

- In the mapping script: out.root should be a python object (dict, list, class, etc); this object is passed to the actual database connector.
- Outgoing channel should be type 'db'.
- Use a communication script for the outgoing channel `usersys/communicationscripts/channelname.py`. This communication script does the actual communication with the database.
- **In the communication script should be 3 functions:**
  - connect - build database connection.
  - out-communicate - put the data in the database using the data as received from the mapping script.
  - disconnect - close database connection.
- Use the database connector as needed, eg: python provides by default the sqlite3 connector, mysql-Python as MySQL database connector, psycopg2 as PostgreSQL database connector, etc

Outbound edi (read from database)

- Incoming channel should be type `db`.
- Use a communication script for the incoming channel `usersys/communicationscripts/channelname.py`. This communication script does the actual communication with the database.
- **In the communication script should be 3 functions:**
  - connect - build database connection.
  - incommunicate - fetch the data from the database, send it to the mapping script.
  - disconnect - close database connection.
- Use the database connector as needed, eg: python provides by default the sqlite3 connector, mysql-Python as MySQL database connector, psycopg2 as PostgreSQL database connector, etc
- Translation should be from edi-type `db`
- In the mapping script: inn.root is the data as received from the database connector.

Note: The data passed from the communication-script can be any python object (eg dict, list). If a list (or tuple) is return from the communication script, bots passes each member of the list as a separate edi-message to the mapping script.

Channel Scripting

When the standard communication of bots does not fit your needs, use communicationscripts. Instruction to make a channel script:

- There must be channel in bots-monitor (or make a new one)
- Make a communicationscript with the **same name** as the channelID
- Place the communicationscript in `bots/usersys/communicationscripts/channelid.py`

Use cases

- Communication method not provided by bots
- Existing communication needs customization
- Call external program to write edi message to your ERP system.

- Additional requirements: Eg. use partner name or order number in output file name.
- Control archive file naming

Types of communication scripts

1. Small user exits: at certain places in normal communication a user script is called. Examples of *small user exits*
2. Subclass: take-over of (parts of) communication script: user script subclasses existing communication type.
3. Communication type `communicationscript`. Bots tries to do the bots-handling of files, you provide the communication details. Examples of communication type 'communicationscript'

Small User Exits

Some examples of small user exists are below:

Example to Filter email attachments: Some edi-partners send signatures etc in their email. Script does a simple check if incoming attachment starts with 'UNB'. (Note: Bots treats any text in the email body as another "attachment")

```
def accept_incoming_attachment(channeldict,ta,charset,content,contenttype,*args,
↪**kwargs):
    if 'UNB' in content[0:50]:
        return True   #attachments is OK
    else:
        return False  #skip this attachment
```

Example to Set email subject: Some edi-partners send signatures etc in their email. By default bots uses a number for emails. Sometimes you want a more meaningfull subject.

```
def subject(channeldict,ta,subjectstring,content,*args,**kwargs):
    ta.synall()         #needed to get access to attributes of object ta (eg ta.
↪frompartner)
    return 'EDI messages from ' + ta.frompartner + '_' + subjectstring
```

Exapmle to Name archive file same as input file: Not needed for bots 3.x where you can **do this via setting in bots.ini**

```
import os
import bots.botslib as botslib

def archivename(channeldict,idta,filename,*args,**kwargs):
     taparent=botslib.OldTransaction(idta=idta)
     ta_list = botslib.trace_origin(ta=taparent,where={'status':EXTERNIN})
     archivename = os.path.basename(ta_list[0].filename)
     return archivename
```

Example to Set the archive path: Path root is set in channel. Add sub-dir per date, then sub-dir per channel under it.

```
import time
import bots.botslib as botslib

def archivepath(channeldict,*args,**kwargs):
    archivepath = botslib.join(channeldict['archivepath'],time.strftime('%Y%m%d'),
↪channeldict['idchannel'])
    return archivepath
```

Example to Partners in the output file name: Not needed for bots 3.x where you can **do this via file name in GUI**.

```
def filename(channeldict,filename,ta,*args,**kwargs):
    ta.synall()         #needed to get access to attributes of object ta (eg ta.
↪frompartner)
    return ta.frompartner + '_' + ta.topartner + '_' + filename
```

Example to Name the output file from botskey: `botskey` can be set in grammar or mapping, eg. from customer's order number. If no botskey is found, the default file naming method will be used. Syntax must contain 'merge':False. Not needed for bots 3.x where you can **do this via file name in GUI**.

```
def filename(channeldict,filename,ta,*args,**kwargs):
    ta.synall()
    if ta.botskey:
        return filename + ta.botskey
    else:
        return filename
```

Example to Name the output file same as input file: Syntax must contain **merge**:False. Not needed for bots 3.x where you can **do this via file name in GUI**.

```
import os
import bots.botslib as botslib

def filename(channeldict,filename,ta,*args,**kwargs):
    ta_list = botslib.trace_origin(ta=ta,where={'status':EXTERNIN})
    filename_in = os.path.basename(ta_list[0].filename) # just filename, remove path
    return filename + filename_in
```

Subclassing

It is possible to overwrite bots communication methods completely. This is done using python subclassing. Again, as with all communication scripting there should be a file in `usersys/communicationscripts` with the same name as the channel (and extension `.py`)

Example 1: In this case communication-type of the channel is 'file'. Bots will check the communication-script file if there is a class called 'file' and use that. The class 'file' subclasses the standard 'file' method of bots.

```
import bots.communication as communication

class file(communication.file):
    def connect(self,*args,**kwargs):
        #do the preparing work
        print 'in connect method'
```

Example 2: In this case communication-type of the channel is 'ftp'. The class 'ftp' subclasses the standard 'ftp' method of bots. The 'outcommunicate' method of the ftp class is taken over with this implementation.

```
import bots.communication as communication
import bots.botslib as botslib
from bots.botsconfig import *

class ftp(communication.ftp):
    @botslib.log_session
    def outcommunicate(self,*args,**kwargs):
        #get right filename_mask & determine if fixed name (append) or files with␣
↪unique names
        filename_mask = self.channeldict['filename'] if self.channeldict['filename']␣
↪else '*'
```

```
        if '{overwrite}' in filename_mask:
            filename_mask = filename_mask.replace('{overwrite}','')
            mode = 'STOR '
        else:
            mode = 'APPE '
        for row in botslib.query('''SELECT idta,filename,numberofresends
                                    FROM ta
                                    WHERE idta>%(rootidta)s
                                      AND status=%(status)s
                                      AND statust=%(statust)s
                                      AND tochannel=%(tochannel)s
                                        ''',
                                    {'tochannel':self.channeldict['idchannel'],
↪'rootidta':self.rootidta,
                                    'status':FILEOUT,'statust':OK}):
            try:
                ta_from = botslib.OldTransaction(row['idta'])
                ta_to = ta_from.copyta(status=EXTERNOUT)
                tofilename = self.filename_formatter(filename_mask,ta_from)
                if self.channeldict['ftpbinary']:
                    fromfile = botslib.opendata(row['filename'], 'rb')
                    self.session.storbinary(mode + tofilename, fromfile)
                else:
                    fromfile = botslib.opendata(row['filename'], 'r')
                    self.session.storlines(mode + tofilename, fromfile)
                fromfile.close()
            except:
                txt = botslib.txtexc()
                ta_to.update(statust=ERROR,errortext=txt,filename='ftp:/'+posixpath.
↪join(self.dirpath,tofilename),numberofresends=row['numberofresends']+1)
            else:
                ta_to.update(statust=DONE,filename='ftp:/'+posixpath.join(self.
↪dirpath,tofilename),numberofresends=row['numberofresends']+1)
            finally:
                ta_from.update(statust=DONE)
```

Example 3: In this case communication-type of the channel is 'ftp' or 'sftp'. The class 'ftp' subclasses the standard 'ftp' method of bots. The 'disconnect' method of the ftp class is taken over with this implementation. The bots channel should be configured to upload either to a 'tmp' sub-directory, or with a '.tmp' extension. This function renames the files once uploads are complete, this preventing the recipient from processing partial files.

```
'''
For safety when uploading to ftp servers, it is a good idea to rename/move
files once complete. This prevents the receiver processing partial files.
When all files have been sent and before the session is disconnected, the
files are renamed so the receiver can process them.

Two methods are available:
 1. Append extension ".tmp" to the channel filename
    This method is simpler, but the receiver may still process the
    .tmp files if it does not look for specific extensions to process.
 2. Append subdirectory "/tmp" to the channel path
    This requires an extra directory created on the server, you may not
    be authorised to do this.

Subclassing of ftp.disconnect. Import this to your communicationscript (ftp or sftp␣
↪as required):
    from _ftp_rename import ftp
```

```
    from _ftp_rename import sftp

Mike Griffin  4/09/2013

'''

import bots.communication as communication
import bots.botslib as botslib
import bots.botsglobal as botsglobal

class ftp(communication.ftp):
    def disconnect(self,*args,**kwargs):

        # rename files to remove .tmp extensions
        if self.channeldict['filename'].endswith('.tmp'):
            for f in self.session.nlst():
                if f.endswith('.tmp'):
                    try:
                        self.session.rename(f,f[:-4])
                    except:
                        pass

        # rename files from tmp subdirectory to parent directory
        if self.channeldict['path'].endswith('/tmp'):
            for f in self.session.nlst():
                try:
                    self.session.rename(f,'../%s' %f)
                except:
                    pass

        try:
            self.session.quit()
        except:
            self.session.close()
        botslib.settimeout(botsglobal.ini.getint('settings','globaltimeout',10))

class sftp(communication.sftp):
    def disconnect(self,*args,**kwargs):

        # rename files to remove .tmp extensions
        if self.channeldict['filename'].endswith('.tmp'):
            for f in self.session.listdir('.'):
                if f.endswith('.tmp'):
                    try:
                        self.session.rename(f,f[:-4])
                    except:
                        pass

        # rename files from tmp subdirectory to parent directory
        if self.channeldict['path'].endswith('/tmp'):
            for f in self.session.listdir('.'):
                try:
                    self.session.rename(f,'../%s' %f)
                except:
                    pass

        self.session.close()
        self.transport.close()
```

Example 4: In this case communication-type of the channel is 'ftp'. The class 'ftp' subclasses the standard 'ftp' method of bots. The 'disconnect' method of the ftp class is taken over with this implementation. This provides a way to submit a remote command to the ftp server, for example to run a program on that server. The bots channel is configured with the command in the 'parameters' field.

```
'''
Before disconnecting, send a remote command
Channel "parameters" holds the command to send

Subclassing of ftp.disconnect. Import this to your communicationscript:
    from _ftp_remote_command import ftp

Mike Griffin  13/09/2013
'''

import bots.communication as communication
import bots.botsglobal as botsglobal

class ftp(communication.ftp):
    def disconnect(self,*args,**kwargs):

        # send remote command to ftp server
        botsglobal.logger.info('Send remote command: %s',self.channeldict['parameters
↪'])
        self.session.sendcmd('RCMD %s' %self.channeldict['parameters'])

        try:
            self.session.quit()
        except:
            self.session.close()
        botslib.settimeout(botsglobal.ini.getint('settings','globaltimeout',10))
```

Communication type `communicationscript`

In this case, the channel must be configured with Type: `communicationscript`. In the communicationscript some functions will be called:

- connect (required)
- main (optional, 'main' should handle files one by one)
- disconnect (optional)

Different ways of working:

1. **For incoming files (bots receives the files):**
    - Connect puts all files in a directory, there is no 'main' function. bots can remove the files (if you use the `remove` switch of the channel). See example 1.
    - Connect only builds the connection, `main` is a generator that passes the messages one by one (using `yield`). bots can remove the files (if you use the `remove` switch of the channel). See example 2.
2. **For outgoing files (bots sends the files):**
    - No `main` function: the processing of all the files can be done in `disconnect`. bots can remove the files (if you use the `remove` switch of the channel). See example 3.
    - If there is a `main` function: the `main` function is called by bots after writing each file. bots can remove the files (if you use the `remove` switch of the channel). See example 4.

Example 1: incoming files via external program all at once

Calls an external program. Think eg of a specific communication module for a VAN. All files are received at once to a folder, then processed like a normal file channel.

```
import subprocess

def connect(channeldict,*args,**kwargs):
    subprocess.call(['C:/Program files/my VAN/comms-module.exe','-receive'])
```

Example 2: incoming files via external program one by one

TODO: make a valid example using yield. main is a generator.

```
import subprocess

def connect(channeldict,*args,**kwargs):
    ''' function does nothing but it is required.'''
    pass

def main(channeldict,*args,**kwargs):

    yield ?
```

Example 3: outgoing files via external program all at once

Calls an external program. Think eg of a specific communication module for a VAN. In this example the 'disconnect' script is called after all files are written to directory; in disconnect all files are passed to external communication-module.

```
import subprocess
import os

def connect(channeldict,*args,**kwargs):
    ''' function does nothing but it is required.'''
    pass

def disconnect(channeldict,*args,**kwargs):
    subprocess.call(['C:/Program files/my VAN/comms-module.exe','-send',os.path.
↪join(channeldict['path'],'\*.xml'])
```

Example 4: outgoing files via external program one by one

Calls an external program. Think eg of a specific communication module for a VAN. In this example the 'main' script is called for each outgoing file.

```
import subprocess

def connect(channeldict,*args,**kwargs):
    ''' function does nothing but it is required.'''
    pass

def main(channeldict,filename,ta,*args,**kwargs):
    subprocess.call(['C:/Program files/my VAN/comms-module.exe','-send',filename])
```

Example 5: outgoing files to a printer

Send data (eg. ZPL code to print fancy labels) directly to a Windows configured printer. The printer can be defined in Windows either as "Generic/Text Only" or with the proper driver, because this script just sends raw data, bypassing the driver.

Dependencies: Requires pywin32
Reference: http://timgolden.me.uk/pywin32-docs/win32print.html

```
import os
import win32print
import bots.transform as transform

def connect(channeldict,*args,**kwargs):
    ''' function does nothing but it is required.'''
    pass

def main(channeldict,filename,ta,*args,**kwargs):

    # set printer values required
    ta.synall()
    printer = transform.partnerlookup(ta.topartner,'attr1')
    jobname = ta.botskey

    # read the output file
    with open(filename,'r') as content_file:
        content = content_file.read()

    # send data to the printer
    hPrinter = win32print.OpenPrinter(printer)
    hJob = win32print.StartDocPrinter(hPrinter,1,(jobname,None,'RAW'))
    win32print.WritePrinter(hPrinter,content)
    win32print.EndDocPrinter(hPrinter)
    win32print.ClosePrinter(hPrinter)
```

Filenames

In a perfect world, filenames are unimportant in the EDI flow. All that is required is that they are unique so no file is ever overwritten. Bots takes care of this automatically by default when creating files, by using counters to generate unique numeric filenames.

But sometimes in the **real world** implementation of system interfaces, you want or need to have specific filenames:

- Because your trading partner requires it
- Limitations of other systems regarding filenames
- To provide easy identification of files
- Nicer for end users (eg. when emailing attachments)

Input Filenames

Bots uses filename pattern matching to select input files. This allows you to select all files, or only specific files, from the channel path. Some points to consider:

- Many types of channel (eg. ftp) are case sensitive. `*.TXT` is not the same as `*.txt`
- Files with and without extensions may be treated differently; `*` is not the same as `*.*`
- For **safety** you should use a partially specified name if possible. This prevents accidentally picking up files that should not be there. eg. use `ORDER*.TXT` rather than `*.*`
- For an in-channel with type=file a wild-card can be used in the path.If directory structure is like this:

- botssys/infile/partner1
- botssys/infile/partner2
- botssys/infile/partner3
- Use path botssys/infile/* to read the files in all these directories.

Output Filenames

- A unique name can be generated with an asterisk; the asterisk is replaced by an unique number. Eg: `order_*.edi` -> order1.edi, order2.edi, order3.edi etc
- (bots >= 3.0) Any ta value can be used; eg. {botskey}, {alt}, {editype}, {messagetype}, {frompartner}, {topartner}, {fromchannel}, {tochannel}, {idroute}.
- (bots > = 3.0) Date/time using {datetime} with any valid date or time format specification; eg. {datetime:%Y%m%d}, {datetime:%H%M%S} etc.
- (bots > = 3.0) Incoming filename can be used (name and extension, or either part separately); eg. {infile}, {infile:name}, {infile:ext}

Some examples are shown in the table below.

Channel filename	Description	Example filename generated
* or blank	create a unique name, no extension	39724
*.txt	create a unique name with .txt extension	39724.txt
{botskey}.txt	use incoming botskey value (eg. order number) with .txt extension. Note: {botskey} can only be used if merge is False for the messagetype	BA7358-0.txt
{infile}	passthrough incoming filename & extension to output	Order001.edi
{infile:name}.txt	passthrough incoming filename but change extension to .txt	Order001.txt
{editype}_{messagetype}_{datetime:%Y%m%d}_*{infile:ext}	use editype, messagetype, date and unique number with extension from the incoming file	edifact_ORDERSD93AUN_20120926_39
{frompartner}/{editype}/{infile}	You can also use subdirectories in the filename, but they must already exist. These will be appended to the path.	KMART/edifact/Order001.edi
{frompartner}/INPUT/ORDER_{botskey}.csv	Fixed values can also be included as part of the directory structure or filename	KMART/INPUT/ORDER_BA7358-0.csv
{overwrite}daily_report.txt	Force overwriting of a file if it exists. **Use this with caution; make sure it is really what you want.** May be required on some sftp servers that do not support append mode.	daily_report.txt
{infile[4]}{infile[5]}{infile[6]}{infile[7]}.xml	This functionality uses the Python Format String Syntax which does not have support for **slicing**, but you can use this workaround to pick a range of single characters. **Beware: this does not check for wrong string positions.**	infile: INV_7389.txt generates: 7389.xml

Warning: Do not change out.ta_info['filename'] in your scripts. Although it may appear to work, it messes up Bots internal file storage.

User scripting for output filenames

Bots has the capability to set output filenames with a *communicationscript*; however this requires a new script for each channel and is somewhat complex. Prior to version 3.0 this was the only method available. It can still be used for difficult requirements (but let us know about your needs through the mailing list, we may be able to integrate it).

Channel Port Numbers

Some types of channels require a host and port number. These details should be provided to you by the server administrator. If no port number is provided to you, most likely the **default** port is being used. The most common default port numbers are listed below.

Channel Type	Port number
smtp	25
smtps	587
smtpstarttls	587
pop3	110
pop3s	995
pop3apop	110
imap4	143
imap4s	993
ftp	21
ftps	21
sftp	22

Safe File Writing/Locking

The Problem

- ERP writes an in-house file; bots starts to read this before the file is completely written.
- Bots writes file over FTP, but file is read before Bots finishes writing.

The Solution

1. Tmp-part file name: bots writes a filename, than renames the file.

   Example 1. In channel: filename is `myfilename_*.edi.tmp`, tmp-part is `.tmp`
   Bots writes: `myfilename_12345.edi.tmp`
   Bots renames: `myfilename_12345.edi`

2. System lock: use system file locks for reading or writing edi files (windows, *nix).
3. Lock-file: Directory locking: if lock-file exists in directory, directory is locked for reading/writing. Both reader and writer should check for this.

Translation

Definition: A translation translates a message of a certain editype, messagetype to another editype, messagetype

Needed for a translation:

- Translation rule in `bots-monitor->Configuration->Translations`; see the screen shot below.
- *Grammar* for incoming message.
- *Grammar* for outgoing message.
- *Mapping script* that for converting incoming message to outgoing message.

Read the 1st translation rule of this screen shot:

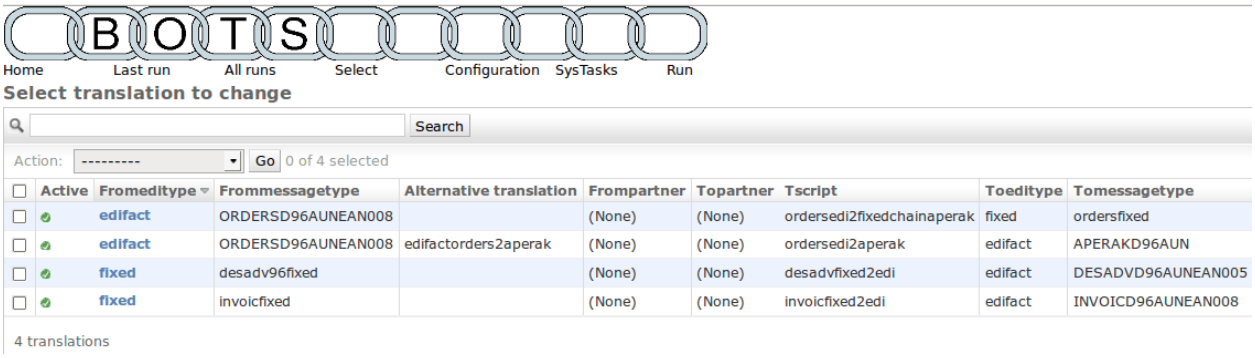
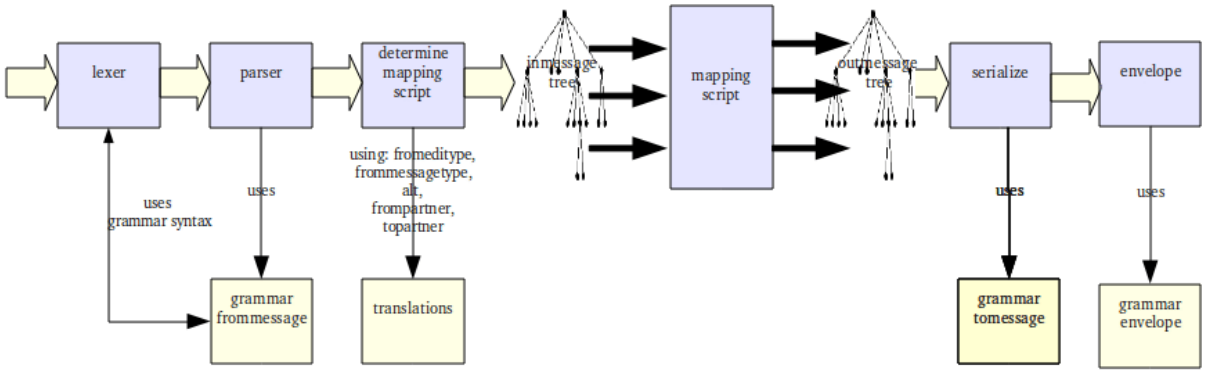


Fig. 3.1: Screenshot of configured translations-rules

Translate `edifact-ORDERSD96AUNEAN008` to `fixed-myinhouseorder` using mapping script `ordersedifact2myinhouse.py`

How Translation Works

Best understood by looking at this schematic:



Step-by-step:

1. The edi file is lexed and parsed using the *grammar*.
2. **The message is transformed into a tree structure. This is similar to the use of DOM in xml. Advantages:**

   - Easy access to message content. This is quite similar to XML-queries or X-path.
   - Choose the logic for the mapping script that is best fit for the situation - instead of being forced to 'loop' over incoming message.
   - Sorting: eg. sort article lines by article number.
   - Counting: eg. count number of lines, total amounts etc
   - Access the data you already written in the tree

3. Split the edi file into separate messages (eg one edi file can contain multiple orders).
4. Find the *right translation* for message.
5. Run the *mapping script* for message.

6. Serialize the outmessage-tree to file. This is checked and formatted according to the *grammar* of the outgoing message.
7. Outgoing messages are enveloped and/or merged.

What Translation When?

Bots figures out what translation to use via the translation rules. Best is to think of the translation rules as a `lookup` table:

> look-up with (from-editype, from-messagetype, alt, frompartner and topartner) to find mappingscript, to-editype and to-messagetype

How the input values for the look-up are determined:

1. **editype**
   - configured in the route
   - if you configure editype=mailbag, bots will figure out if editype is x12, edifact or tradacoms.
2. **messagetype can be:**
   - configured in the route, eg for editype csv
   - for edifact, x12 and tradacoms: bots figures out the detailed messagetype. Example:
   - in route: editype: edifact, messagetype: edifact
   - in incoming edi file bots finds detail messagetype **ORDERSD96AUN**.
3. **frompartner can be:**
   - configured in the route
   - determined by the grammar using QUERIES
4. **topartner can be:**
   - configured in the route
   - determined by the grammar using QUERIES
5. **alt can be:**
   - configured in the route
   - determined by the grammar using QUERIES
   - set by mapping script in a *chained translation*

Note:

- **For frompartner and topartner: bots finds the most specific translation.**
  - **Eg example with 2 translation rules:**
    - fromeditype = edifact, frommessagetype = ORDERSD96AUNEAN008
    - fromeditype = edifact, frommessagetype = ORDERSD96AUNEAN008, frompartner=RETAILERX
  - If bots receives an ORDERS message from RETAILERX, the 2nd translation is used.
  - For other partners the first translation is used.
- for alt-translations: only find the translation with that specific **alt**.

Chained Translations

Definition: chained translations translate one incoming format to multiple outgoing formats.

Example: Translate edi-order to in-house format AND send an email to inform sales representative. To use this bots uses the `alt`. How this works:

- receive incoming file
- do a translation (using mapping script)
- mapping script returns alt-value
- do another translation using the alt-value.

Background information: *TranslationRules how bots determines what to translate*

By example

- **Set up first translation rule (to csv-format) as usual:**
  - Translate x12-850 to csv-orders using mapping script 850_2_csv
- At end of mapping script 850_2_csv: `return 'do_email_translation' #alt-value is returned`
- Set up the second translation rule: * translate x12-850 to csv-orders using mapping script 850_2_email **where alt=do_email_translation**

By returning the alt-value `do_email_translation` mapping script 850_2_csv triggers the 2nd translation (with mapping script 850_2_email).

Plugin

In plugin edifact_ordersdesadvinvoic on bots sourceforge site:

1. incoming is edifact orders.
2. translate fixed inhouse format.
3. translate to edifact APERAK/acknowledgement (if acknowledgement is asked).

Details

- Of course it is possible to 'chain' more than one translation.
- **I have used this to do complex 'sorts' on incoming documents, eg:**
  - write/sort to multiple outgoing messages sorting for destination of goods

Note: In this type of set-up multiple formats are outgoing, you'll want to use a composite route.

Special Chained Translations

You can also return an alt value that is a python dict. The dict must contain 'type' and 'alt' strings; there are several special types available for different processing requirements.

out_as_inn

Do chained translation: use the out-object as inn-object, new out-object. Use cases:

1. Detected error in incoming file; use out-object to generate warning email.
2. Map inputs to standard format, also map standard format to human readable version (eg. html template)

Both out-objects are output by Bots and can be sent to same or different channels using channel filtering.

```
# if the first output is not needed to send somewhere, discard it
    # omit this step and you will get both outputs
    from bots.botsconfig import *
    out.ta_info['statust'] = DONE

    # use output of first mapping as input of second mapping
    return {'type':'out_as_inn','alt':'do_email_translation'}
```

no_check_on_infinite_loop

Do chained translation: allow many loops with same alt-value. Normally, Bots will detect and prevent this, by stopping after 10 iterations of the loop. You are disabling this safety feature, so your mapping script will have to handle this correctly to ensure the looping is not infinite.

```
# we MUST have a way to exit the loop.
if (some condition):
    return

# loop through this mapping multiple times...
return {'type':'no_check_on_infinite_loop','alt':'repeat'}
```

Multiple EDI Versions

There are several situations where you need to send different versions of messages and receive different versions of an EDI standard. In real world versions are often so similar that the same mapping script can be used (or a simple if-then can cater for the differences). We can handle such requirements using any of the below solutions:

Send multiple versions using partners

Use the `topartner` to determine the right version to send.

- **One grammar for in-house message:**
  - myinhouseorder.py
  - This grammar uses QUERIES to extract 'topartner'.
- **Grammars for both the EDI versions:**
  - ORDERSD93AUN
  - ORDERSD96AUN
- **2 translation rules:**
  - fixed-myinhouseorder to edifact-ORDERSD93AUN using mapping script orders-fixed2edifact93.py for topartner=XXX
  - fixed-myinhouseorder to edifact-ORDERSD96AUN using mapping script orders-fixed2edifact93.py for topartern=YYY

Send multiple versions using ''alt''

Information about the version is in in-house-message: a field that contains either '93' or '96'.

- **one grammar for in-house message:**

  - myinhouseorder.py
- **Grammars for both the EDI versions:**
  - ORDERSD93AUN
  - ORDERSD96AUN
- **2 translation rules**
  - fixed-myinhouseorder to edifact-ORDERSD93AUN using mapping script orders_fixed2edifact93.py for alt=93
  - fixed-myinhouseorder to edifact-ORDERSD96AUN using mapping script orders_fixed2edifact93.py for alt=96

Receive Multiple Versions

- **Grammars for both the EDI versions:**
  - ORDERSD93AUN
  - ORDERSD96AUN
- **one grammar for in-house message:**
  - myinhouseorder.py
  - This grammar uses QUERIES to extract 'alt'-value.
- **2 translation rules**
  - edifact-ORDERSD93AUN to fixed-myinhouseorder using mapping script orders_edifact93_2_fixed.py
  - edifact-ORDERSD6AUN to fixed-myinhouseorder using mapping script orders_edifact96_2_fixed.py

Mapping Scripts

Definition: Instructions to get data from incoming edi-message and put it in the outgoing edi-message

- Mapping scripts are python programs.
- Mapping script are in: `usersys/mappings/editype/mapping-script-name.py`.
- Within a mapping script function `main()` is called.
- **Some important things to be noted about mapping scripts:**
  - All data in the incoming/outgoing messages are strings.
  - Errors in a mapping script are caught by Bots and displayed in bots-monitor.
  - You can Raise an exception in your mapping script if you encounter an error situation.
  - (Bots>=3.0) if an error is raised in a script, other translations for message in same edi-file will continue.
  - (Bots>=3.0) when an error situation is met in script: you can send specific error-email to responsible people.

How Mapping Works

Explained in the below example

```
#this mapping script maps an incoming edifact message to a fixed record file.
#bots calls the 'main' function in this mapping script

def main(inn,out):
    #inn: the object for the incoming message; via get() and getloop() the content of␣
↪the message can be accessed.
    #out: the object for the outgoing message; via put() and putloop() content is␣
↪written for this message.

    ordernumber = inn.get({'BOTSID':'UNH'},{'BOTSID':'BGM','1004':None})  #get the␣
↪order-number.
                                                                          #The order-
↪number is in field '1004' of record BGM.
                                                                          #Record BGM␣
↪is nested under record UNH.
    out.put({'BOTSID':'HEA','ORDERNUMBER':ordernumber})                   #put the␣
↪order-number in the outgoing fixed message, field 'ORDERNUMBER' in record HEA.


    #We first did a get(), than a put(). This can be done in one line:
    out.put({'BOTSID':'HEA','ORDERTYPE':inn.get({'BOTSID':'UNH'},{'BOTSID':'BGM',
↪'C002.1001':None})})


    #There can be several dates, all in a DTM record.
    #The 'qualifier' determines the type of date.
    #Fetch the dates like this:
    out.put({'BOTSID':'HEA','ORDERDATE':inn.get({'BOTSID':'UNH'},{'BOTSID':'DTM',
↪'C507.2005':'137','C507.2380':None})})     #get statement ONLY looks for DTM with␣
↪qualifier 137
    out.put({'BOTSID':'HEA','DELIVERY_DATE':inn.get({'BOTSID':'UNH'},{'BOTSID':'DTM',
↪'C507.2005':'2','C507.2380':None})})
    #please note: no looping over the DTM-records, just get the data.

    #The orderlines are in a LIN-recordgroup; so we want each line in its own fixed␣
↪LIN-record.
    #start looping the lines (LIN-segments) in the incoming message:
    for lin in inn.getloop({'BOTSID':'UNH'},{'BOTSID':'LIN'}):
        #write a fixed LIN record:
        lou = out.putloop({'BOTSID':'HEA'},{'BOTSID':'LIN'})
        #Note: in this loop get() is used on the lin-object, and put() is used for␣
↪the lou-object.
        lou.put({'BOTSID':'LIN','LINENUMBER':lin.get({'BOTSID':'LIN','1082':None})})
        lou.put({'BOTSID':'LIN','ARTICLE_GTIN':lin.get({'BOTSID':'LIN','C212.7140
↪':None})})
        lou.put({'BOTSID':'LIN','QUANTITY':lin.get({'BOTSID':'LIN'},{'BOTSID':'QTY',
↪'C186.6063':'21','C186.6060':None})})

    #OK, we did the lines, but forgot to map the party's - the parties are before the␣
↪lines  in the incoming message.
    #No problem, get/put can be done anywhere in the mapping - we are NOT looping the␣
↪incoming or outgoing message:

    out.put({'BOTSID':'HEA','BUYER_ID':inn.get({'BOTSID':'UNH'},{'BOTSID':'NAD','3035
↪':'BY','C082.3039':None})})
```

```
    out.put({'BOTSID':'HEA','SUPPLIER_ID':inn.get({'BOTSID':'UNH'},{'BOTSID':'NAD',
↪'3035':'SU','C082.3039':None}))
    out.put({'BOTSID':'HEA','DELIVERYPLACE_ID':inn.get({'BOTSID':'UNH'},{'BOTSID':'NAD
↪','3035':'DP','C082.3039':None}))
```

MPATH: Query the edi data

In the examples above the data in the edi messages is queried using eg `{'BOTSID':'UNH'}, {'BOTSID':'NAD','3035':'BY','C082.3039':None}`. This **thing** is called a mpath (just a name). Mpath is quite important in building a mapping.

More about mpath:

- An mpath consists of one or more python 'dicts' (dictionaries), separated by commas: *dict1,dict2, dict3*
- `BOTSID` is the record identifier.
- In get(): `None` indicates that the value of this field should be returned. All others fields are interpreted as: if field==value.
- In put(): all name-values are written, but not if one of the values is `None`. If one of the values is None no values at all are written to outgoing message. This is used in the combined out.put(...inn.get(...)...)-statements.

Detailed information about get, getloop, put, putloop is *here*

Warning: Never do this (use 2 inn.get's in one out.put):

```
out.put({'BOTSID':'ADD','7747':inn.get('BOTSID':'HEA','name1':None),'7749':inn.get(
↪'BOTSID':'HEA','name2':None)})}
```

Reason: if either name1 or name2 is not there (empty, None) nothing will be written in this statement.

Another Mapping Example

```
def main(inn,out):
    #The incoming message object (inn) has a dict 'ta_info'.
    #ta_info contains information from the QUERIES in the grammar;
    #mostly  envelope data like sender, reciever etc.about the message as specified␣
↪in the grammar (queries, SUBTRANSLATION).
    out.put({'BOTSID':'HEA','SENDER':inn.ta_info['frompartner']})

    #The outgoing message object (out) also has a dict ta_info.
    #You can change it if required. eg. to set topartner:
    BuyerID = inn.get({'BOTSID':'HEA','BUYER_ID':None})
    inn.ta_info['topartner'] = BuyerID       #use BuyerID as topartner (when␣
↪eveloping)

    #Inn-get() either return a value, or 'None'. Look at the next line:
    out.put({'BOTSID':'UNH'},{'BOTSID':'DTM','C507.2005':'137','C507.2380':inn.get({
↪'BOTSID':'HEA','ORDERDATE':None}))})
    #if there is no ORDERDATE in the HEA record, put() receives a 'None'-value.
    #Nothing will be written to the outgoing message.

    #in the next lines 2 values from the inhouse record are written to the same␣
↪record:
    out.put({'BOTSID':'UNH'},{'BOTSID':'NAD','3035':'DP','C082.3055':'9','C082.3039
↪':inn.get({'BOTSID':'HEA','DELIVERYPLACE_ID':None})})
    out.put({'BOTSID':'UNH'},{'BOTSID':'NAD','3035':'DP','C058.3036':inn.get({'BOTSID
↪':'HEA','DELIVERYPLACE_NAME':None})})
```

```
    #both the deliveryplace_id and deliveryplace_name are written to a NAD-record␣
↪with qualifier 'BY .
```

Get Data from Incoming Message

Use any of these functions to get data from an incoming message:

get(mpath)

Get 1 field from the incoming message; mpath specifies which field to get. Returns: string or, if field not found, `None`.

```
#get the message date from an edifact invoice:
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.

getnozero(mpath)

Like get(), but: return a numeric string not equal to '0', otherwise None. Eg useful in fixed records, where a numeric field is often initialized with zero's.

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 below:

```
#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})
```

Put Data in Outgoing Message

Use any of these functions to put data into an outgoing message:

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.

```
#put a message date in a edifact message
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.

putloop(mpath)

- Used to generate repeated records or record groups.
- **Recommended**: only use it as in: line = putloop(mpath)
- 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.

```
#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})})
```

Warning: Never use 2 inn.get's in one out.put (unless you really know what you are doing ;-)

```
out.put({'BOTSID':'ADD','7747':inn.get('BOTSID':'HEA','name1':None),'7749':inn.get(
↪'BOTSID':'HEA','name2':None)})
```

Because: if either name1 or name2 is not there (empty, None) nothing will be written in this statement.

Mapping Functions

Bots provides a set of helper functions out of the box to ease the map development. Let us see the available functions:

sort(mpath)

Sorts the incoming message according to certain value. Sorts alphabetically.

```
inn.sort({'BOTSID':'UNH'},{'BOTSID':'LIN','C212.7140':None})
#sort article lines by EAN article number (GTIN).
```

transform.useoneof(first get, second get, etc)

Use for default values or when data can be in different places in a message.

```
value = transform.useoneof(inn.get({'BOTSID':'IMD','C960.4294':None}),inn.get({'BOTSID
↪':'IMD','7078':None}))
#returns the result of the first get() that is succesful.
#remarked was that this is simular to:
```

```
value = inn.get({'BOTSID':'IMD','C960.4294':None}) or inn.get({'BOTSID':'IMD','7078
↪':None})
#Usage for default values (if field not there, use default):
value = inn.get({'BOTSID':'IMD','C960.4294':None}) or 'my value'
```

transform.datemask(value,frommask,tomask)

- Does format conversions based upon pattern matching.
- Especially useful for date/time conversion.
- **Note**: only simple pattern matching, without 'intelligence' about date/time.

```
transform.datemask('09/21/2011','MM/DD/CCYY','YY-MM-DD')
#returns '11-09-21'
```

- Funny trick with datamask:

```
print transform.datemask('201512312359','YYYYmmDD0000','YYYYmmDD0000')
print transform.datemask('201512310000','YYYYmmDD0000','YYYYmmDD0000')
print transform.datemask('20151231','YYYYmmDD0000','YYYYmmDD0000')
#returns date always in CCYYmmDDHHMM, if no tiem in original tiem is '0000'
```

transform.concat(*args)

Concatenate a list of strings. If argument is None, nothing is concatenated.

```
transform.concat('my',None,'string')
#returns 'mystring'
```

transform.sendbotsemail(partner,subject,reporttext)

- Send a simple email message to any bots partner (in partner-table) from a mapping script.
- Mail is sent to all To: and cc: addresses for the partner (but send_mail does not support cc).
- Email parameters are in config/settings.py (EMAIL_HOST, etc).

```
transform.sendbotsemail('buyerID1','error in messsge','There is an error in message␣
↪blahblah')
```

transform.unique(domain)

- Returns counter/unique number.
- For each **domain** separate counters are used.
- Counter start at **1** (at first time you use counter).
- The counter can be changed in `bots-monitor->SysTasks->view/edit counters`

```
transform.unique('my article line counter')
#returns a number unique for the domain 'my article line counter'
```

transform.unique_runcounter(domain))

- Returns counter/unique number during the bots-run.
- For each **domain** separate counters are used.
- Counter start at **1** (at first time you use counter).
- In a next run the counter will start again at **1**.
- Useful for eg a message-counter per interchange.

transform.inn2out(inn,out)

Use the incoming message as the outgoing message. Is useful to translate the message one-on-one 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/messagetype including changes).

Code Conversion

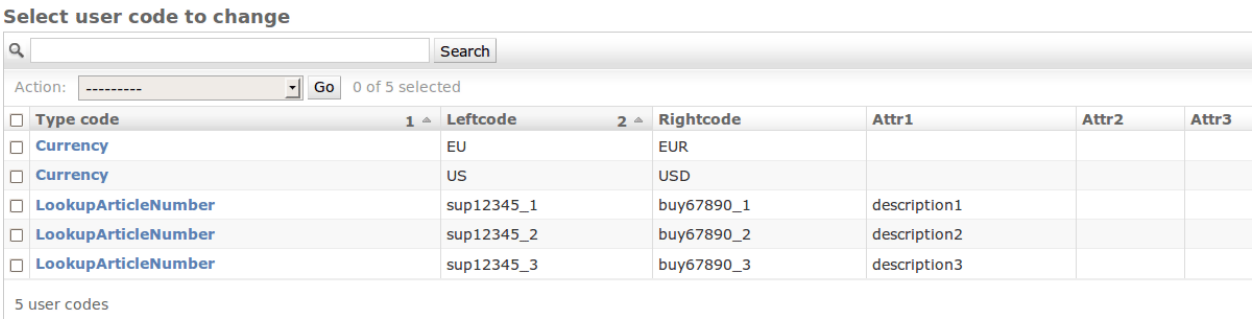
Bots supports code conversions. The code conversion is done in a mapping script; maintenance for the codes can be done via `bots-monitor->Configuration->User codes as list`. This page contains 3 examples of code conversions:

1. Convert currency code list.
2. Convert internal article code to buyers article code.
3. Convert internal article code to description.

Code Maintenance in GUI

First configure 2 code lists (`bots-monitor->Configuration->user codes by type`):

Make the code conversions (`bots-monitor->Configuration->user codes as list`):



Code Conversion in Mapping Script

```
import bots.transform as transform

#convert currency code
our_currency_code = inn.get({'BOTSID':'HEA','VALUTA':None})
converted_currency_code = transform.ccode('Currency',our_currency_code)

#convert internal article code to buyers article code:
buyer_article_number = transform.ccode('LookupArticleNumber',our_article_number)

#get description (in field 'attr1') for article
description = transform.ccode('LookupArticleNumber',our_article_number,field='attr1')

#code conversion also works via reverse lookup:
our_article_number = transform.reverse_ccode('LookupArticleNumber',buyer_article_
↪number)
```

Code Conversion Functions

transform.ccode(codelist, value, field, safe)

Convert **value** to value in **field** using a user-maintained code list. Parameters:

- *codelist*: codelist as in `bots-monitor->Configuration->user` codes by type.
- *value* to be converted (should be in **leftcode**)
- *field*: the field to lookup (if not specified: **rightcode**)
- *safe*: if False (default): raise exception when value is in found in codelist. If True: just return **value**.

Example of usage for leftcode to rightcode:

```
transform.ccode('articles','8712345678906')
```

Example of usage for leftcode to attr1:

```
transform.ccode('articles','8712345678906','attr1')
```

transform.reverse_ccode(codelist, value, field)

Same as transform.ccode(), but conversion is from **rightcode** to **field**.

Changes in Code Conversion Functions

These functions have changed over versions. The old functions are deprecated but still work.

bots<2.1	bots<3.0	bots>=3.0
codetconversion	ccode	ccode
safecodetconversion	safe_ccode	ccode with parameter safe=True
rcodetconversion	reverse_ccode	reverse_ccode
safercodetconversion	safe_reverse_ccode	reverse_ccode with parameter safe=True

Calculations and Counting

- Sometimes it is needed to do calculations in mappings.
- Realize that the get() functions always return strings, so these strings need to be converted.
- Always convert to python's decimals, this is the **only** way to avoid rounding errors.

- More details: pythons decimal documentation

Example calculation

```
import decimal

def main(inn,out):
    TOTAL_AMOUNT = decimal.Decimal('0')          #initialize total amount
    for lin inn.getloop({'BOTSID':'UNH'},{'BOTSID':'LIN'}):
        amount = lin.get({'BOTSID':'LIN'},{'BOTSID':'QTY','1234':None})
        TOTAL_AMOUNT += decimal.Decimal(amount)    #add amount to TOTAL_AMOUNT

    #convert TOTAL_AMOUNT back to string, indicating the number of decimals to be␣
↪used (precision)
    total_amount = TOTAL_AMOUNT.quantize(decimal.Decimal('1.00'))    #2 decimals␣
↪precision
```

Bots version 3.1 has a new method getdecimal(); code above can now be:

```
import decimal

def main(inn,out):
    TOTAL_AMOUNT = decimal.Decimal('0')          #initialize total amount
    for lin inn.getloop({'BOTSID':'UNH'},{'BOTSID':'LIN'}):
        TOTAL_AMOUNT += lin.getdecimal({'BOTSID':'LIN'},{'BOTSID':'QTY','1234':None})␣
↪   #add amount to TOTAL_AMOUNT

    #convert TOTAL_AMOUNT back to string, indicating the number of decimals to be␣
↪used (precision)
    total_amount = TOTAL_AMOUNT.quantize(decimal.Decimal('1.00'))    #2 decimals␣
↪precision
```

Note: The example above could be done using function getcountsum(), see below. The recipe above gives you detailed control over the calculation.

getcount()

Returns the number of records in the tree or node. Typically used for UNT-count of segments.

```
out.getcount()
#returns the numbers of records in outmessage.
```

getcountoccurrences(mpath)

Returns the number of records selected by mpath. Typically used to count number of LIN segments.

```
out.getcountoccurrences({'BOTSID':'UNH'},{'BOTSID':'LIN'})
#returns the numbers of LIN-records.
```

getcountsum(mpath)

Counts the totals value as selected by mpath. Typically used to count total number of ordered articles.

```
out.getcountsum({'BOTSID':'UNH'},{'BOTSID':'LIN'},{'BOTSID':'QTY','C186.6063':'12',
↪'C186.6060':None})
#returns total number of ordered articles.
```

Persist (Store Data)

Persist allows to store data for use in other messages/transactions.

Use cases:

- Store data from an incoming order, and use the data later for DESADV and/or INVOIC. Often the buyer want you to return data in the orders that are hard to store in your internal system.
- Is used in plugin alto_seperate_headers_details: input: 2 csv files, one with headers, one with details lines. This is processed into one document (so the headers and details are merged) using persist.

Details:

- Persist data is store in the bots database.
- You can store and retrieve **any** python data (python pickle is used).
- bots < 3.0: storage size (per item) is limited to 1024 positions; bots >= 3.0: unlimited.
- Parameter **maxdayspersist** in *bots.ini* controls how long the values are kept in bots. This is done using a timestamp for the persisted data. However, the timestamp is not updated if you update the information in the persist database. This can be done via a database trigger. In bots 3.2.0 and later the SQLite database includes this trigger (might be somewhat SQLite specific):

```
CREATE TRIGGER persist_update AFTER UPDATE OF content ON persist
BEGIN
    UPDATE persist SET ts = datetime('now','localtime')
    WHERE domein = new.domein and botskey = new.botskey ;
END
```

Example

You receive orders from buyer **XXX**. In the order they have a **reference number** that has to be returned in ASN's and invoices.

```
#in the order:
buyerID = inn.ta_info['frompartner']
po_number = inn.get({'BOTSID':'ST'},{'BOTSID':'BEG','BEG03':None})
ref_number = inn.get({'BOTSID':'ST'}{'BOTSID':'REF','REF01':'IP','REF02
↪':None})
transform.persist_add(buyerID, po_number, ref_number)
#now this referenced number is in persist; domain is the buyerID, key is the␣
↪po_number (that will be return eg in invoice)

#in the invoice find the domain and key:
buyerID = inn.ta_info['frompartner']     #via QUERIES in invoice in-house␣
↪grammar
po_number = inn.get({'BOTSID':'HEADER','PO_NUMBER':None})
#fetch reference number from persist:
ref_number = transform.persist_lookup(buyerID, po_number)
```

Functions

transform.persist_add(domain, key, value)

Add value to persist. If not possible, eg. because domain-key exists already, botslib.PersistError is raised.

transform.persist_update(domain, key, value)

Update the value. If domain-key does not exist: does not add it, gives no error.

transform.persist_add_update(domain, key, value)

Add, but if domain-key already exist: update.

transform.persist_lookup(domain, key)

Returns value. If domain-key does not exist: returns None.

transform.persist_delete(domain, key)

Deletes value. If domain-key does not exits: gives no error.

EAN/UCC Check Digits

Synonyms: GTIN, ILN, GLN, UPC, EAN, UAC, JAN

Can be used for UPC-A, UPC-E, EAN8, EAN13, ITF-14, SSCC/EAN-128 etc. These numbers end with a check-digit.

transform.checkean(EAN_number)

Returns True is if check-digit is OK, False if not. When not a string with digits, raises botslib.EanError

```
transform.checkean('8712345678906')
#returns 'True' for this EAN number.
```

transform.addeancheckdigit(EAN_number)

Returns EAN number including check digit (adds checkdigit).

```
transform.addeancheckdigit('871234567890')
#returns '8712345678906' for EAN number '871234567890'.
```

transform.calceancheckdigit(EAN_number)

Returns the checkdigit-string for a number without check-digit.

```
transform.calceancheckdigit('871234567890')
#returns '6' for EAN number '871234567890'.
```

Change and Delete Functions

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).
- For deleting all records (repeating records) use getloop() to access the loop, and delete within the loop.

Returns: if successful, True, otherwise False.

```
#delete a message date in a edifact INVOICD96AUNEAN008:
out.delete({'BOTSID':'UNH'},{'BOTSID':'DTM','C507.2005':'137'})
#delete DTM record where field C507.2005 = '137' ; DTM-record is nested under UNH-
↪record.
```

```
#delete all ALC segments in edifact message:
while message.delete({'BOTSID':'UNH'},{'BOTSID':'ALC'}):
    pass
```

Note: If you want to delete a field, you can use the change option and put as value **None**

```
lot = lin.get({'BOTSID':'line','batchnumber':None})
if not lot == None:
    lin.change(where={'BOTSID':'line','batchnumber':lot},),change={'batchnumber
↪':None})
#In this case I want to remove a wrong batchnumber from a specific supplier
```

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
- After change, searching stops (if more than one records exists for this mpath, only the first one is changed).
- For changing all records (repeating records) use getloop() to access the loop, and change within the loop.

```
inn.change(where={'BOTSID':'UNH'},{'BOTSID':'NAD','3035':'DP'}),change={'3035':'ST'})
#changed qualifier 'DP' in NAD record to 'ST'
```

Note: `where` must be a tuple; if you want to change the root of document, add a comma to make it a tuple.

```
inn.change(where={'BOTSID':'UNH'},),change={'S009.0054':'96A'})
#                                  ^ note comma here
```

Grammars

Definition: A grammar is a description of an edi-message.

- A grammar describes the records, fields, formats etc of an edi file.
- Bots uses grammars to parse, check and generate edi files.
- Grammars files are independent of the editype: a grammar for a csv files looks the same as a grammar for a x12 file.
- Grammar files are in: usersys/grammars/editype/grammar name.py

Learn grammar by example

Best way to get the idea of a grammar is to look at the (simple) example in the chapter. Consider the below CSV file:

```
HEADER,ordernumber1,buyer1,20120524
LINE,1,article1,24,description1
LINE,2,article2,288,
LINE,3.article3,6,description3
```

The corresponding grammer for this file would be:

```
from bots.botsconfig import *        #always needed


syntax = {                           #'syntax' section
     'field_sep' : ',',               #specify field separator
     'charset'   : 'utf-8',           #specify character-set
     }


structure = [                        #'structure' section
 {ID:'HEADER',MIN:1,MAX:999,LEVEL:[   #each order starts with a HEADER record
     {ID:'LINE',MIN:1,MAX:9999},      #nested under the HEADER record are the LINE␣
↪records, repeat up to 9999 times
     ]}
 ]


recorddefs = {                       #'recorddefs' section
 'HEADER':[                           #specify the fields in the HEADER record
     ['BOTSID','M',6,'A'],            #BOTSID is for the HEADER tag itself
     ['order number', 'M', 17, 'AN'], #for each field specify the format, max length,␣
↪M(andatory) or C(onditional)
     ['buyer ID', 'M', 13, 'AN'],
     ['delivery date', 'C', 8, 'AN'],
     ],
 'LINE':[
     ['BOTSID','M',6,'A'],
     ['line number', 'C', 6, 'N'],
     ['article number', 'M', 13, 'AN'],
     ['quantity', 'M', 20, 'R'],
     ['description', 'C', 70, 'R'],
     ],
 }
```

The example above is simple, but fully functional.

Sections of a grammar

A grammar file consists of these sections:

- *syntax*: parameters for the grammar like field separator, merge or not, indent xml, etc.
- *structure*: sequence of records in an edi-message: start-record, nested records, repeats.
- *recorddefs*: fields per record.
- *nextmessage*: to split up an edi file to separate messages.
- *nextmessageblock*: to split up a cvs-file to messages.

A section can be reused/imported from another grammar file. Purpose: better maintenance of grammars. Example: edifact messages from a certain directory use the same recorddefs/segments:

```
from recordsD96AUN import recorddefs
```

One edifact grammar consists of four parts. Example:

- edifact.py (contains syntax common to all edifact grammars)

- envelope.py (contains envelope structure and recorddefs common to all edifact grammars)
- recordsD96AUN.py (contains recorddefs common to all edifact D96A grammars)
- ORDERSD96AUN.py (contains structure specifically for ORDERS D96A)

Problems for some edifact grammars on sourceforge site

Sometimes you might meet this error for a grammar:

`GrammarError: Grammar "...somewhere...", in structure: nesting collision detected at record "etc etc".`

- This is the case eg with INVRPT D96A.
- UN says about this that you have to make additional choices in message; either you make some segments mandatory or leave out some segment groups.
- EANCOM did make such choices in their implementation guidelines.
- So: you can not the grammar directly, edit it according to your needs. This is according to what UN-edifact wants...

How to Get Grammars

- **edifact and x12**: There are grammars for all edifact and x12 messages on the bots sourceforge site.
- **xml**: use command line tool `bots-xml2botsgrammar.py` to generate a grammar from an xml file. Help for this utility script is provided using parameter `-h`
- **csv, fixed etc**: use examples in plugins, expand these

Syntax Parameters

- Syntax contains parameters that are used in reading or writing edi-files.
- The complete list of syntax parameters including default values is in bots/grammars.py (in the classes of the editypes).
- Syntax is a python dict (dictionary).

Example of syntax parameters:

```
syntax = {
        'charset'                   : 'uft-8', #character set is utf-8
        'checkfixedrecordtooshort' : True,     #check if fixed record is to short
        'indented'                  : True,     #xml: produced indented output
        'decimaal'                  : ',',      #decimal sign is ', '
        }
```

Usage and Overriding

Syntax parameters can be set at different places; these settings override (somewhat luke CSS). The order in which overriding is done:

- default values are in `bots/grammars.py` (per editype). **Do not change these values here**

- envelope grammar (eg for x12: `bots/usersys/grammars/x12/x12.py`, for edifact: `bots/usersys/grammars/edifact/edifact.py`)
- message grammar
- frompartner grammar (eg in `bots/usersys/partners/x12/partnerID.py`)
- topartner grammar (eg in `bots/usersys/partners/x12/partnerID.py`)

Example 1: edifact charset

- default value is UNOA
- value in envelope (`edifact.py`) is UNOA
- for invoices: a description is used so the message grammar for invoices has charset UNOC
- retailer ABC insists on receiving invoices as UNOA, so this is indicated in the topartner grammar.

Example 2: x12 element separator

- in grammar.py: `'field_sep':'*'` (bots default value)
- in x12.py: `'field_sep':'|'` (default value company uses when sending x12)
- retailer ABC insists: `'field_sep':'\x07'` (that is `\a`, or BEL)

List of most useful Syntax Parameters

Parameter	Direction	Description
add_crlfafterrecord_sep	Out	put **extra** character after a record/segment separator. Value: string, typically `\n` or
acceptspaceinnumfield	In	Do not raise error when numeric field contains only spaces but assume value is 0
allow_lastrecordnotclosedproperly	In	(csv) allows last record not to have record separator
charset	In	charset to use; (edifact, xml) is overridden by charset-declaration in content.
	Out	charset to use in output. Bots is quite strict in this.
checkcharsetin	In	what to do for chars not in charset. Possible values 'strict' (gives error) or 'ignore'
checkcharsetout	Out	what to do for chars not in charset. Possible values 'strict' (gives error), 'ignore' (sl
checkfixedrecordtoolong	In	(fixed): warn if record too long. Possible values: True/False, default: True
checkfixedrecordtooshort	In	fixed): warn if record too short. Possible values: True/False, default: False
checkunknownentities	In/Out	(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: read from UNA-string if present.
endrecordID	In	(fixed) end position of record ID; value: number, default 3. See startrecordID
envelope	Out	envelope to use; if nothing specified: no envelope - files are just copied/appended. I
envelope-template	Out	(template/html) the template for the envelope.
escape	In/Out	escape character used. Default: edifact: '?'.
field_sep	In/Out	field separator. Default: edifact: '+'; csv: ':' x12: '')
forcequote	Out	(csv) Possible values: 1 (quote only if necessary); 1 (always quote), 2 (quote only a
forceUNA	Out	(edifact) Always use UNA-segment in header, even if not needed. Possible values:
indented	Out	(xml, json) Indent message for human readability. Nice while testing. Indented mes
merge	Out	if merge is True: merge translated messages to one file (for same sender, receiver, n
namespace_prefixes	Out	(xml) to over-ride default namespace prefixes (ns0, ns1 etc) for outgoing xml. is a l

Ta

Parameter	Direction	Description
noBOTSID	In/Out	(csv) use if records contain no real record ID.
output	Out	(template) values: 'xhtml-strict'
pass_all	In	(csv, fixed) if only one recordtype and no nextmessageblock; False: pass record for
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: '
record_tag_sep	Out	(tradacoms) separator used after segment tag. Defaults: '='
replacechar	Out	(x12) if a separator value is found in the data, replace with this character. Default: '
skip_char	In	char(s) to skip, not interpreted when reading file. Typically 'n' in edifact.
skip_firstline	In	(csv) skip first line (often contains field names). Possible values: True/False/Integer
startrecordID	In	(fixed) start position of record ID; value: number, default 0. See endrecordID
template	Out	(template) Template to use for HTML-output.
triad	In	triad (thousands) symbol used (e.g. '1,000,048.35'). If specified, this symbol is skip
version	Out	(edifact,x12) version of standard generate. Value: string, typically: '3' in edifact or
wrap_length	Out	Wraps the output to a new line when it exceeds this length. value: number, default 0

Structure

Definition: the structure section in a grammar defines the sequence of records in a message.

A structure is required in a grammar except for template grammars. Example of a simple structure:

```
structure = [
{ID:'ENV',MIN:1,MAX:999, LEVEL:[         #envelope record
    {ID:'HEA',MIN:1,MAX:9999,LEVEL:[     #header record
        {ID:'LIN',MIN:0,MAX:9999},       #line record
        ]},
    ]}
]
```

The example above should be read as:

- ENV-record is required; max 999 repeats.
- 'Nested' under the ENV-record: HEA-record; per ENV max 9999 HEA-records; min is 1, so required.
- Per HEA-record max 9999 LIN-records; the LIN-record is not required (MIN=0).

Example of a more elaborate structure:

```
structure = [
{ID:'ENV',MIN:1,MAX:1, LEVEL:[         #envelope record: max one per file
    {ID:'HEA',MIN:1,MAX:9999,LEVEL:[   #header record of messageat least one message␣
↪per ENV, max 9999 messages
        {ID:'PAR',MIN:2,MAX:10},       #party record
        {ID:'ALC',MIN:0,MAX:9},        #allowances/charges per message
        {ID:'LIN',MIN:0,MAX:9999},     #line record
        ]},
    {ID:'TRL',MIN:1,MAX:1},            #each ENV record has a trailer record
    ]}
]
```

A structure consists of

1. A structure is a nested list of records.

2. Each record is a python dict.
3. **Per Record:**
    - **ID**: tag of record.
    - **MIN**: minimum number of occurrences.
    - **MAX**: maximum number of occurrences.
    - **LEVEL** (optional): indicate nested record. The nested records are in a list of records.
    - **QUERIES** (optional)
    - **SUBTRANSLATION** (optional). Advanced. Purpose: identify separate messages within within a standard envelope.

QUERIES

QUERIES in a structure are used to extract values from edi files before the mapping script starts. Example of an structure with QUERIES:

```
structure = [
{ID:'ENV',MIN:1,MAX:999,                    #envelope record
    QUERIES:{
        'frompartner':  {'BOTSID':'ENV','sender':None},   #extract sender from ENV␣
↪record.
        'topartner':    {'BOTSID':'ENV','receiver':None}, #extract receiver from ENV␣
↪record.
        'testindicator':({'BOTSID':'ENV'},{'BOTSID':'MESSAGE','test':None}), #extract␣
↪testindicator from 'deeper' level; note that this is in tuple (with brackets).
        },
    LEVEL:[
    {ID:'HEA',MIN:1,MAX:9999,LEVEL:[    #header record
        {ID:'LIN',MIN:0,MAX:9999},      #line record
        ]},
    ]}
]
```

Use cases of the information extracted by QUERIES: Think of frompartner, topartner, reference, testindicator etc.

1. To choose the right translation (QUERIES can extract frompartner, topartner, alt)
2. Update the database.
3. In a mapping script this information is inn.ta_info'.
4. Data in the envelope can be accessed in the mapping.
5. One of the most import uses is the extraction of **botskey**.
6. **Typical information extracted by QUERIES:**
    - frompartner
    - topartner
    - reference
    - testindicator
    - botskey
    - alt

QUERIES using multiple values (concatenated)

Pass a list into the QUERIES, and the values returned will be concatenated.

```
QUERIES:{
    'botskey': [ ({'BOTSID':'UNH'},'BOTSID':'BGM','C002.1001':None}), ({'BOTSID':'UNH
↪'},'BOTSID':'BGM','1004':None})],
}
```

QUERIES using a function

Sometimes you might need something more complex, eg. to extract a substring or do partner lookup and transformation. You can add a function in the grammar to do this and call it in QUERIES.

```
# frompartner needs some transformation...
import bots.transform as transform
def get_frompartner(thisnode):
     partner = thisnode.get({'BOTSID':'invoice','supplier':None})
     new_partner = transform.partnerlookup(partner,'attr1',safe=True)
     return new_partner

structure = [
{ID:'envelope',MIN:0,MAX:99999,LEVEL:[
     {ID:'invoice',MIN:0,MAX:99999,
          QUERIES:{
               'frompartner': get_frompartner,
               },
          LEVEL:[
               {ID:'line',MIN:0,MAX:99999},
```

Recorddefs

Definition: Recorddefs defines the layout of each record.

- Recorddefs is required, except for grammars for editypes `templatehtml`, `xmlnocheck`, `jsonnocheck`.
- Recorddefs is a dictionary; key is record ID/tag, value is list of fields.
- There is **always** a field `BOTSID` in a record; mostly BOTSID is the first field (fixed format is the exception).

Some specifics for different editypes

- **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).
- **Fixed**: bots has to know where record ID is. Default: first three positions in record; this can be set with syntax parameters
- **Csv**: Csv/Excel does not always have a record ID (all records in an edi-file are of the same type). Bots uses the name of the record as BOTSID if you use syntax parameter: `'noBOTSID':True`

Each field is a list of:

1. **field ID**: has to be unique (bots checks for this).
2. **M/C**: mandatory or conditional.
3. **length: Examples:**

```
['field name','C', 9, 'A'],        #max length
['field name','C', (3,9), 'A'],    #min length, max length. Use eg as:
['field name','C', (9,9), 'A'],    #field length should always be 9
['field name','C', 8.3, 'N'],      #length max 8 field, must have 3 decimals.
```

4. **format: Different per editype. Especially edifact, x12 and tradacoms have their own formatting codes, just use these. For**

   - **A** (or AN): alphanumeric.
   - **AR**: right aligned alphanumeric (fixed only)
   - **N: fixed decimals**
     - Numeric
     - Fixed numbers of decimals. If no decimals are indicated: integer
     - Incoming: format is checked.
     - Outgoing: bots formats this (rounding if needed, add leading zeros)
   - **NL**: (for fixed only) as fixed decimals, left aligned, trailing blancs
   - **NR**: (for fixed only) as fixed decimals, right aligned, leading blancs
   - **R: floating point**
     - Numeric
     - May have any number of decimals
     - Bots does no checking of formatting.
   - **RL**: (for fixed only) as floating point, left aligned, trailing blancs
   - **RR**: (for fixed only) as floating point, left aligned, leading blancs
   - **I: fixed decimals implicit. Example:**
     - Numeric
     - Fixed number of decimals. Is converted from/to 'normal' amount; rounded if outgoing.
     - There is no decimal sign in the field, the number of decimals is implicit from the definition. Eg:
     - Can be negative
     - **Example:**

```
['field name','C', 8.2, 'I'],    #max 8 long, last 2 positions are␣
↪decimals; eg for an amount.
                                 #Valid is eg:  12345
                                 #Bots converts this to 123.45 (and␣
↪vice versa)
```

- **D** (or DT): date. Either 6 or 8 positions; format YYMMDD or CCYYMMDD. Bots checks for valid dates both in- en outgoing.
- **T** (or TM): time. Either 4 or 6 positions; format HHMM or HHMMSS. Bots checks for valid times both in- en outgoing.

Composite fields

Edifact, x12 and tradacoms have composite fields. Example:

```
['S005', 'C',                       #composite field
    [
    ['S005.0022', 'M', 14, 'AN'],   #subfield nr1
    ['S005.0025', 'C', 2, 'AN'],    #subfield nr2
    ]],
```

A composite has:

- field ID; has to be unique (bots checks for this).
- M/C (mandatory/conditional).
- A list of subfields.

Note: Bots requires that each composite ID and sub-field has an unique ID. If a composite occurs more than once, do eg like:

```
['C090', 'C', [                         #composite contains 2 subfields with same ID
    ['C090.3286#1', 'M', 70, 'AN'],
    ['C090.3286#2', 'C', 70, 'AN'],
    ]],
['C542#1', 'M', [                       #composite occurs twice
    ['C542.9425#1', 'M', 3, 'AN'],
    ['C542.9424#1', 'C', 35, 'AN'],
    ]],
['C542#2', 'C', [                       #composite occurs twice
    ['C542.9425#2', 'M', 3, 'AN'],
    ['C542.9424#2', 'C', 35, 'AN'],
    ]],
```

Details of field format handling

- M/C of fields are always checked.
- min/max length of field are always checked.
- **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 '.'; use syntax-parameter 'decimaal' to change this ('decimaal' is Dutch, sorry about that. Noticed to late to change ;-)). Internally bots only uses/accepts decimal point.
- (incoming) leading zeros are removed
- (incoming) trailing zeros are kept
- '.45' is converted to '0.45'.
- '4.' is converted to '4'.
- Note that eg edifact numeric lengths do NOT include decimal sign and negative...

Utility for positions in fixed records

A grammar does not contains the position/offset of each field in a fixed record, but it sure is useful to have this. Add this code to the bottom of the grammar and run/execute the code:

```
if __name__ == "__main__":
    for key, record in recorddefs.items():
        length = 0
        for field in record:
            print '            ', field, '      #pos',length+1, length + field[2]
            length += field[2]
        print 'Record',key,'  has length ',length, '\n'
```

Slightly different version - output can be pasted back into grammar

```
if __name__ == "__main__":
    space = 0
    for key, record in recorddefs.items():
        for field in record:
            space = max(space,len(str(field))+1)
    for key, record in recorddefs.items():
        length = 0
        print '    \'' + key + '\':['
        for field in record:
            print '            ', (str(field) + ',').ljust(space), ' # pos',length+1,
↪'-', length + field[2]
            length += field[2]
        print '    ],'
```

Next Message

The `nextmessage` section of a grammar is used to split the messages in an edi-file; this way a mapping script receives one message at a time. Example:

```
structure=    [
{ID:'ENV',MIN:1,MAX:999,LEVEL:[          #envelope record
    {ID:'HEA',MIN:1,MAX:9999,LEVEL:[     #header record
        {ID:'LIN',MIN:0,MAX:9999},       #line record
        ]},
    ]}
]

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

Using this `nextmessage` the mapping script receives one HEA-record with the LIN-records under it. The sender and receiver of the envelope can be accessed via QUERIES.

Next Message Block

- The `nextmessageblock` section in a grammar is mostly used for csv-files consisting of one record-type.
- Think of excel, where every row has the same layout.
- Via nextmessageblock all subsequent records with eg the same ordernumber are passed as one message to the mapping script.
- This is best understood by an example, consider the below CSV file:

```
ordernumber1,buyer1,20120524,1,article1,24
ordernumber1,buyer1,20120524,2,article2,288
ordernumber1,buyer1,20120524,3.article3,6
ordernumber2,buyer2,20120524,1,article5,124
ordernumber3,buyer1,20120524,1,article1,24
ordernumber3,buyer1,20120524,2,article4,48
```

which will have the grammer:

```
from bots.botsconfig import *

syntax = {
    'field_sep' : ',',                  #specify field separator
    'noBOTSID'    : True,               #does not have record-ID's
    }

nextmessageblock = ({'BOTSID':'HEADER','order number':None})    #feed mapping script␣
↪with separate orders (where ordernumber is different)

structure = [
{ID:'HEADER',MIN:1,MAX:9999}          #only 1 record-type
]

recorddefs = {
'HEADER':[
    ['BOTSID','M',6,'A'],             #Note: BOTSID is ALWAYS needed
    ['order number', 'M', 17, 'AN'],
    ['buyer ID', 'M', 13, 'AN'],
    ['delivery date', 'C', 8, 'AN'],
    ['line number', 'C', 6, 'N'],
    ['article number', 'M', 13, 'AN'],
    ['quantity', 'M', 20, 'R'],
    ],
}
```

The mapping script in the translation receives 3 separate orders (so mapping script will run 3 times):

```
order with number 1:
ordernumber1,buyer1,20120524,1,article1,24
ordernumber1,buyer1,20120524,2,article2,288
ordernumber1,buyer1,20120524,3.article3,6

order with number 2:
ordernumber2,buyer2,20120524,1,article5,124
```

```
order with number 3:
ordernumber3,buyer1,20120524,1,article1,24
ordernumber3,buyer1,20120524,2,article4,48
```

Note: Use multiple fields for splitting up (bots > 3.1); Example: `nextmessageblock = ([{'BOTSID':'HEADER','order number':None},{'BOTSID':'HEADER','buyer ID':None}])`

Note: `nextmessageblock` works for fixed files with one type of records (bots > 3.1)

EDIFACT Character-Sets

Edifact uses its own naming of character-sets such as UNOA, UNOB, etc.. And some character-sets are quite typical or old.

Bots has 2 ways of supporting these edifact character-sets:

1. via specific character-sets UNOA and UNOB in `bots/usersys/charsets`
2. in bots.ini aliases are giving for edifact character-sets, eg: UNOC=latin1=iso8859-1

There are some problems with character-sets UNOA and UNOB:

- This is not always handle correct by some, eg they send UNOA with lower-case characters.
- In practice often <CR/LF> is send; officially these are not in UNOA or UNOB.

A default bots installation includes by default the UNOA and UNOB character set. These are not strict interpreted, but **tuned to reality**, so that they will not often lead to problems.

In the download **charsetvariations** on sourceforge site are some variations on these character-sets (stricter, less strict). Read the comments in these files first.

XML Namespaces

Dealing with xml namespaces can be a bit difficult in Bots. They can greatly complicate grammars and mappings. Often your EDI partner will want to send or receive XML with namespaces, but actually Bots does not need (or want) them as there are rarely any naming conflicts within a single XML file.

Where an XML file has only one namespace, usually the "default" namespace is used. eg. `xmlns="schemas-hiwg-org-au:EnvelopeV1.0"`

If the file has multiple namespaces, then namespace prefixes are used. These prefixes can be any value. eg. `xmlns:ENV="schemas-hiwg-org-au:EnvelopeV1.0"`

There are several ways to deal with namespaces in Bots:

1. **Ignore incoming XML namespace**

   For incoming XML files that are to be mapped into something else, you probably don't need the namespaces at all. In my opinion this is the "best" way. See an *example preprocessing script* to achieve this.

2. **Use incoming XML namespace**

When a namespace must be used, it is needed in many places (structure, recorddefs, mapping) but you want to specify the namespace string, which can be quite long, only in one place (DRY principle). This has several benefits:

- If it needs to change, this is easy to do in one place only
- grammar and mapping will be smaller, and look neater

For example, the incoming XML file may look like ...

```
<?xml version="1.0"?>
<Orders xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns="http://www.company.com/EDIOrders"
     targetNamespace="http//www.company.com/EDIOrders">
      <Order>
        <OrderNumber>239062415</OrderNumber>
        <DateOrdered>2014-02-24</DateOrdered>
        <LineCount>10</LineCount>
etc...
```

If you created your grammar with `xml2botsgrammar`, it probably has the namespace repeated over and over in structure and recorddefs, like this:

```
structure=    [
{ID:'{http://www.company.com/EDIOrders}Orders',MIN:1,MAX:1,LEVEL:[
    {ID:'{http://www.company.com/EDIOrders}Order',MIN:1,MAX:99999,
        QUERIES:{
             'botskey':      {'BOTSID':'{http://www.company.com/EDIOrders}
↪Order','{http://www.company.com/EDIOrders}OrderNumber':None},
            },
        LEVEL:[
        {ID:'{http://www.company.com/EDIOrders}OrderItems',MIN:0,MAX:1,
↪LEVEL:[
            {ID:'{http://www.company.com/EDIOrders}OrderLine',MIN:1,
↪MAX:99999},
            ]},
        ]},
    ]}
]

recorddefs = {
   '{http://www.company.com/EDIOrders}Orders':[
            ['BOTSID','M',256,'A'],
            ['{http://www.company.com/EDIOrders}Orders__targetNamespace',
↪'C',256,'AN'],
          ],
   '{http://www.company.com/EDIOrders}Order':[
            ['BOTSID','M',256,'A'],
            ['{http://www.company.com/EDIOrders}OrderNumber','C', 20,'AN
↪'],
            ['{http://www.company.com/EDIOrders}DateOrdered','C', 10,'AN
↪'],
            ['{http://www.company.com/EDIOrders}LineCount','C', 5,'R'],

# etc...
```

So, we want to improve this. In your grammar, first add the namespace as a string constant. This is the only place it should be specified, and everywhere else refers to it.

Note: If the XML has multiple namespaces, you can use the same technique and just add more constants (xmlns1, xmlns2, etc). Also include it in the syntax dict. This allows us to reference it later in mappings.

```
xmlns='{http://www.company.com/EDIOrders}'
syntax = {
    'xmlns':xmlns,
}
```

In your grammar, replace all instances of the namespace string with the constant, so it looks like this:

```
structure=    [
{ID:xmlns+'Orders',MIN:1,MAX:1,LEVEL:[
    {ID:xmlns+'Order',MIN:1,MAX:99999,
        QUERIES:{
             'botskey':      {'BOTSID':xmlns+'Order',xmlns+'OrderNumber
':None},
            },
        LEVEL:[
        {ID:xmlns+'OrderItems',MIN:0,MAX:1,LEVEL:[
            {ID:xmlns+'OrderLine',MIN:1,MAX:99999},
            ]},
        ]},
    ]}
]

recorddefs = {
   xmlns+'Orders':[
            ['BOTSID','M',256,'A'],
            [xmlns+'Orders__targetNamespace','C',256,'AN'],
          ],
   xmlns+'Order':[
            ['BOTSID','M',256,'A'],
            [xmlns+'OrderNumber','C', 20,'AN'],
            [xmlns+'DateOrdered','C', 10,'AN'],
            [xmlns+'LineCount','C', 5,'R'],

# etc...
```

Now in the mapping script, read the xmlns value from grammar.

```
import bots.grammar as grammar
xmlns = grammar.grammarread('xml',inn.ta_info['messagetype']).syntax[
'xmlns']
```

Then use it wherever needed in the mapping, like this:

```
# Get OrderNumber from XML with namespace
OrderNumber = inn.get({'BOTSID':xmlns+'Order',xmlns+'OrderNumber':None})
```

3. **Outgoing XML with only a default namespace**

   This can be done by defining xmlns as a tag, and setting it in mapping. It eliminates the need to use namespace prefix on every record and field. example grammar

```
recorddefs = {
    'shipment':
        [
        ['BOTSID', 'M', 20, 'AN'],
        ['shipment__xmlns', 'C', 80, 'AN'],    # xmlns is added as a tag␣
↪(note double underscore)
        ['ediCustomerNumber', 'M', 12, 'N'],
        ['ediParm1', 'M', 1, 'N'],
        ['ediParm2', 'M', 1, 'AN'],
        ['ediParm3', 'M', 1, 'AN'],
        ['ediReference', 'M', 35, 'AN'],
        ['ediFunction1', 'M', 3, 'AN'],
        ['ediCustomerSearchName', 'M', 20, 'AN'],
        ],
}
```

Example Mapping

```
# xmlns tag for shipment
out.put({'BOTSID':'shipment','shipment__xmlns':'http://www.company.com/
↪logistics/shipment'})
```

Example Output

```
<?xml version="1.0" encoding="utf-8" ?>
<shipment xmlns="http://www.company.com/logistics/shipment">
    <ediCustomerNumber>191</ediCustomerNumber>
    <ediParm1>4</ediParm1>
    <ediParm2>s</ediParm2>
    <ediParm3>d</ediParm3>
    <ediReference>SCN1022164911</ediReference>
    <ediFunction1>9</ediFunction1>
    <ediCustomerSearchName>SCHA</ediCustomerSearchName>
</shipment>
```

4. **Outgoing XML with default namespace prefixes (ns0, ns1, etc)**

   This is the default behaviour of the python elementtree module used in Bots.The actual prefix used is not important to XML meaning, so in theory your EDI partners should not care what prefixes you use.

   Example Grammar:

```
xmlns_env='{schemas-hiwg-org-au:EnvelopeV1.0}'
xmlns='{schemas-hiwg-org-au:InvoiceV3.0}'

syntax = {
        'xmlns_env':xmlns_env,
        'xmlns':xmlns,
        'merge':False,
        'indented':True,
}

nextmessage = ({'BOTSID':xmlns_env+'Envelope'},{'BOTSID':'Documents'},{
↪'BOTSID':xmlns+'Invoice'})

structure = [
{ID:xmlns_env+'Envelope',MIN:0,MAX:99999,
```

```
    QUERIES:{
        'frompartner':{'BOTSID':xmlns_env+'Envelope',xmlns_env+'SenderID
↪':None},
        'topartner':{'BOTSID':xmlns_env+'Envelope',xmlns_env+'RecipientID
↪':None},
        },
    LEVEL:[
    {ID:'Documents',MIN:0,MAX:99999,LEVEL:[
        {ID:xmlns+'Invoice',MIN:0,MAX:99999,
            QUERIES:{
                'botskey':({'BOTSID':xmlns+'Invoice',xmlns+'DocumentNo
↪':None}),
                },
            LEVEL:[
            {ID:xmlns+'Supplier',MIN:0,MAX:99999},
            {ID:xmlns+'Buyer',MIN:0,MAX:99999},
            {ID:xmlns+'Delivery',MIN:0,MAX:99999},
            {ID:xmlns+'Line',MIN:0,MAX:99999},
            {ID:xmlns+'Trailer',MIN:0,MAX:99999},
        ]},
    ]},
]},
]
```

Example Output

```
<?xml version="1.0" encoding="utf-8" ?>
<ns0:Envelope xmlns:ns0="schemas-hiwg-org-au:EnvelopeV1.0" xmlns:ns1=
↪"schemas-hiwg-org-au:InvoiceV3.0">
    <ns0:SenderID>sender</ns0:SenderID>
    <ns0:RecipientID>recipient</ns0:RecipientID>
    <ns0:DocumentCount>1</ns0:DocumentCount>
    <Documents>
        <ns1:Invoice>
            <ns1:TradingPartnerID>ID1</ns1:TradingPartnerID>
            <ns1:MessageType>INVOIC</ns1:MessageType>
            <ns1:VersionControlNo>3.0</ns1:VersionControlNo>
            <ns1:DocumentType>TAX INVOICE</ns1:DocumentType>
```

5. **Outgoing XML with specific namespace prefixes**

   Your EDI partner may request a specific namespace prefix be used; This is technically un-necessary and bad design, but they may insist on it anyway.

   Example Grammar

```
xmlns_env='{schemas-hiwg-org-au:EnvelopeV1.0}'
xmlns='{schemas-hiwg-org-au:InvoiceV3.0}'

syntax = {
        'xmlns_env':xmlns_env,
        'xmlns':xmlns,
        'namespace_prefixes':[('ENV',xmlns_env.strip('{}')),('INV',xmlns.
↪strip('{}'))], # use ENV, INV instead of ns0, ns1
        'merge':False,
        'indented':True,
}

structure = [
```

```
{ID:xmlns_env+'Envelope',MIN:0,MAX:99999,
    QUERIES:{
        'frompartner':{'BOTSID':xmlns_env+'Envelope',xmlns_env+'SenderID
↪':None},
        'topartner':{'BOTSID':xmlns_env+'Envelope',xmlns_env+'RecipientID
↪':None},
        },
    LEVEL:[
    {ID:'Documents',MIN:0,MAX:99999,LEVEL:[
        {ID:xmlns+'Invoice',MIN:0,MAX:99999,
            QUERIES:{
                'botskey':({'BOTSID':xmlns+'Invoice',xmlns+'DocumentNo
↪':None}),
                },
            LEVEL:[
            {ID:xmlns+'Supplier',MIN:0,MAX:99999},
            {ID:xmlns+'Buyer',MIN:0,MAX:99999},
            {ID:xmlns+'Delivery',MIN:0,MAX:99999},
            {ID:xmlns+'Line',MIN:0,MAX:99999},
            {ID:xmlns+'Trailer',MIN:0,MAX:99999},
        ]},
    ]},
]},
]
```

Example Output

```
<?xml version="1.0" encoding="utf-8" ?>
<ENV:Envelope xmlns:ENV="schemas-hiwg-org-au:EnvelopeV1.0" xmlns:INV=
↪"schemas-hiwg-org-au:InvoiceV3.0">
    <ENV:SenderID>sender</ENV:SenderID>
    <ENV:RecipientID>recipient</ENV:RecipientID>
    <ENV:DocumentCount>1</ENV:DocumentCount>
    <Documents>
        <INV:Invoice>
            <INV:TradingPartnerID>ID1</INV:TradingPartnerID>
            <INV:MessageType>INVOIC</INV:MessageType>
            <INV:VersionControlNo>3.0</INV:VersionControlNo>
            <INV:DocumentType>TAX INVOICE</INV:DocumentType>
```

Document View

Bots focuses mostly on edi files. Another useful way of looking at edi is to view at **business documents**: orders, asn's, invoices etc.

Bots support this, but to have this work satisfactory some configuration needs to be done. Essential is the use of document numbers (eg order number, shipment number, invoice number); in bots this is called `botskey`.

Usage

Once `botskey` is set correct for your documents, it can be used for:

- **Viewing and searching business documents:**
  - View last run: `bots-monitor->Last run->Document`

  - View all runs: `bots-monitor->All run->Document`
  - Select/search for documents: `bots-monitor->Select->Document`
- Set output *file name in a channel* or in a *communicationscript*

Configure botskey

This can be done in two ways:

1. Using QUERIES in the *grammar* of the incoming edi file.

```
# Example: botskey in a simple csv grammar
structure = [
    {ID:'LIN',MIN:1,MAX:99999,
        QUERIES:{
            'frompartner': ({'BOTSID':'LIN','AccountCode':None}),
            'topartner':   ({'BOTSID':'LIN','CustomerCode':None}),
            'botskey':     ({'BOTSID':'LIN','PurchaseOrderNo':None}),
        },
    }
]
```

2. In your *mapping script*

```
out.ta_info['botskey'] = inn.get({'BOTSID':'LIN','PurchaseOrderCode':None}
↪)
```

User Scripting

Bots has many places (exit points) where user scripting can be used. These user scripts are optional; bots will work without them but they provide great control over functionality for advanced users.

- Channels can have a *communicationscript*
- Routes can have a *routescript*
- Bots-engine can also have *it's own routescript*.
- Enveloping can be modified with an *envelopescript*.

Route Script for Bots-engine

A special routescript called botsengine.py can be added. This is called by the bots-engine at various points. You may wish to use this for your own logging, reporting or cleanup routines.

In Bots v3.2+ There are **pre** and **post** exit points for all runs, and for each type of run (command).

- pre (before any run)
- prenew
- preresend
- prerereceive
- preautomaticretrycommunicationcommunication
- precleanup

- postnew
- postresend
- postrereceive
- postautomaticretrycommunicationcommunication
- postcleanup
- post (after any run)

```
# user script example for some bots-engine exit points,
# showing the passed args.

# before any run
def pre(commandstorun,routestorun):
    print 'pre',commandstorun,routestorun

# before "new" run
def prenew(routestorun):
    print 'prenew',routestorun

# after "cleanup" run
def postcleanup(routestorun):
    print 'postcleanup',routestorun

# after any run
def post(commandstorun,routestorun):
    print 'post',commandstorun,routestorun
```

Example of doing something useful with exit points

The **postcleanup** exit point can be used to add your own daily tasks (eg. cleanup, backup or reporting capabilities). By default, bots runs a cleanup **once per day** at the end of the first run that day. (setting whencleanup=daily in bots.ini)

This example automatically activates or deactivates partners on the dates you configure for the partner (startdate, enddate)

```
import bots.botslib as botslib
import datetime

def postcleanup(routestorun):

        # activate any partners with a "start date" of today
        botslib.changeq(u'''UPDATE partner
                            SET active = 1
                            WHERE startdate = %(today)s''',
                            {'today':datetime.datetime.today().strftime('%Y-%m-%d')})

        # deactivate any active partners with an "end date" before today
        botslib.changeq(u'''UPDATE partner
                            SET active = 0
                            WHERE active = 1
                            AND  enddate is not null
                            AND enddate < %(today)s''',
                            {'today':datetime.datetime.today().strftime('%Y-%m-%d')})
```

Confirm/acknowledge/997

In edi often a confirmation (or acknowledgement) is needed. Bots has a built-in confirmation framework that supports:

- 997 (x12)
- CONTRL (edifact)
- MDN (email)
- APERAK (edifact)

Confirmations have 3 aspects:

- Bots sends a confirmation for an incoming edi-file.
- Bots asks a confirmation for an outgoing edi-file.
- Bots processes an incoming confirmation.

You can view the results of asked or send confirmations in `bots-monitor->All runs->Confirmations`. All send or asked confirmations are registered in bots if configured correct.

Examples in plugins

- **plugin `x12-850-856-810-997`**
  - generates a 997 for incoming orders, and dispatches this via a composite route.
  - inbound route is configured to process 997's.
- plugin `demo_mdn_confirmations` sends and receives email-confirmations (MDN)
- plugin `edifact_ordersdesadvinvoic` sends APERAK for incoming orders - if indicated in incoming order.

Send Confirmation

1. First configure correct processing of incoming edi-files.
2. **Configure confirmrules in `bots-monitor->Configuration->Confirmrules`. You can use several confirmation ru**
   - Indicate the kind of confirmation to send (997, CONTRL, etc).
   - Indicate how the rules works - per route, channel, messagetype, etc. (prior to bots 3.0, frompartner and topartner where swapped :-(
   - An option is to include first and than exclude using 'negativerule'. Bots first checks the positive rules, than the negative rules. Eg: send 997 for all incoming x12-files in a route, and exclude partner XXX.
3. Use a *composite route* to send the confirmations to the right destination.

Ask Confirmation

1. First configure correct processing of outgoing edi-files.
2. **Then configure `bots-monitor->Configuration->Confirmrules` You can configure several confirmation rules, b**
   - Indicate the kind of confirmation to ask (997, CONTRL, etc).

- Indicate how the rules works - per route, channel, messagetype, etc. (prior to bots 3.0, frompartner and topartner where swapped :-(
- An option is to include first and than exclude using 'negativerule'. Bots first checks the positive rules, than the negative rules. Eg: ask 997 for all outgoing x12-files in a route, and exclude partner XXX.

3. You will need to process the incoming confirmation.

Process incoming confirmation

- 997 (x12): an additional translation and mapping script is needed.
- CONTRL (edifact): an additional translation and mapping script is needed.
- MDN-confirmations: no additional comfiguration is needed.

Send 997/Acknowledgement for Incoming X12 Orders

- Before attempting to configure confirmations create and test a route for a typical X12 message e.g. 850.
- Once that is working correctly, confirmations can be configured for the route.
- Bots will now generate a 997 message for each 850 recipient. Notice that the 997 is in the same Route as the 850.
- Probably you will want the 997 to go back out to the partner that sent the 850. Use a *composite route* for this.
- Tell bots to send confirmations (997 message) in `bots-monitor->Configuration->Confirmrules`. Note that in the route established for the 850, there is a sequence number next to the routeID.
- The `seq` for the 850 is probably **1**. A new route-part must now be created to handle the 997 that bots has generated.
- The new route-part has the same routeID but a higher `seq` number (e.g. **2**). Now configure to route the 997 message that bots will generate.
- The 997 is an internally generated file. The user will have very little to do with its creation but everything to do with where the file goes (i.e. its route).
- Route with seq 2 will have `none` as its incoming channel, and does not translate. Its outgoing channel will most likely be to FTP or a VAN.
- The key to success is in using the "Filtering for outchannel" at the bottom of the "Change Route" page.
- In "Filtering for outchannel" for 'seq' 1, set toeditype and tomessagetype so that only the translated 850 file will take this part of the Composite Route.
- Now for seq 2 set toeditype to X12. This will cause only the X12 997 message to follow this part of the composite route.

Split, Merge and Envelope

- The advised way of working with bots is to have a translation work with a message (x12: transaction).
- In edi an incoming file will often have multiple messages.
- And often you will have to have multiple outgoing messages in one edi-file (merging, enveloping).

Splitting EDI files

One edi-file can contain several edi-message. Eg:

- one edifact file, multiple ORDERs.
- one x12 file, multiple 850's.
- export routine of your ERP software puts all exported invoices in one file
- email with multiple edi-attachments

A bots mapping works best for one message (one order, one invoice). So Bots splits up edi-files:

- Incoming email: different attachments are saved as a separate edi-files.
- Receiving zipped edi files: the files in the zip-file as saved as separate edi-files.
- For edifact, x12, tradacoms: interchanges are being split up to separate files.
- Incoming edi-files are being fed to the mapping-script as messages. This is done by *nextmessage* in the grammar syntax.
- **Spit within a mapping script. Think of eg splitting up a shipment to the different orders. There are 2 ways of doing this:**

  1. Write multiple message to the same file .
  2. Write each generated message to the same file, using alt translations.

Merge/Envelope EDI message

Reasons to merge/envelope edi files:

- Your ERP-system expects one file (message-queue) with fixed records
- Your edi-partners wants to limit the number of edi-files/interchanges received.
- costs: this can reduce transmission costs for some VAN's
- It is better to have one file with outgoing invoices than 165 separate files ;-)

Merge options:

- **Merging**: if the 'merge' parameter is set in the syntax of the outgoing message, bots will try to merge seperate messages to one file. Messages are only merged if: same from-partner, same to-partner, same editype, same messagetype, same testindicator, same characterset, same envelope.
- **Enveloping**: (edifact, x12, tradacoms) bots will envelope these messages (add UNB-UNZ for edifact, ISA-GS-GE-IEA for x12). Enveloping is independent from merging: bots can envelope without merging, or merge without enveloping.
- Write to a message-queue in outgoing channel: if you use a fixed filename in an outgoing channel, bots will append all messages to this file. This is often used in eg a configuration where all orders go to one file containing all incoming orders in fixed file format.

Envelope Scripting

You can use an envelopescript to do custom enveloping for edifact, x12, tradacoms, xml. Exit points that can be used:

1. **ta_infocontent(ta_info)**

- At start of enveloping process. ta_info cotains the values that are used to write the envelope; you can change these values.

2. **envelopecontent(ta_info,out)**
   - after bots has built the envelope tree you can make changes to the envelope tree by using put(), change() etc.

Functions should be in `bots/usersys/envelopescripts/<editype>/<editype>.py`, eg:

- `bots/usersys/envelopescripts/edifact/edifact.py`
- `bots/usersys/envelopescripts/x12/x12.py`

Example 1

Note that this affects all outgoing edifact documents. Add conditional logic if needed. In file `bots/usersys/envelopescripts/edifact/edifact.py`:

```
def envelopecontent(ta_info,out,*args,**kwargs):
    ''' Add extra fields to the edifact envelope.'''
    out.put({'BOTSID':'UNB','S002.0008': 'PARTNER1'}) #Interchange sender internal␣
↪identification
    out.put({'BOTSID':'UNB','S003.0014': 'PARTNER2'}) #Interchange recipient internal␣
↪identification
    out.put({'BOTSID':'UNB','0029': 'A'})             #Processing priority code
    out.put({'BOTSID':'UNB','0032': 'EANCOM'})        #Interchange agreement␣
↪identifier
```

Example 2

In file `bots/usersys/envelopescripts/edifact/edifact.py`:

```
def ta_infocontent(ta_info,*args,**kwargs):
    ''' function is called before envelope tree is made.
        values in ta_info will be used to create envelope.
    '''
    if ta_info['topartner'] == '1111111111111':
        ta_info['topartner'] = 'XXXXXXXXXXXXXXXXX'
        ta_info['UNB.S003.0014'] = '012345'
```

Example 3

In file `bots/usersys/envelopescripts/edifact/edifact.py`:

```
def envelopecontent(ta_info,out,*args,**kwargs):
    ''' function is called after envelope tree is made, but not written yet.
        manipulate envelope tree itself.
    '''
    if ta_info['topartner'] == '1111111111111':
        out.change(where={'BOTSID':'UNB'},change={'S003.0010': 'XXXXXXXXXXXXXXXXX'}
↪) #field S003.0010 (receiver) is written to envelope tree
        out.put({'BOTSID':'UNB','S003.0014': '012345'}) #field S003.0014 is written␣
↪to envelope tree
        ta_info['topartner'] = 'XXXXXXXXXXXXXXXXX'      #takes only care of changing␣
↪the partnerID in bots interface (does not change envelope itself)
```

EDI Partners

Plugin `demo_partnerdependent` at the bots sourceforge site demonstrates working with edi partners.

Partner look-up

Works with bots>=3.0. Often there is a need to retrieve data from a partner, using a `IDpartner`. Think of:

- check if partner is active
- get name/address of partner
- get senderID for partner

Recipe

Get the value in field `attr1` for the `IDpartner`. See partners in `bots-monitor->Configuration->Partners` for possible fields.

```
transform.partnerlookup('buyer1','attr1',safe=True)
```

Check if a partner exists, and is active. If not, raise an exception.

```
transform.partnerlookup('buyer1','active')
```

Value for safe	If a record matching your lookup does not exist, or the requested field is empty
safe=False (default)	An exception is raised. You can check for this in your script if required and take action
safe=True	No exception is raised, just returns the lookup value. eg: to lookup a user defined field for partner translation on some partners
safe=None (Bots>=3.2)	No exception is raised, returns None. eg: to get address lines where not all partners have addresses, but this is not an error:

Lookup using a field other than IDpartner

This is similar to reverse code conversion, but can look up any partner field and return any other partner field. Beware of performance issues if you have a large number of partners. Also there may be multiple matches if the lookup is not unique, only one is returned.

Get the value in field `name` by looking up value in `attr2`.

```
transform.partnerlookup('my attribute','name','attr2',safe=True)
```

Note:

- In bots<3.0 this was possible using code conversion. But this could lead to situations where partners where both in bots-monitor->Configuration->Partners & groups and in code conversions.
- Partners have more fields in bots>=3.0 like name, address, `free` fields.

Email addresses

- For email channels, you need to configure partners with their email addresses.
- Configure partners: `bots-monitor->Configuration->Partners`.
- A partner can have a different email address per channel, if not the default eail addres is used.
- Is it needed to configure the partners because:
  - for incoming messages, to determine whether an email is from a valid sender. Email from unknown partners are errors (think of spam).

  - for outgoing message, to determine the destination address.
- Configuring a partner for email:
  - Configure a partner: `bots-monitor->Configuration->Partners`.

Partner Groups

EDI partners can be assigned to groups. This might come in handy:

- Partner dependent translations (in `bots-monitor->Configuration->Translations`). Select a group here, and translation is done for all members of the group.
- Partner based filtering for routes/outchannel (in `bots-monitor->Configuration->Routes`): the outchannel is used for all members of the group.
- In selections you can use partner-groups.

Note: partner-groups do not work for confirmations.

How to

1. Create a group in `bots-monitor->Configuration->Partners & groups`. Indicate it is a group using tick-box **Isgroup**.
2. For the partners in a group, assign them to this group using **Is in groups**.

Plugin

Plugin 'demo_partnerdependent' at the bots sourceforge site demonstrates partner-groups.

Partner Dependent Syntax

- For outgoing messages it is possible to specify a partner dependent syntax.
- This is especially useful for x12 and edifact, for setting envelope values and partner specific separators.
- These parameters override the settings in the message grammar; you only need to specify the partner-specific parameters.

Note: no need to set partner specific separators for incoming messages; bots will figure this out by itself.

To set partner specific syntax parameters, create according to editype used:

- `bots/usersys/partners/x12/partnerid.py`
- `bots/usersys/partners/edifact/partnerid.py`

Example file with partner specific setting (x12):

```
syntax = {
    'ISA05'                 : 'XX',     #use different communication qualifier for␣
↪sender
    'ISA07'                 : 'ZZ',     #use different communication qualifier for␣
↪receiver
    'field_sep'             : '|',      #use different field separator
    }
```

Example file with partner specific setting (edifact):

```
syntax = {
          'merge':False,
          'forceUNA':True,
          'UNB.S002.0007':'ZZ',         # partner qualifier
          'UNB.S003.0007':'ZZ',         # partner qualifier
          }
```

Partner Specific Translation

Explain by example

You receive edifact ORDERSD96AUNEAN008 from several partners. Partner `retailer-abroad` fills the orders in a different way; the difference is so big that it is better to have a separate mapping script. Configure this like:

- one grammar for incoming edifact ORDERSD96AUNEAN008 message. (It is a standard message, isn't it?)
- one grammar for the inhouse import format. (We definitely want one import for all orders)
- note that the incoming edifact grammar uses QUERIES to determine the from-partner and to-partner before the translation.
- **make the 2 mapping scripts:**
  - mapping script `ordersedi2inhouse_for_retailerabroad.py` (specific for partner `retailer-abroad`) .
  - mapping script `fixed-myinhouseorde`‘ (for all other retailers).
- add `retailer-abroad` to partners (via bots-monitor->Configuration->Partners & groups).
- **Use 2 translations rules:**
  - edifact-ORDERSD96AUNEAN008 to fixed-myinhouseorder using mapping script `ordersedi2inhouse.py`
  - edifact-ORDERSD96AUNEAN008 to fixed-myinhouseorder using mapping script `ordersedi2inhouse_for_retailerabroad.py` for from-partner `retailer-abroad`

Often there are lots of similarities between the mappings - the ‘many similar yet different mappings’ problems. This can be *handled in bots* in a nice way.

Plugin

Plugin ‘demo_partnerdependent’ at the bots sourceforge site demonstrates partner-groups.

Organize Partner Specific Translations

Often in edi you need a number of very similar translations for different partners - the **many similar yet different mappings** problems. You want to organize your mapping scripts so that code is not duplicated many times, creating maintenance problems. A nice approach is:

1. Create a default mapping.
2. For some partners you can use the default mapping, for others use a *partner specific translation*
3. The partner-specific mappings import the default mapping and builds upon that (additional pre- or post-mapping).

Note: Plugin 'demo_partnerdependent' at the bots sourceforge site demonstrates partner-groups.

Consider the below example

The **default mapping** script file 'orders2idoc.py', translates edifact ORDERS to idoc ORDERS05)

```
def main(inn,out):
    # Order number
    out.put({'BOTSID':'EDI_DC40'},{'BOTSID':'E1EDK01'},{'BOTSID':'E1EDK02','QUALF':
↪'001',
                                    'BELNR':inn.get({'BOTSID':'UNH'},{'BOTSID':'BGM',
↪'C106.1004':None}))})

    # Buyer name
    out.put({'BOTSID':'EDI_DC40'},{'BOTSID':'E1EDK01'},{'BOTSID':'E1EDKA1','PARVW':'AG
↪',
                                    'BNAME':inn.get({'BOTSID':'UNH'},{'BOTSID':'NAD',
↪'3035':'BY','C056.3412':None}))})

    # blahblahblah.....lots more complex mapping code for the order
```

The **partner specific mapping** script file 'customer2_orders2idoc.py' for a **customer 2**. The default mapping is mostly OK, but a few changes are needed:

```
import orders2idoc              #here the default mapping is imported

def main(inn,out):
    #*** pre-mapping ******************
    # do partner-specific mapping before the default mapping eg to make the incoming␣
↪order "more standard" :-)
    # In this example:
    #     customer2 sends RFF+PR:BULK to indicate a stock order. Delete this and␣
↪change to BGM+120
    #     This must be done pre-mapping because we have complex mapping rules based␣
↪on BGM order type.
    if inn.get({'BOTSID':'UNH'},{'BOTSID':'RFF','C506.1153':'PR','C506.1154':None})␣
↪== 'BULK':
        inn.delete({'BOTSID':'UNH'},{'BOTSID':'RFF','C506.1153':'PR','C506.1154':'BULK
↪'})
        inn.change(where={'BOTSID':'UNH'},{'BOTSID':'BGM'}),change={'C002.1001':'120
↪'})


    #*** run the default mapping*****************
    orders2idoc.main(inn,out)


    #*** post-mapping ******************
    # Post-mapping to adjust or add to the mapped output.
    # Delete unwanted text that is sent on their orders
    out.delete({'BOTSID':'EDI_DC40'},{'BOTSID':'E1EDK01'},{'BOTSID':'E1EDKT2','TDLINE
↪':'TOTAL EXCL. GST AUD'})
    # Additional mapping: map buyer name from NAD+AB:
    out.put({'BOTSID':'EDI_DC40'},{'BOTSID':'E1EDK01'},{'BOTSID':'E1EDKA1','PARVW':'AG
↪',
```

```
            'BNAME':inn.get({'BOTSID':'UNH'},{'BOTSID':'NAD','3035':'AB','C056.3412
↪':None})})
```

Note:

- Make the default mapping is as generic as possible (eg. checking multiple fields).
- Do not not put any partner specific implementation mapping in here
- All mapping scripts are in the same directory (for the incoming editype)

Characters Sets

The good news: in general Bots handles character sets well. Incoming files are read using the character-set specified in the `incoming` syntax, outgoing files are written using the character-set as specified in the `outgoing` syntax. You do not need to do anything in the mapping, Bots does the character-set conversion automatically.

- Lots of information about this topic can be found on wikipedia.
- Information about the different character-sets in python is at python site.
- Outgoing character sets can be set in syntax; this can be done on envelope level, messagetype level of per partner.
- Sometimes character-set conversion is not possible, eg uft-8->ascii, as uft8 can contain characters not in utf8.
- For edifact there is an option to do character-set mapping (eg é->e); this is also possible for x12 (bit more work).
- Note that x12 can be tricky: the separators used can not be in the data. There is no good workaround for this, best way is to change the separators used.

Specifics of different character-sets

- Most used in edi is ascii, iso-8859-1, uft-8 (xml).
- Most familiar is ascii. Note that ascii has only 128 characters. One character is one byte.
- **The extended asccii character-sets.**
  - One character is always one byte.
  - The upper 128 (above ascii) are used as special characters (eg éëè).
  - These different character-sets are about displaying (on screen, print etc). If the texts is in iso-8859-1, but displayed as eg IBM850 looks *wrong*. Note that the content is still the same, it's only the display that is different.
  - In general these extended ascii character-sets will not be problematic in translations, as segment ID's/tags are in ascii and one character is one byte. In Bots the content of the message will just be fetched and passed to the outgoing message. So if a iso-8859-3 is handled as iso-8859-1 that will generally not be problematic.
  - Examples: iso-8859-1, windows-1252, iso-8859-2, iso-8859-3, IBM850, UNOC, latin1.
- **Unicode.**
  - Examples: utf-8, utf-16, utf-32, UCS-2
  - Unicode is designed at accommodate much more characters, think of eg Greek, Japanese, Chinese and Klingon characters.

  - One character is not one byte. (the different Unicode characters-sets use different representation schemes.
  - Much used is utf-8. Advantage of utf-8 is that the first 128 ascii characters are one byte; this is way uft-8 is *upward compatible* with ascii: a file with only ascii characters is the same in ascii and uft-8.
  - In Microsoft environments often utf-16 and utf-32 is used.
  - Fixed files can not have utf-8 character-set, as one character might be more than one byte.
- EBCDIC :related to (extended) ascii: one character is one byte. There are some variations of EBCDIC (eg extended EBCDIC).

A nasty situation is eg when one partner sends Unicode (eg utf-8), and another sends extended ascii (eg iso-8859-1). These extended characters work quite differently: Bots has to know what character-set is used before reading them, as these character-sets are treated quite differently. Best way to solve this is to have receive these files via different route(parts). Bots does no *guessing of character-sets*, as this is not appropriate for business data.

X12 Incoming

Character set as in envelope is used (if not set here default value of grammar.py is used (ascii)).

Some typical issues and solutions:

- If partner sends x12 as eg iso-8859-1, just specify this in the syntax of the envelope (`usersys/grammars/x12/x12.py`). Note that this is a system-wide setting, this will be used for all incoming 12. Should not be a problem, iso-8859-1 is a superset of ascii. The character-set of outgoing files should also be set to handle the extended characters. Also check your ERP software: what can it handle?
- Same solution works for utf-8

EDIFACT Incoming

Edifact has its own character-sets: UNOA, UNOB, UNOC, etc. In default bots setup:

- UNOA and UNOB have own character-set mapping `in usersys/charsets`.
- Other edifact character-sets are aliased in `config/bots.ini`, section charsets.
- Default UNOA and UNOB are not 100% strict but allow some extra characters like `<CR/LF>`.
- There are some variations of default UNOA/UNOB in sourceforge downloads.

Some typical issues and solutions:

- Edifact files have UNOA in them, but in fact they send more (UNOC). Solution: use `unoa_like_unoc.py` from downloads, save in `usersys/charsets` and rename to `unoa.py`
- Partner sends UNOC character-set, but my system can only handle ascii. Solution: use `unoa_like_unoc.py` from downloads; if you open this file you can see a character mapping (note that incoming is a different character mapping as outgoing). You can map eg incoming é->e (etcetc)
- Our system uses iso-8859-1, but partner can handle only UNOB. Solution: use `unoa_like_unoc.py` from downloads; if you open this file you can see a character mapping (note that incoming is a different character mapping as outgoing). You can map eg outgoing é->e (etc etc)

Deployment

This part of the wiki is about using bots in production. There are extra points to consider when deploying bots in a 24x7 production environment:

1. Consider the best way of *running bots-engine*.
2. When errors in edi-files occur, receive a *notification by email*.
3. *Use multiple environments*; having different environments for at least test and production is standard IT practice.
4. Consider if you need *extra archiving* fro edi files.
5. Install bots as a *service/daemon*.
6. Use limited rights for users.
7. If you use bots-monitor over the internet/outside your LAN, use HTTPS/SSL connection.
8. use apache as web server (instead of default cherrypy)
9. Use MySQL or PostgreSQL as database for Bots.
10. Use AS2 as communication method.
11. Bots has options to push changes from test to production.

Running Bots-Engine

Options for running bots-engine:

1. **Run automatic or manual**
    - Manual runs by using the `run` options in the menu.
    - *Schedule* bots-engine.
    - Use the *directory monitor/watcher*.
    - **This can be combined; eg**
        - schedule `new` every 15minutes
        - manually `rereceive` and `resend`
2. Direct or via *jobqueue-server*.
3. **Specify the routes to run:**
    - Run all routes. Default way of running (nothing specified).
    - Run all routes with excludes. Indicate in route if routes ought not to run in a default run.
    - Run specific routes: include route as parameters. Eg (command-line): `bots-engine.py myroute1 myroute2`
4. **Run new or other**
    - **new**: Via run-menu or command-line: `bots-engine.py`
    - **rereceive**: rereceive user indicated edi files. Via run-menu or command-line: `bots-engine.py --rereceive`
    - **resend**: resend user indicated edi-files. Via run-menu or command-line: `bots-engine.py --resend`

- **automaticretrycommunication**: resend edi-files where out-communication failed. Command-line: `bots-engine.py --automaticretrycommunication`

Scheduling Bots-Engine

- **Bots does not have a built-in scheduler. Scheduling is done by the scheduler of your OS.**
  - Windows: use eg Windows Task Scheduler.
  - Linux/unix: use eg cron.
- Bots-engine does not run concurrently (in parallel). If a previous run is still in progress, a new run will not start. From version 3.0 onwards, Bots includes an optional *job queue server* to use when scheduling Bots engine. Using this is recommended, to prevent discarding runs that overlap.
- **Strong advice**: when scheduling bots-engine, activate the sending of *automatic email-reports* for errors.

Possible scheduling scenarios

(command lines below are for Windows):

- If all (or most) routes can be run on the same schedule, then just schedule bots-engine "new" run as often as you need, Eg. every 5 minutes. To exclude some routes from this run, tick the Notindefaultrun box in the route advanced settings. These can then be scheduled separately by specifying route names on the command line.

```
c:\python27\python c:\python27\Scripts\bots-engine.py --new

c:\python27\python c:\python27\Scripts\bots-engine.py "my hourly route"
```

- If you have few routes but with varying schedules, then schedule them individually (by putting route names on the command line). Disadvantage: newly added routes are not automatically run, you must adjust your schedule.

```
c:\python27\python c:\python27\Scripts\bots-engine.py "my orders route"
↪"my invoice route"

c:\python27\python c:\python27\Scripts\bots-engine.py "my daily route"
```

- Consider whether you need to schedule retries periodically. Particularly with accessing remote servers, sometimes there may be communication errors that would be ok next time bots tries. Otherwise you will need to retry these yourself. File errors are not retried automatically because the same error will just come up again.

```
c:\python27\python c:\python27\Scripts\bots-engine.py --
↪automaticretrycommunication
```

My Setup (Mike)

I am using Windows task scheduler and Bots *job queue* is enabled. I have five scheduled tasks:

1. Bots-engine (every 5 minutes, 24x7)
2. Bots-engine-hourly (every hour on the hour)
3. Bots-engine-daily (1am daily)
4. Bots-engine-weekly (1am every Monday morning)
5. Bots-engine-monthly (1am first of the month)

Each task has a corresponding batch file in the scripts directory. This makes task configuration and changes easier; the scheduled tasks simply call the batch files. Within the batch files I use job2queue.py for adding jobs. Some add only a single job, while some add multiple jobs. (You could also put the command lines directly into Windows task scheduler, each one as a separate task). I use appropriate priorities for each job, as some times of the day Bots can get very busy. Several examples are shown below.

```
:: bots-engine.bat

:: Regular run of bots engine (eg. every 5 minutes, highest priority)
C:\python27\python.exe C:\python27\scripts\bots-job2queue.py -p1 C:\python27\python.
↪exe C:\python27\scripts\bots-engine.py --new
```

```
:: bots-engine-hourly.bat

:: Hourly monitoring alerts
C:\python27\python.exe C:\python27\scripts\bots-job2queue.py -p2 C:\python27\python.
↪exe C:\python27\scripts\bots-engine.py hourly_alerts

:: Hourly cleanup and low priority routes
C:\python27\python.exe C:\python27\scripts\bots-job2queue.py -p6 C:\python27\python.
↪exe C:\python27\scripts\bots-engine.py ftp_cleanup ProductionOrders RemitAdvice

:: automatic retry of failed outgoing communication
C:\python27\python.exe C:\python27\scripts\bots-job2queue.py -p7 C:\python27\python.
↪exe C:\python27\scripts\bots-engine.py --automaticretrycommunication
```

```
:: bots-engine-daily.bat

:: daily housekeeping
C:\python27\python.exe C:\python27\scripts\bots-job2queue.py -p3 C:\python27\python.
↪exe C:\python27\scripts\bots-engine.py daily_housekeeping

:: daily reporting &amp;amp; SAP data downloads
C:\python27\python.exe C:\python27\scripts\bots-job2queue.py -p9 C:\python27\python.
↪exe C:\python27\scripts\bots-engine.py daily_reports SAP_Expired_Contracts
```

Job Queue Server

Purpose of the bots jobqueue is to enable better scheduling of bots engine:

- ensures only a single bots-engine runs at any time.
- no engine runs are lost/discarded.
- next engine run is started as soon as previous run has ended.

Use of the job queue is recommended if you *schedule bots-engine*.

Details:

- Launch sequence from the queue can be controlled using different priorities when adding jobs.
- Other (non bots-engine) jobs can also be added to the queue if they need to be run **in between** bots-engine runs.
- If you add a duplicate of another job **already waiting on the queue** the request is discarded. This is because the job on the queue will perform the same action when it runs. If that job is already running, the new job **will** be added to the queue.
- Logging in `bots/botssys/logging/jobqueue.log`

- When using Bots monitor run-menu the job queue will be used if enabled in bots.ini; jobs are added with default priority of 5.
- In production you'll probably want to run bots-jobqueueserver as a *daemon process/service*.
- Full command-line usage instructions for bots-job2queue.py and bots-jobqueueserver.py when started up with `--help`
- **The bots job queue server does 3 things**
  - maintains a queue of jobs for bots-engine.
  - receives new jobs via the `bots-job2queue.py` (or via `bots-monitor->Run`)
  - launches a new job from the queue as soon as previous job ended.

Starting with the job queue

1. First, enable in *bots.ini* (jobqueue section, `enabled = True`).
2. Start the bots-jobqueueserver. Command-line: `bots-jobqueueserver.py`.
3. **Put jobs in the job queue:**
   - via menu using `bots-monitor->Run`
   - start from command-line (using `bots-job2queue.py`).
   - start from scheduler (using `bots-job2queue.py`).

Command examples

Job2queue on windows example 1:

```
c:\python27\python c:\python27\Scripts\bots-job2queue.py c:\python27\python␣
↪c:\python27\Scripts\bots-engine.py
```

Job2queue on windows example 2:

```
c:\python27\python c:\python27\Scripts\bots-job2queue.py -p3␣
↪c:\python27\python c:\python27\Scripts\bots-engine.py --new -Cconfigprod
```

Job2queue on windows example 3 (Adding other commands to the job queue):

```
c:\python27\python c:\python27\Scripts\bots-job2queue.py c:\program files\my_
↪program.exe my_parm_1 my_parm_2
```

Job2queue on linux example 4:

```
bots-job2queue.py bots-engine.py
```

Job2queue on linux example 5:

```
bots-job2queue.py -p3 bots-engine.py --new -Cconfigprod
```

Directory Monitor

This provides a method of monitoring specific **local** directories, and running Bots engine when files are ready to be processed.

Use of the directory monitor is optional. It may be useful for processing files that only arrive occasionally and at random times.

Prerequisites

- Directory monitor uses the *job queue*.
- Monitoring must be configured in *bots.ini* (`dirmonitorX` sections)
- Directory monitor *daemon process* must be started (`bots-dirmonitor.py`)

Return codes for bots-engine

Bots-engine uses the following return codes:

- 0: OK, no errors.
- 1: (system) errors: could not connect to database, not correct command line arguments, database damaged, unexpected system error etc.
- 2: bots ran OK, but there are errors/process errors in the run.
- 3: Database is locked, but **maxruntime** has not been exceeded. (use the job queue server to avoid this type of errors).

Return code **2** is similar/equivalent to the error reports by email.

Handling errors in Production

Different errors:

1. **Errors in edi files:**
    - inbound: contact your edi-partner they should send valid messages. (yes, and sometimes you will just have to adapt the definitions of the standards).
    - outbound: check out why your ERP-system sends invalid messages. Adapt the export-module.
2. **Communication errors**
    - inbound: no problem, next run bots will try again to fetch the messages correctly. No action needed (but if it keeps happinging....).
    - **outbound. You can handle this manually or automated:**
        - **manually:**
            (a) in the outgoing view, select edi-files with failed outbound communication and mark these as 'resend'
            (b) do: 'Run user-indicated resends'
            (c) check if communication was OK this time.
            (d) Note: number of resends is indicated in the outgoing view.
        - **automatic:**
            (a) schedule bots-engine with option `--automaticretrycommunication`.
            (b) all edi-files for which out-communication failed will be resend automatically.
            (c) if communication fails (again) in the automatic retry this is indicated in the error notification. So probably it is best to react on this.

(d) Note: number of resends is indicated in the outgoing view.

(e) The first time you use `automaticretrycommunication` nothing is resend, but automaticretry is initialised: all failed communications from that moment on will be resend. This is to prevent having all older failed communications resend.

(f) Scheduling: often bots is scheduled to run eg every 10 minutes, and automaticretry once an hour. For this type of scheduling use the *jobqueue server*.

Email Notifications for Errors

When bots runs scheduled, it is cumbersome to keep checking for errors in the bots-monitor. It is possible to receive an email-report in case of errors. Configure this:

1. Set option 'sendreportiferror' in bots/config/bots.ini to True.
2. In bots/config/settings.py set relevant data for email, eg like:

```
MANAGERS = (    #bots will send error reports to the MANAGERS
    ('name_manager', 'myemailaddress@gmail'),
    )
EMAIL_HOST = 'smtp.gmail.com'              #Default: 'localhost'
EMAIL_PORT = '587'              #Default: 25
EMAIL_USE_TLS = True        #Default: False
EMAIL_HOST_USER = 'username'         #Default: ''. Username to use for␣
↪the SMTP server defined in EMAIL_HOST. If empty, Django won't attempt␣
↪authentication.
EMAIL_HOST_PASSWORD = '*******'     #Default: ''. PASSWORD to use for the␣
↪SMTP server defined in EMAIL_HOST. If empty, Django won't attempt␣
↪authentication.
SERVER_EMAIL = 'botserrors@gmail.com'           #Sender of bots error␣
↪reports. Default: 'root@localhost'
EMAIL_SUBJECT_PREFIX = ''    #This is prepended on email subject.
```

3. To test if it works OK, restart the bots-webserver and bots-monitor->Systasks->Send test report. (I know, the restarting is annoying.).

Note: Email notifications are not sent while running acceptance tests.

Archiving of Files

- Bots always uses the `current archive`.
- The current archive is what can be seen in the bots-monitor: incoming files, outgoing files, document view etc.
- Edi-files are kept 30 days in the current archive (by default, can be set via parameter maxdays in `bots/config/bots.ini`).
- After this time the edi-files and data about their processing (like runs, incoming files, etc) are discarded.

Discussion about long term archiving

How long should edi-files be archived? There is no fixed rule about this. Some argue that the edi-files are temporary information carriers and that the real data is processed and archived in the ERP software of you and your edi-partner. But:

- for eg invoices there might be legal issues about keeping the 'original invoice'. (OTOH I get a big smile thinking of legal/tax people wading though these EDI-files. ;-))
- it might be needed to keep the original in case there are any errors or discrepancies that need to be investigated later.

The (default) 30 days of the current archive is a compromise between **keep all data always** and performance. If all data are kept, performance will degrade in the long run.

The long term archive

Bots has an option for a long term archive. This is how it works:

- Per channel: if you specify the 'Archive path' in a channel, all files coming in or going out for that channel are archived.
- for incoming files: archived as received (unchanged); for emails the (un-mimified) attachments are saved.
- for outgoing files: archived as send (unchanged). If outgoing communication fails: nothing is send, so nothing is archived.
- The long term archive contains copies of the files only. Once bots has cleaned details from it's database you will need to use tools like 'grep' to find what you need.
- Bots creates a sub-directory per date, eg. myarchive_path/20131202, myarchive_path/20131203, etc. The sub-directories contains the edi-files archived in that day.
- Within this daily directory, Bots uses unique numeric filenames by default.
- Edi-files are kept 180 days in the long term archive (by default, can be set via parameter maxdaysarchive in bots/config/bots.ini). Keeping files longer will not affect performance significantly. Of course the files will occupy disc space.

Example of archiving

- Archive path for channel `my_inchannel` is set to C:/edi/archive/my_inchannel.
- Archive path for channel `my_outchannel` is set to C:/edi/archive/my_outchannel.

After a few days this will looks someting like:

```
C:/edi/archive/
              my_inchannel/
                           20131202/
                                    13892848
                                    13892872
                                    13892876
                           20131203/
                                    13892991
                                    13893009
                           20131204/
                                    13893421
              my_outchannel/
                           20131202/
                                    13892861
                                    13892886
                           20131203/
                                    13893123
```

```
                    20131204/
                            13893479
```

Note: It is strongly advised to archive outside of the bots directories.

Additional options for long term archive

- archiveexternalname (setting in `bots/config/bots.ini`). If set to True, name of archived file is the name of the incoming/outgoing file. If this name already exists in the archive: a timestamp is added to the filename; eg. order.txt becomes order_112506.txt. External filenames are only used for some channel types where the channel defines the filename (file, ftp, ftps, ftpis, sftp, mimefile, communicationscript). Default setting in bots.ini is False. New in version 3.0.
- archivezip (setting in `bots/config/bots.ini`). If set to True, each archive folder will be a zip file. If you keep archives for a long time, zipping them can save a lot of disc-space; most EDI files compress to just a few percent of original size. Disadvantage: harder to search. Default setting is False. New in version 3.0.
- user scripting for archive path. See *communicationscript*. Note: please archive within the archive path as set in channel, else the cleanup routines for parameter maxdaysarchive will not function.
- user scripting for name of archive file. See *communicationscript*.

Run as Service/Daemon

A daemon is a computer program that runs continuously as a background process.

After Bots installation there are no service/daemon processes active. This is recommended for a production environment. Bots has several parts that you may want to run as services/daemons:

- bots-webserver.py
- bots-jobqueueserver.py (bots >= 3.0.0, is optional)
- bots-dirmonitor.py (bots >= 3.0.0, is optional)

Note: bots-engine itself is not a daemon process; bots-engine is best *scheduled*.

How these daemons are created and managed depends on the operating system being used:

- In linux/unix, you can start them as *Linux daemons*.
- In Windows, you can set them up as *Windows Services*.

Linux Daemons

Below are some examples that run bots-webserver or jobqueserver as a daemon:

Example 1

I have been starting bots-webserver on my Linux (CentOS) servers via an entry in rc.local and wanted to provide a means to gracefully shut down the process on reboot. There have been other examples of init scripts posted which do the trick, but I wanted to put something together that conformed to LSB. I took an example script I found online

and configured it for the bots.webserver process. Since CentOS doesn't handle pid files correctly when using LSB functions, I tweaked it to work around the issue, and it should work on most distributions without a lot of modification

```
### BEGIN INIT INFO
# Provides:          bots-webserver
# Required-Start:    $remote_fs $network
# Required-Stop:     $remote_fs $network
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: BOTS webserver daemon
# Description:       BOTS webserver daemon
### END INIT INFO

# Using the lsb functions to perform the operations.
. /lib/lsb/init-functions
# Process name ( For display )
NAME=bots-webserver
# Daemon name, where is the actual executable
DAEMON=/usr/bin/bots-webserver.py
# pid file for the daemon
PIDFILE=/var/run/bots-webserver.pid
# Arguments for the daemon
ARGS="> /dev/null 2>&1 &"

# If the daemon is not there, then exit.
test -x $DAEMON || exit 5

case $1 in
 start)
  # Checked the PID file exists and check the actual status of process
  if [ -e $PIDFILE ]; then
   pidofproc -p $PIDFILE $DAEMON && status="0" || status="$?"
   # If the status is SUCCESS then don't need to start again.
   if [ $status = "0" ]; then
    log_success_msg "$NAME process is already running"
    exit # Exit
   fi
  fi
  # Start the daemon.
  # Start the daemon with the help of start-stop-daemon
  # Log the message appropriately
  if start_daemon -p $PIDFILE $DAEMON $ARGS; then
   # For older LSB functions that don't handle -p argument correctly (i.e. CentOS)
   if [ ! -e $PIDFILE ]; then
    pidofproc $DAEMON > $PIDFILE
   fi
   log_success_msg "Starting the process $NAME"
  else
   log_failure_msg "Failed to start the process $NAME"
  fi
  ;;
 stop)
  # Stop the daemon.
  if [ -e $PIDFILE ]; then
   pidofproc -p $PIDFILE $DAEMON > /dev/null && status="0" || status="$?"
   if [ "$status" = 0 ]; then
    killproc -p $PIDFILE $DAEMON
    /bin/rm -f $PIDFILE
    log_success_msg "Stoppping the $NAME process"
```

```
   fi
  else
   log_warning_msg "$NAME process is not running"
  fi
  ;;
 restart)
  # Restart the daemon.
  $0 stop && sleep 2 && $0 start
  ;;
 status)
  # Check the status of the process.
  if [ -e $PIDFILE ]; then
   pidofproc -p $PIDFILE $DAEMON > /dev/null && log_success_msg "$NAME process is␣
↪running" && exit 0 || exit $?
  else
   log_warning_msg "$NAME process is not running"
  fi
  ;;
 reload)
  # Reload the process. Basically sending some signal to a daemon to reload
  # it configurations.
  if [ -e $PIDFILE ]; then
   killproc -p $PIDFILE $DAEMON -signal USR1
   log_success_msg "$NAME process reloaded successfully"
  else
   log_failure_msg "$PIDFILE does not exists"
  fi
  ;;
 \*)
  # For invalid arguments, print the usage message.
  echo "Usage: $0 {start|stop|restart|reload|status}"
  exit 2
  ;;
esac
```

Example 2

Works on debian/ubuntu servers and uses `start-stop-daemon`.

```
#! /bin/sh
#
# uses 'start-stop-daemon' , which is used in debian/ubuntu
#
NAME=bots-webserver
PIDFILE="/var/run/$NAME.pid"
DAEMON="/usr/local/bin/bots-webserver.py"
DAEMON_ARGS="-cconfig"

case "$1" in
    start)
        echo "Starting "$NAME" "
        start-stop-daemon --start --verbose --background --pidfile $PIDFILE --make-
↪pidfile --startas $DAEMON -- $DAEMON_ARGS
        ;;
    stop)
        echo "Stopping "$NAME" "
        start-stop-daemon --stop --verbose --pidfile $PIDFILE
        rm -f $PIDFILE
```

```
        ;;
    restart)
        echo "Restarting "$NAME" "
        start-stop-daemon --stop --verbose --pidfile $PIDFILE
        rm -f $PIDFILE
        sleep 1
        start-stop-daemon --start --verbose --background --pidfile $PIDFILE --make-
↪pidfile --startas $DAEMON -- $DAEMON_ARGS
        ;;
    \*)
        echo "Usage: ""$(basename "$0")"" {start|stop|restart}"
        echo "    Starts the bots webserver as a daemon."
        echo "    Bots-webserver is part of bots open source edi translator (http://
↪bots.sourceforge.net)."
        exit 1
        ;;
esac

exit 0
```

Example 3

A script for starting the job queue server as a upstart in Ubuntu. Add the following file: `/etc/init/bots-jobqueue.conf`

```
description "Bots Job queue server"
author "bots@yourmail.com"

start on runlevel [2345]
stop on runlevel [!2345]

respawn

exec bots-jobqueueserver.py
```

Windows Services

If running Bots on a Windows server, you can create services to keep the important background processes running. This is the equivalent of a **daemon** process in Linux.

Prerequisites

- **srvany.exe** - This is a Microsoft utility included in the Windows Server Resource Kit Tools.
- **sc.exe** - The SC command is included by default in most Windows installations and is also available in the resource kit.
- Python and Bots are already installed and working, of course.

Procedure

- Copy `srvany.exe` to *C:WindowsSystem32*.
- Open a command prompt and enter the following commands, according to the service required. Note: position of equal signs and spaces must be exactly as shown.

```
sc create "Bots Webserver" binPath= "C:\Windows\System32\srvany.exe"␣
↪start= auto DisplayName= "Bots Webserver"
sc description "Bots Webserver" "This is the webserver for Bots EDI␣
↪translator."
```

```
sc create "Bots Job Queue" binPath= "C:\Windows\System32\srvany.exe"
↪start= auto DisplayName= "Bots Job Queue"
sc description "Bots Job Queue" "Provides job queue and launch
↪functionality for Bots EDI Translator"
```

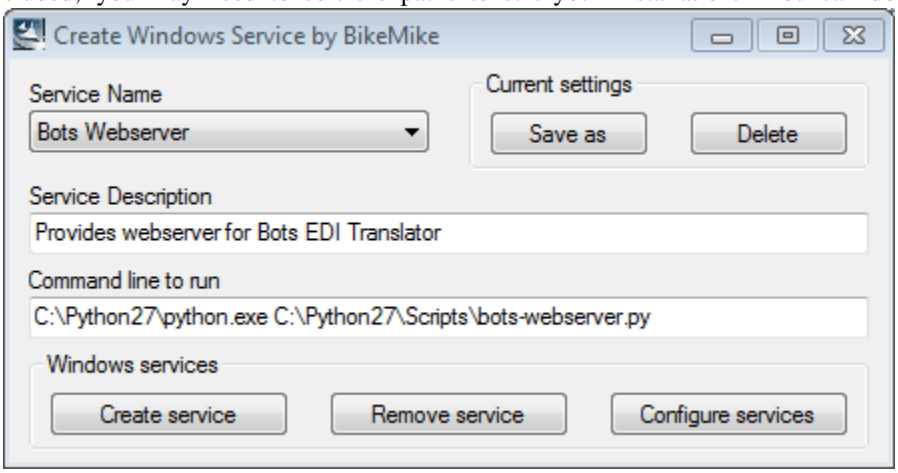
```
sc create "Bots Directory Monitor" binPath= "C:\Windows\System32\srvany.
↪exe" start= auto DisplayName= "Bots Directory Monitor"
sc description "Bots Directory Monitor" "Monitors one or more
↪directories for new files and creates Bots jobs to process them"
```

- Run regedit and navigate to `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<Service Name>`
- In the right hand pane of regedit, right click, New, Key, call it `Parameters`.
- Click the new Parameters key to select it.
- In the right hand pane, right click, New, String value, call it `Application`.
- Double click Application, enter the command to run the required Bots script. This will vary according to your installed location and Bots version, eg.

```
C:\Python27\python.exe C:\Python27\Scripts\bots-webserver.py
C:\Python27\python.exe C:\Python27\Scripts\bots-jobqueueserver.py
C:\Python27\python.exe C:\Python27\Scripts\bots-dirmonitor.py
```

- Run `services.msc` to start/stop/configure your new services.

If any of the above doesn't make sense to you, I have created a small free utility program to do it all. This is a general purpose program for creating services. My service configuration for Bots is included; you may need to edit the paths to suit your installation. You can download the program from here.



Reference links

- http://support.microsoft.com/?kbid=137890
- http://support.microsoft.com/?kbid=251192

Multiple Environments

There is more than one way of configuring multiple environments. Directories relevant for separate environments are:

- `bots/config`: global configuration; eg values for usersys, botssys, database connection, port number.

- `bots/botssys`: storage of edi files, database, loggings etc.
- `bots/usersys`: grammars, mappings, route scripts etc.

Use config parameter

Each bots program *start script* accepts a parameter (-c) that indicates which config directory to use. Within config there are settings for botssys and usersys, so this gives us the ability to have separate environments. The default, if this parameter is not used, is **config**.

By example:

1. Start with configuration with the default `config`, `usersys` and `botssys` directories in bots directory.
2. Purpose is to create a 2nd environment (env2).
3. **Make copies of 3 directories within bots directory. Advised is to use the same name-suffix:**
   - Make copy of `config`, name it `config-env2`
   - Make copy of `botssys`, name it `botssys-env2`
   - Make copy of `usersys`, name it `usersys-env2`
4. **Edit the configuration files in `config-env2` and change (at least) the following settings:**
   - in bots.ini

     ```
     botssys = botssys-env2
     usersys = usersys-env2
     ```

   - settings.py

     ```
     DATABASE_NAME = os.path.join(PROJECT_PATH,
     'botssys-env2/sqlitedb/botsdb')
     ```

5. Start bots scripts using the -c parameter to refer to the new environment, eg:

   ```
   $ bots-webserver.py -cconfig-env2
   $ bots-engine.py -cconfig-env2
   ```

Note: On linux the use of symlinks in bots directory might be useful.

Use different computers

- Your production environment is on a server.
- Your development environment is on your desktop/laptop PC.
- In this way, you can replicate exactly the same setup (same python version etc), and transfer things from one to the other once tested.

Using different python installations

- I used this for a long time for windows.

- I had python2.5 and 2.6 installed; Bots in python 2.6 was my development environment; bots in python 2.5 was production.
- This is a very simple way to have 2 environments in windows.

Using python virtualenv tool

- virtualenv is a tool to create isolated Python environments.
- Each environment has a separate installation of bots and its dependencies, so different versions can be tested.
- You can activate and deactivate the environments as needed.
- You can use in combination with method 1 to have *config environments within python environments*.

Installation

1. Install python: You have probably done this already. Make sure you have the paths of your `python27` and `python27\scripts` included in the system path variable.
2. Install distribute: Download distribute_setup.py and invoke it using python.

   ```
   >> python C:\Path\to\distribute_setup.py
   ```

3. Install Pip: Download get-pip.py (right click, save as) and invoke it using python.

   ```
   >> python C:\Path\to\get-pip.py
   ```

4. Install Virtualenv: Once Pip is installed, installing any other package (that is available in the Python Package Index) is easy.

   ```
   >> pip install virtualenv
   ```

5. Install C and C++ compilers (optional):
   - Not all python libraries are pure python, some may contain C code that must be compiled to install successfully.
   - Download MinGW, run it, and select the C and C++ compilers to install.
   - When complete, edit the file `Python27\Lib\distutils\cygwincompiler.py` and remove `-mno-cygwin` from lines `322-326`.
   - This flag is no longer supported.
   - You also need to add a setting to each virtual environment that needs to use the compilers (described in the next section).
   - Make sure you have the path of your `MinGW\bin` included in the system path variable.

Create your environments

Technically your virtual environments can be stored and scattered anywhere, but it makes sense to keep them all grouped together. I suggest you create a "root" directory for environments, each environment will be a subdirectory, eg.

```
D:\> mkdir PythonEnv
D:\> cd PythonEnv
D:\PythonEnv>
```

Create as many environments as you need (for ease of use, keep environment names short but meaningful and without spaces), eg. bots310

```
D:\PythonEnv> virtualenv bots310
New python executable in bots310\Scripts\python.exe
Installing Setuptools.....................................................
↪...
.........................done.
Installing Pip...........................................................
↪...
.....................done.

D:\PythonEnv>
```

If you installed compilers in the previous section and want to use them for this environment, then edit Libdistutilsdistutils.cfg within the environment folder, and add

```
[build]
compiler=mingw32
```

Activate and deactivate environments

To activate an environment, use the activate command in it's script directory. Notice your command prompt changes to show the active environment in brackets. Only one environment can be **activated** at a time, in order to install modules etc.

```
D:\PythonEnv> bots310\scripts\activate
(bots310) D:\PythonEnv>
```

To deactivate the current environment, use the deactivate command. Notice your command prompt changes back to show no active environment in brackets.

```
(bots310) D:\PythonEnv> deactivate
D:\PythonEnv>
```

Optional; create an `activate.bat` file in your environment root directory. This gives you a shortcut to activate environments.

```
REM activate.bat gives you a shortcut to activate python environments
REM eg. activate bots310
call "%1\scripts\activate"
```

Install Bots in a virtual environment

First, activate the required environment. Install Bots and dependencies using pip (don't use the Bots Windows installer, because it installs to the default python folder)

- Install Bots from local downloaded .tar.gz file

```
(bots310) D:\PythonEnv> pip install .\bots-3.1.0.tar.gz
Unpacking d:\pythonenv\bots-3.1.0.tar.gz
  Running setup.py egg_info for package from file:///d7C
↪%5Cpythonenv%5Cbots-3.1.0.tar.gz

Installing collected packages: bots
  Running setup.py install for bots

Successfully installed bots
Cleaning up...

(bots310) D:\PythonEnv>
```

- Install Django (version 1.4.x)

```
(bots310) D:\PythonEnv>pip install Django==1.4.6
Downloading/unpacking Django==1.4.6
  Downloading Django-1.4.6.tar.gz (7.7MB): 7.7MB downloaded
  Running setup.py egg_info for package Django

Installing collected packages: Django
  Running setup.py install for Django

Successfully installed Django
Cleaning up...

(bots310) D:\PythonEnv>
```

- Install cherrypy (latest)

```
(bots310) D:\PythonEnv> pip install cherrypy
Downloading/unpacking cherrypy
  Downloading CherryPy-3.2.4.tar.gz (424kB): 424kB downloaded
  Running setup.py egg_info for package cherrypy

Installing collected packages: cherrypy
  Running setup.py install for cherrypy

Successfully installed cherrypy
Cleaning up...

(bots310) D:\PythonEnv>
```

- Install Genshi (optional, required for template-html output)

```
(bots310) D:\PythonEnv> pip install Genshi
Downloading/unpacking Genshi
  You are installing a potentially insecure and unverifiable␣
↪file. Future versions of pip will default to disallowing␣
↪insecure files.
  Downloading Genshi-0.7.tar.gz (491kB): 491kB downloaded
  Running setup.py egg_info for package Genshi

    warning: no files found matching 'COPYING' under directory
↪'doc'
    warning: no previously-included files matching '*' found␣
↪under directory 'doc\logo.lineform'
    warning: no previously-included files found matching
↪'doc\2000ft.graffle'
    warning: no previously-included files matching '\*.pyc'␣
↪found anywhere in distribution
Installing collected packages: Genshi
  Running setup.py install for Genshi
    building 'genshi._speedups' extension
    C:\MinGW\bin\gcc.exe -mdll -O -Wall -ID:\Python27\include -
↪ID:\PythonEnv\bots310\PC -c genshi/_speedups.c -o build\temp.
↪win32-2.7\Release\genshi\_speedups.o
    C:\MinGW\bin\gcc.exe -shared -s build\temp.win32-2.
↪7\Release\genshi\_speedups.o build\temp.win32-2.
↪7\Release\genshi\_speedups.def -LD:\Python27\Libs -
↪LD:\PythonEnv\bots310\libs -LD:\PythonEnv\bots310\PCbuild -
↪lpython27 -lmsvcr90 -obuild\lib.win32-2.7\genshi\_speedups.pyd
```

```
    warning: no files found matching 'COPYING' under directory
↪'doc'
    warning: no previously-included files matching '\*' found␣
↪under directory 'doc\logo.lineform'
    warning: no previously-included files found matching
↪'doc\2000ft.graffle'
    warning: no previously-included files matching '\*.pyc'␣
↪found anywhere in distribution
Successfully installed Genshi
Cleaning up...

(bots310) D:\PythonEnv>
```

- Install cdecimal (optional, improves *performance*)

  This will not install/compile correctly on Windows using pip, and the installer is an msi (not exe) so easy_install won't work either. You can install it manually though; the two files needed are `cdedimal.pyd` and `cdecimal-2.3-py2.7.egg-info` and they go in your virtual environment's `site-packages` directory. There are two ways to get these files.

  - Install cdecimal in the default python folder (eg. `C:\python27\lib\site-packages`) using the windows msi installer, then copy the two files to your virtual environment.
  - Extract the files from the windows msi installer using a tool such as universal extractor.

- Install pycrypto and paramiko (optional, required for sftp channels)

  These will not install/compile correctly on Windows using pip. Instead, I used easy_install with a downloaded Windows installer exe.

```
(bots310) C:\PythonEnv>easy_install pycrypto-2.1.0.win32-py2.7.
↪exe

Processing pycrypto-2.1.0.win32-py2.7.exe
creating 'c:\docume~1\adadmi~3\locals~1\temp\1\easy_install-
↪wy9qt4\pycrypto-2.1.
0-py2.7-win32.egg' and adding 'c:\docume~1\adadmi~3\locals~
↪1\temp\1\easy_install
-wy9qt4\pycrypto-2.1.0-py2.7-win32.egg.tmp' to it
Moving pycrypto-2.1.0-py2.7-win32.egg to␣
↪c:\pythonenv\bots310\lib\site-packages
Adding pycrypto 2.1.0 to easy-install.pth file

Installed c:\pythonenv\bots310\lib\site-packages\pycrypto-2.1.0-
↪py2.7-win32.egg
Processing dependencies for pycrypto==2.1.0
Finished processing dependencies for pycrypto==2.1.0

(bots310) C:\PythonEnv>easy_install paramiko-1.7.6.win32.exe

Processing paramiko-1.7.6.win32.exe
creating 'c:\docume~1\adadmi~3\locals~1\temp\1\easy_install-
↪mwtlnu\paramiko-1.7.
6-py2.7-win32.egg' and adding 'c:\docume~1\adadmi~3\locals~
↪1\temp\1\easy_install
```

```
-mwtlnu\paramiko-1.7.6-py2.7-win32.egg.tmp' to it
Moving paramiko-1.7.6-py2.7-win32.egg to␣
↪c:\pythonenv\bots310\lib\site-packages
Adding paramiko 1.7.6 to easy-install.pth file

Installed c:\pythonenv\bots310\lib\site-packages\paramiko-1.7.6-
↪py2.7-win32.egg
Processing dependencies for paramiko==1.7.6
Finished processing dependencies for paramiko==1.7.6
```

Start Bots Webserver in the virtual environment

You can simply start the webserver manually from commandline. The variable `%VIRTUAL_ENV%` contains the path to the activated environment. Using the start command causes a new console window to be opened.

```
(bots310) D:\PythonEnv> start python %VIRTUAL_ENV%\scripts\bots-webserver.
py
```

Alternatively you can modify the environment's activate command (it is a small batch file of about 25 lines). Edit `%VIRTUAL_ENV%\scripts\activate.bat` and add the above command to the end. Then every time you activate the environment, the webserver is started too. Same for jobqueueserver (if required). I also add a window title. eg.

```
start "webserver (bots310)" python %VIRTUAL_ENV%\scripts\bots-webserver.
py
```

Running multiple Bots environments concurrently

Yes, this is possible. Although only a single environment can be "activated", once they are created you can run multiple bots webservers and engines simultaneously from different environments, if configured with different ports.

bots.ini - use different ports for each environment. eg.

```
[settings]
#port used to assure only one instance of bots-engine is running. default:␣
↪28081
port = 28091

[webserver]
#port at which at bots-gui is server. default is 8080
port = 8090

[jobqueue]
# Port to use for the job queue xmlrpc server (on localhost). Default: 28082
port = 28092
```

settings.py - add new setting for the session cookie name, and use a different name for each environment. This allows simultaneous login to each environment from the same browser. eg. use the environment name (default is `sessionid`)

```
#*********sessions, cookies, log out time*************************
SESSION_COOKIE_NAME = 'bots310'
```

User Rights

To add Bots users, select **Users** on the **SysTasks** menu. The Change User screen has a section called **Permissions** which contains the following settings controlling that user's access.

Active

If this box is checked, the user is able to log on to the webserver and view run details.

Note: A bug in version 3.1.0 prevents users with only Active status from changing their own passwords. To be fixed in v3.2.0.

Staff Status

These users may also see the Configuration and Run menu items, depending on specific user permissions (see below). Only the permitted menu items will be visible. If you do not give any extra permissions, only an empty configuration menu is shown.

Superuser Status

These users can also see the Systasks menu. They have access to all Configuration and Run options without being specifically given permission. The default user (bots) is a superuser.

Note: Recommendation: Create user names according to your company policy. Disable the default user or change the password; anybody can find out the default user/password for bots.

User Permissions for Staff status users

Here you can add permissions for specific configuration objects. Only some of the permissions shown in the list are relevant; they control access to the corresponding configuration menu options. Any permissions not listed below can be ignored as they have no effect in Bots GUI. Permissions can be given directly to a user, or to a user group which is then added to multiple users.

- bots route
- bots channel
- bots translation
- bots partner
- bots confirm rule
- bots user code
- bots user code type
- bots mutex - gives access to the Run menu [new in version 2.2.0]

You can use the search box under Available user permissions, then select the required permissions. Use the Ctrl-key to select multple items. Normally, you would give add/change/delete permissions together for the required configuration.

eg. To allow a staff user to configure channels, give these permissions.

```
bots | channel | Can add channel
bots | channel | Can change channel
bots | channel | Can delete channel
```

To allow a staff user access to the Run menu, give this special permission. (mutex is a table that bots uses to indicate a database lock while the engine runs)

```
bots | mutex | Can change mutex
```

Bots-Monitor over HTTPS

- This feature is introduced in bots 2.1.0.
- This works with cherrypy > 3.2.0 in combination with python 2.6 or 2.7. In python 2.5 this works (using extra dependency pyOpenssl) but gives problems with reading plugins.

Procedure

1. You will need an SSL certificate. You can use self-signed certificates.
2. In `bots/config/bots.ini` uncomment options ssl_certificate and ssl_private_key (in section webserver), and set these to the right value, eg:

```
ssl_certificate = /mysafeplace/mycert.pem
ssl_private_key = /mysafeplace/mycert.pem
#In this example certificate and private key are in the same pem-file.
```

3. Restart bots-webserver
4. Point your browser to the right https-address, eg: https://localhost:8080

Note:

- If you are using cherrypy and receive an error "ssl_error_rx_record_too_long" try the 2-line fix
- You can create self-signed certificates here
- You can configure port=443 in bots.ini then just point your browser to https://localhost

---

Use Apache2 as Webserver

- Bots uses cherrypy webserver Out of the Box and in order to scale it, you need to run it on an Apache2 Web Server
- One Advantage is that with this is that apache2 will take care of staring the Bots Monitor so there is no need to create a daemon for the bots webserver.

Procedure

1. Ensure that apache2 is installed and running on the system. (try `httpd -M`)
2. The `mod_wsgi` package is needed for this set up, so install it and add the line `LoadModule wsgi_module modules/mod_wsgi.so` to apache `httpd.conf` file.
3. Restart the apache server, run the command `httpd -M` and you should see the `wsgi_module` at the end.
4. Now we need to shuffle a few directories, create a base directory called **bots_app** in your home folder (do not run under root)
5. Move the folders **botssys**, **usersys**, **media** and **config** from the bots installation directory to **bots_app** folder.
6. Do not forget to create sym links for these folders in the installation dir to this new location.
7. Now add the following files to the **bots_app** folder:

```
##bots.wsgi

import sys
import django.core.handlers.wsgi
import mod_wsgi

#set PYTHONPATH...not needed if bots is already on PYTHONPATH
#sys.path.append('/usr/local/lib/python2.7/dist-packages')
from bots import apachewebserver

config = mod_wsgi.process_group
apachewebserver.start(config)
application = django.core.handlers.wsgi.WSGIHandler()
```

```
##apache2.conf

Listen {PORT}
NameVirtualHost \*:{PORT}
## Use this if you run into socket errors on linux
WSGISocketPrefix /var/run/wsgi

<VirtualHost \*:{PORT}>
   WSGIScriptAlias /{PATH TO UR HOME}/bots_app/bots.wsgi
   WSGIDaemonProcess config user={System User} group={System User Group}
   WSGIProcessGroup config

## Use this section only when enabling https for bots app
   SSLEngine on
   SSLCertificateFile /path/to/www.example.com.cert
   SSLCertificateKeyFile /path/to/www.example.com.key

   Alias /media {PATH TO UR HOME}/bots_app/media

   <Directory {PATH TO UR HOME}/bots_app/>
        Order deny,allow
        Allow from all
   </Directory>

   <Directory {PATH TO UR HOME}/bots_app/media>
        Order deny,allow
        Allow from all
   </Directory>

</VirtualHost>
```

8. Add execute permission to the `bots.wsgi` file and add the line `Include {PATH TO UR HOME}/bots_app/apache2.conf` at the end of the apache2 `httpd.conf` file.
9. Restart apache server and open `http://{ip}:{port}/`, you should see the bots welcome screen. If you get permission errors or images are not loading then correct permissions need to be given to **bots_app** and its parent directory.
10. Once the server is running the logs will be generated in the folder `botssys/logging` as `apache_webserver_config.log`
11. The script `bots-webserver.py` should **not** be used now as apache2 will automatically start up the application.
12. Add `botsengine_path = /usr/local/bin/bots-engine.py` in the Bots ini file.

Use MySQL or PostgreSQL as Database

- Default bots uses a SQLite database. This is included in the standard installation and works out-of-the-box. Performance of SQLite is good.
- Bots always uses `utf-8` in the database communication.
- Database tables are installed using `django-machinery`. See django docs.
- After installation, you may want to *migrate some data*.
- You might have to manually add a database trigger if using *persist* functions.

PostgreSQL

1. Install PostgreSQL.
2. Install `psycopg2` as python database adapter.
3. Login in command line client of database.
4. Create database (CLI) : `CREATE DATABASE botsdb WITH ENCODING 'UTF8';`
5. Create user (CLI): `CREATE USER bots WITH PASSWORD 'botsbots';`
6. Give user rights (CLI): `GRANT ALL PRIVILEGES ON DATABASE botsdb TO bots;`
7. make sure database accepts connections over (non-local) TCP/IP: change parameter `listen_addresses` in `postgresql.conf`; add/change setting for user in `pg_hba.conf`.
8. Set connection parameter in bots configuration in `bots/config/settings.py`. Examples are provided - see comments in `settings.py`
9. **Create the required tables (CLI):**
    - <CLI>: `django-admin syncdb --settings='bots.config.settings`
    - Or when bots is not installed in the default directory <CLI>: `django-admin syncdb --pythonpath='path to bots' --settings='bots.config.settings'`
    - Django asks for name etc of default user.

MySQL

1. Install MySQL.
2. Install `mysql-Python` as python database adapter.
3. Login in command line client of database.
4. Create database (CLI): `CREATE DATABASE botsdb DEFAULT CHARSET utf8;`
5. Create user (CLI): `CREATE USER 'bots' IDENTIFIED BY 'botsbots';`
6. Give user rights (CLI): `GRANT ALL PRIVILEGES ON botsdb.* TO 'bots';`
7. Make sure database accepts connections over (non-local) TCP/IP: change parameter `binds` in `/etc/mysql/my.cfg`.
8. Set connection parameter in bots configuration in `bots/config/settings.py`. Examples are provided - see comments in `settings.py`
9. **Create the required tables:**

- <CLI>: `django-admin syncdb --settings='bots.config.settings'`
- Or when bots is not installed in the default directory <CLI>: `django-admin syncdb --pythonpath='path to bots' --settings='bots.config.settings'`
- Django asks for name etc of default user.

MySQL on Windows

1. Download MySQL community server msi installer. (tested version 5.5.20)
2. Run the installer, do a **typical** installation and follow the prompts. On the final screen, make sure *Launch the MySQL Instance Configuration Wizard* box is ticked.
3. **When the configuration wizard starts, make the following selections:**
   - Detailed configuration
   - Server machine (you may choose developer for your bots test environment)
   - Transactional database
   - DSS/OLAP (20 connections)
   - Enable TCP/IP and Strict mode (defaults)
   - Best support for multilingualism (**UTF8**)
   - Install as Windows service (default)
   - Modify security settings; enter a root password and write it down.
   - Execute configuration
   - Create a shortcut to `C:\Program Files\MySQL\MySQL Server 5.5\bin\MySQLInstanceConfig.exe` in case you want to modify any of these settings later.
4. Go to `Start > Programs > MySQL > MySQL Server > MySQL Command Line Client`. Enter your root password when prompted. You should now have a `mysql>` command prompt.
5. Enter the following MySQL commands at the prompt:

```
mysql> CREATE DATABASE botsdb DEFAULT CHARSET utf8;
 Query OK, 1 row affected (0.01 sec)

mysql> CREATE USER 'bots' IDENTIFIED BY 'botsbots';
 Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL ON botsdb.* TO 'bots';
 Query OK, 0 rows affected (0.03 sec)
```

6. Download and install *MySQL-python for Windows to suit your python version (tested ''MySQL-python-1.2.3.win32-py2.7.exe'*)
7. Set connection parameters in bots configuration in `bots/config/settings.py`. MySQL example is provided, I only changed HOST.

```
#MySQL:
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'botsdb',
```

```
        'USER': 'bots',
        'PASSWORD': 'botsbots',
        'HOST': 'localhost',  #database is on same server as Bots
        'PORT': '3306',
        'OPTIONS': {'use_unicode':True,'charset':'utf8','init_command':
↪'SET storage_engine=INNODB'},
        }
    }
```

8. Create the required tables from a command prompt. Django asks for name etc of superuser. (enter user: bots, password: botsbots)

```
> D:\python27\python.exe D:\Python27\lib\site-packages\django\bin\django-
↪admin.py syncdb
  --settings=bots.config.settings

Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_admin_log
Creating table confirmrule
Creating table ccodetrigger
Creating table ccode
Creating table channel
Creating table partnergroup
Creating table partner
Creating table chanpar
Creating table translate
Creating table routes
Creating table filereport
Creating table mutex
Creating table persist
Creating table report
Creating table ta
Creating table uniek

You just installed Django's auth system, which means you don't have any␣
↪superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'mike'): bots
E-mail address: bots@bots.com
Password: botsbots
Password (again): botsbots
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

9. Now start bots-webserver and log in as bots.

Note: The database is stored in C:/ProgramData/MySQL/MySQL Server 5.5/Data/ by default (on Windows 7). To move the database, do the following

1. Stop the MySQL service
2. Move the `/Data/` folder only to your required location (eg. `D:/MySQL Server 5.5/Data/`)
3. Make sure the permissions are moved with it. `NETWORK SERVCE` must have full control
4. Change the setting `datadir` in `C:/ProgramData/MySQL/MySQL Server 5.5/my.ini` to indicate the new folder location

```
# Path to the database root
datadir=D:/MySQL Server 5.5/Data
```

5. Restart the MySQL service. If it will not start, check permissions.

Using AS2 with Bots

- AS2 (Applicability Statement 2) is a specification about how to transport data securely and reliably over the Internet.
- Security is achieved by using digital certificates and encryption.
- There are multiple forms of AS2, most used is AS2 over HTTP(S).
- AS2 (over HTTP) is a server protocol i.e. to use AS2 you will need an AS2-server.
- Each partner must have their own AS2 communications gateway.
- All AS2 software is designed to be interoperable. There are several open source implementations, as well as commercial software and hosted services.

How to use AS2 with Bots

Several threads in the mailing list about this:

- http://groups.google.com/group/botsmail/browse_thread/thread/b98c49f9ec9c4e8a/15e9ed6dbae987f5
- http://groups.google.com/group/botsmail/browse_thread/thread/54c7462a32fb9741/c594d71fb214b8c1
- http://groups.google.com/group/botsmail/browse_thread/thread/9bd0cc65461478b/0bbc729349392a1f

AS2 Software

- http://sourceforge.net/projects/mec-as2/
- http://sourceforge.net/projects/openas2/
- https://github.com/abhishek-ram/pyas2
- http://www.cecid.hku.hk/hermes.php
- http://stackoverflow.com/questions/7426951/is-anyone-using-python-for-gs1-xml-and-as2-edi

Configuration Change Management

Having different environments for at least test and production is a sound IT practice. But having different environments also brings problems:

- push changes in a controlled way to production-environment
- are changes done right, and does the existing configuration still run right?
- keep test-environment in line with production-environment

To handle these problems bots has some features called *configuration change management*.

Bots configuration change management

Configuration change management in bots has 2 aspects:

1. **Use *tools* for:**
    - Comparing differences in configuration of environments
    - Pushing the changes from test->production environment in a controlled, automated way
2. **Use of *isolated acceptance test* to:**
    - check if acceptance test runs OK in test
    - check if acceptance test runs OK in production
    - make the test-environment (very) equal to production-environment

Configuration change management works best if both aspects are combined. See *recipe* for this.

Isolated Acceptance Testing

This is a *mode* for testing which is **isolated**:

- No external communication: all I/O is from/to file system
- System state does not change (eg no increased counters, no test runs in system, etc)

Idea is to run an acceptance test without affecting your system (or the system of your edi-partner).

Note: all plugins since 20130101 are suited for isolated acceptances tests.

Advantages

1. Isolated acceptance tests are 'repeatable' (because same counters are used, same time is used etc). This is great for testing: each run of a test gives the same results. This makes it easy to compared the results automatically.
2. By using isolated acceptance tests your test-environment can be equal to production-environment. So you can use standard directory comparison tools to push changes test->production.
3. As the acceptance tests are isolated, this means an acceptance test can be run in a production-environment without affecting this environment. This way you can verify that after a change, all still runs as before in your production environment.

Running isolated acceptance tests

- In all channels set `testpath`. Testpath should point to a directory with test-files.
- Set option `runacceptancetest` in bots.ini to `True`.
- Run `new`
- Check results of run with what you expect
- Delete runs/files from acceptance test (menu: `Systasks->Bulk delete`; select **only** `Delete transactions in acceptance testing`.)

Acceptance tests in plugins

Plugins since 20130101 can be run as acceptance tests. About the plugins with build-in acceptance test:

- `testpath` for incoming files are in `botssys\infile`
- `testpath` for outgoing files go to `botssys\outfile`
- Each plugin also contains the the expected outgoing files (`botssys\infile\outfile`); these are used to compare the results.
- **Included is a route script (`bots/usersys/routescripts/bots_acceptancetest.py`):**
  - **before run**: discard directory `botssys\outfile`.
  - **after run**: global run results are compared with expectation (#in, #out. #errors, etc)
  - **after run**: compare files in `botssys\outfile` with expected files in `botssys\infile\outfile`
  - the output of these comparisons can be seen in terminal (`dos-box`)

Implementation Details

- channel-type is set to `file`.
- channel-path is set to the value in `testpath`. if testpath is empty: use path.
- channel-remove is set to `off`: no deletion of incoming files.
- error-email: not send.
- Fixed date/time in envelopes and in mappings (if function transform.strftime() is used)
- Counters/references: fixed; counters are not incremented
- Incoming files are always read in same order.
- outgoing-filename options: date/time is fixed
- No archiving
- **Additional user exits are run. User exits are in file `usersys/routescripts/botsacceptancetest.py`:**
  - **before run**: function pretest. use eg to empty out-directories etc
  - **after run**: function post-test. This can be used to check results, compare files etc

After running an isolated acceptance test, the `reports/filereports/data-files/etc` generated during acceptance-testing can be deleted via: `menu:Systasks->Bulk delete`; select only `Delete transactions in acceptance testing`.

Note:

- GUI does not have the results of post-test script (runs after automaticmaintanance); view this in terminal/dos-box.
- Communication scripts: script should check explicitly if in acceptance test and act accordingly.
- Database communication: script should check explicitly if in acceptance test and act accordingly.

Recipe to push test->production

Recipe to use a standard directory comparison tool to manage the differences in configuration between test and production:

1. For both environments, make configuration index file (`menu->Systasks->Make configuration index`)
2. **Compare both environments using a directory comparison tool. What you should compare is:**
    - All files in `bots/usersys`.
    - Note that the file `bots/usersys/index.py` contains the configuration as in the database (routes, channels, partners).
3. Push changes using the tool.
4. And read the configuration index file (`menu->Systasks->Read configuration index`) to the database.

Details

1. If the configuration index file is generated all configuration is in usersys (routes, mappings, partners etc).
2. By using the *isolated acceptance test* both environments can be **very equal**.
3. The configuration index file can also be generated by command line tool but cannot be read by command line tool.
4. Look like it is possible to use a version control system. I have not tried it, but recipes and experiences are welcome.

Build a good test-set

When you do changes in your edi environment, you want to know that every *ran as before*. What can be helpful for this is to use a *isolated acceptance test* for this.

This is easy to demonstrate:

- Download a plugin from bots sourceforge site and install it (not on your production environment;-))
- In `config/bots.ini` set `runacceptancetest` to True
- Run bots-engine via command-line

How this works: in acceptance tests an extra script `usersys/routescripts/bots_acceptancetest.py` runs when all routes are finished. This script does 2 things:

- It compares the results (#files received, errors, send, etc) with the expected results. If results are different you'll see this (on command-line window).
- The files in `botssys/infile/outfile` are compared with files as generated by the run in `botssys/outfile`. If results are different you'll see this (on command-line window).

Some things to look at when you build a test-set:

- Use the `acceptance test path` in the channels to point to your file system for incoming and outgoing channels (prevents using communication methods like pop3, ftp, etc).
- Test file in `botssys/infile` are added to plugins (I find this very convenient).

- Counters (for message numbers, file names etc (via unique()) are the same in every run, so results are the same every run.
- If date/times need to made, use `transform.strftime()` for this; it is like pythons `time.strftime()` but gives always the same date/time in acceptance testing.

Troubleshoot/FAQ

Why is it called 'bots'? Where are the bots?

EDI is about cooperation in the supply chain (collaboration, tying together, etc). That is true, but on the other hand it is also a clash of cultures. In Dutch **bots** is like **crash**.

bots is also the nickname my grandfather-in-law used for my girlfriend. It was not clear where that name came from, probably that was how she pronouced her name (betsy) when she learned to talk.

In Dutch **bots** is singular.

but...its just a name.

General FAQ

Files are 'stuck' Basically bots expects input to lead to output. When a file is 'stuck' input did not lead to output. Somehow the setup you made is not OK. Files go in, but nothing comes out. Often this is the problem: Have you put any *filtering* on the output of your route? (under 'Filtering for outchannel' in the route configuration). Filtering allows you send only particular files via an out-channel; but a file that does not match one of the filters will end up "stuck" because Bots does not know where to send it. If you have only one outchannel, there is no need for filtering.

Incoming edi-files are read over and over again In the in-channel, there is a 'remove'-option. Turn it on to have incoming files removed after reading. The 'not-remove' option is mostly used in development.

Incoming files can be seen but are not picked up by Bots (particularly with unc paths) This is usually a permission problem. Remember that if you are running Bots Engine as a daemon/service it may be running under a system user account. Make sure the user has rights to the folder/files, or run the engine with a special user account that does have the required rights. It is best to use the *jobqueue server* so that engine runs started from the GUI also run under the special user account.

Error in Internet Explorer: 'ValueError: invalid literal for int() with base 10' This has to do with the compatibility-settings of IE (tools->compatibility view settings). Text from Microsoft: 'Websites that were designed for earlier versions of Internet Explorer might not display correctly in IE8, IE9, or IE10. When you turn on Compatibility View, the webpage you're viewing, as well as any other webpages within the website's domain, will be displayed as if you were using an earlier version of Internet Explorer. So if the compatibility view is used this error occurs, else it goes OK.'

Error in Internet Explorer: 'DoesNotExist?: report matching query does not exist.' Same problem as last question: compatibility-settings of IE (tools->compatibility view settings).

"root" of incoming message is empty; either split messages or use inn.getloop This may occur with incoming files that have only one record format. Even if the whole file is one edi message, you need to use *nextmessageblock* in your incoming grammar.

Outgoing files to sftp server result in 'IOError: [Errno 13] Requested operation is not supported.' Error occurs during out-communication via FTP. Some FTP-servers do no support 'APPE' command (only 'STOR'). To resolve add '{overwrite}' to filename, eg: `{overwrite}OUT_{datetime:%Y%m%d%H%M%S}_*.edi`. Please make sure that filename is always unique by using `*`.

error_perm: 502 Command not implemented. Error occurs during out-communication via FTP. Some FTP-servers do no support 'APPE' command (only 'STOR'). To resolve add '{overwrite}' to filename, eg: `{overwrite}OUT_{datetime:%Y%m%d%H%M%S}_*.edi`. Please make sure that filename is always unique by using `*`.

Another option is to set the 'FTP active mode' in channel (under FTP specfic).

ftp server gives a timeout when writing file (connect is OK) In channel, set the 'FTP active mode' (under FTP specfic).

Installation FAQ

I try to install bots at Windows 7/10, but.....

- Probably a rights problem - you'll have to have administrator rights in order to do a proper install.
- Right click the installer program, and choose 'Run as Administrator'.
- sometimes the shortcut is not installed in the menu, and you will have to make this manually.

Does bots have edifact and x12 messages installed out-of-the-box? No. But this can be downloaded on the sourceforge site either as part of a working configuration (plugin) of separate (grammars).

Bots is not working on linux - rights problems. Did you start bots-webserver and/or bots-engine with sufficient rights - e.g. as root. Change the owner/rights of the files in botssys, usersys and config; run bots-webserver/bots-engine without root rights.

Error during windows installation

```
close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr
```

- seems to happen when UAC is turned off.
- Actually bots just seems to be installed OK, and works OK.....
- Fixed this in version 3.2

Start FAQ

When starting bots-webserver the window disappears after a few seconds? Start the bots-webserver from the command line; you will be able to see what goes wrong. (Windows, python 2.7) go to command line and: `c:\python27\python c:\python27\Scripts\bots-webserver.py` For the most common cause for the problem see the next question.

Bots-webserver gives error: IOError: Port 8080 not free on 'x.x.x.x' (or similar).

Another program already uses this 'port'.

Adapt the port bots uses: in configuration file bots/config/bots.ini look for 'port'.

Change port to eg 8090

Start bots-webserver again.

In your browser you will have to indicate another port eg: http://localhost:8090

Can I run multiple instances of bots-engine in parallel?

No, this is not possible.

Instead bots >= 3.0 has better control of running the engine: *jobqueue server*.

Debugging in bots

Basics: use bots-monitor/GUI

- The reports-screen gives an overview of what happened in a run.
- If there are process errors in a run, first view the process errors.
- The incoming-screen gives information about the results of each incoming file.
- For each incoming edi-file: view the detail-screen for information about the processing steps for incoming/outgoing edi file. Usage: in incoming-screen move mouse over star at the start of the line; drop-down box appears; choose 'details'.
- View the outgoing files, including communication errors for outgoing files.

Options to get more information

- Use explicit checks for all gets/puts: parameter 'get_checklevel' in config/bots.ini:
  - '2': all gets/puts are checked with the grammar(s). Do not use in production, bad for performance.
  - '1': sanity checks on get/puts.
  - '0': fastest, might give strange error-messages. But this should have been debugged by you.
- Use 'print' in your mapping script. Output can be viewed on the console/command line. Simple, very efficient.
- Use root.display() to see message content. Not the nicest output, but is definitely what you received or generated.
  - for incoming: at the start of the main function use *inn.root.display()*
  - for outgoing: at the end of the main function use *out.root.display()*
- Logging. Logging files are written to botssys/logging. Set parameters in config/bots.ini to get more debug information.
  - *log_file_level*: set to DEBUG to get extended information.
  - *readrecorddebug*: information about the records that are read and their content.
  - *mappingdebug*: information about the results of get()/put() in the mapping script.
- Logging for communication protocols:
  - Set in config/bots.ini eg ftpdebug, smtpdebug, pop3debug.
  - This debug-information is on the console/command line (but not for SFTP, uses the normal logging).
- Set parameter 'debug' in config/bots.ini; gives information about the place in the code where the error occurred.

Tips

- Always check first if a run has process errors; if so check the process errors.
- The errors bots gives for incoming edi-files are quite accurate.
- Edifact and x12 messages can contain errors. Especially when **found on internet** or taken from documentation/pdf; eg lot of ISA headers for X12 are not OK.
- If you start using python debugger (pdb) you are definitly on the wrong track.
- Be careful with character-set/encoding.
  - Can your editor handle this? Be careful with editing files and saving these again, you might run into issues with character-set/encodings.

  - Are you familiar with the character-set you use? Are you familiar with eg utf-8? If not, please check this first.

Plugins

- A plugin is a file with predefined configuration for bots. It includes routes, channels, grammars, mappings etc.
- Plugins are installed via `bots-monitor-Systasks->Read plugin`.
- In order to run the routes in a plugin you first have to activate the routes.
- There are useful plugins on the bots sourceforge site.
- *Documentation* for these plugins.

Note: grammars for edifact and x12 are NOT plugins, and can not be installed.

For what is a bots plugin useful?

- An easy way to distribute and share edi configurations for bots.
- Learn from existing configurations
- Backup your configuration.
- Share your configuration with others (It would be nice if you want to share your configuration; other can use and/or learn from what you have done.).
- Do another install of the same configuration.
- Transfer your environment from test to production.
- Plugins are used to update to a new version of bots (only between minor versions).

Install a Plugin

- First download the plugin from the bots sourceforge site to your file system.
- Go to `bots-monitor-Systasks->Read plugin`.
- Select the plugin from your system.
- After selecting choose `Read`.

Bots will now report that is has read the plugin.

Note: In order to run the routes in a plugin you first have to activate the routes.

Note: grammars for edifact and x12 are NOT plugins, and can not be installed.

Make a Plugin

It is very easy to generate a plugin from your own configuration:

1. Go to `bots-monitor->Systasks->Make plugin`.
2. Select what your want in the plugin (defaults are for a configuration plugin).
3. Click button `Make plugin`.
4. Give filename/path to save the plugin.

Plugins Explained

- A plugin is just a zip-file, you can view the contents.
- A plugin has to contain a file (at the root) called `botsindex`. This contains the database configuration.
- The plugins on the web-site contain example edi files so you can run a translation and see the results.
- The edi files are in `bots/botssys/infile`; the results will be in `bots/botssys/outfile`.
- When reading a plugin your existing current configuration is backupped (in `bots/botssys`)
- When reading a plugin already existing database-entries, mapping scripts, grammars etc are overwritten.
- In the web server log is logged what is installed (logging is in `bots/botssys/logging`).
- There is no un-install.
- Beware if you make a plugin and share this: do not use e.g. password for an email account.
- Plugins contain source code. This is a potentially security risk. Use only plugins from trusted sources.
- Grammars for edifact and x12 are NOT plugins, and can not be installed.

Plugins at SourceForge

Downloads at the bots sourceforge site

- **my_first_plugin**
  - This plugin is used in the the *Quick Start Guide*.
  - edifact ORDERS to fixed records.
- **edifact2xml_ordersdesadvinvoice**
  - The most used edifact messages in one package.
  - edifact ORDERS to xml
  - ASN/shipping list: edifact2xml_ordersdesadvinvoice to edifact DESADV.
  - Invoice: xml to to edifact INVOIC.
- **x12toxml_supplier_version_850-856-810-997**
  - The most used x12 messages in one package.
  - translate x12 850 -> xml orders.
  - Generates 997's for the received orders.
  - translate xml ASN's -> x12 856; including partner specific translation to 856.

  - translate xml invoices -> x12 810.
- **x12toxml_retailer_version_850-856-810-997**
  - The most used x12 messages in one package.
  - translate xml orders -> x12 850.
  - translate x12 856 -> xml ASN's
  - translate x12 810 -> xml invoices
- **x12toxml_one-on-one_835-837**
  - translate x12 835 > xml and reverse (xml->x12 835)
  - translate x12 837 > xml and reverse (xml->x12 837)
  - one-on-one mapping: structure of xml is similar to structure of x12
  - x12 envelopes are included in mapping
- **demo_composite_route**
  - demonstrates the use of composite routes.
  - 2 different sources are used for incoming edifact files
  - convert edifact orders to fixed format.
  - also converts these orders to print format (html).
  - convert edifact SLSRPT to csv format.
  - all outgoing messages go to different destination using filtered outchannels
- **edifact2fixed_orders-desadv-invoic**
  - The most used edifact messages in one package.
  - Suited for Dutch non-food retailers; probably for a lot of other retailers as well.
  - edifact ORDERS to fixed file format.
  - generate a edifact APERAK message (if requested in the order).
  - fixed format ASN/shipping list to a edifact DESADV.
  - fixed format invoice to edifact INVOIC.
- **demo_working_with_partners demonstrates the use of partner dependent functionalities in bots:**
  - parter dependent translations.
  - user of 'default translation' and imports.
  - partner dependent syntax.
  - use of partner-lookup in the database.
  - use of partner groups.
  - different destinations/outchannels for partners.
  - advanced usage of ISA qualifiers/partnerID. Your partnerID is used to lookup the ediID and ISA qualifier in database (and of course vice versa).
- **demo_sap_idoc_orders_WPPLU**
  - edifact D96A ORDERS to idoc ORDERS01.

  - edifact D96A PRICAT to idoc WP_PLU02.
  - idoc ORDERS01 to edifact D96A ORDERS.
  - idoc WP_PLU02 to edifact D96A PRICAT.
- **demo_communicationscript**
  - Demonstrates user communication scripting.
  - Incoming edi-messages can be passed one by one or as one batch.
- **demo_databasecommunication**
  - Demonstrates database communication scripts; both reading and writing (in separate routes).
  - A test database comes with the plugin.
- **demo_mdn_confirmations**
  - Demo configuration for MDN (email confirmations)
  - See also documentation about confirmations.
- **demo_one-on-one_edifactorder2xml**
  - Translates a edifact ORDERS D96A message to xml-message using edifact tags as xml-elements and vice versa.
  - For those who like processing xml instead of edifact.
  - It is quite easy to add other edifact messages types for similar translations.
- **edifact2json_invoic**
  - Translates edifact D96A invoice to JSON en vice versa.
- **edifact_ordertoprint**
  - Translates a edifact ORDERS D96A message into a readable format (HTML).
  - Of course you can print the order, so it is like a (edi)fax.
- **edifact_orders2csv_and_vv**
  - Translates edifact D96A orders to csv en vice versa.
  - There a 2 variants: csv with or without record tags (csv from eg excel is often without records tags).
- **edifact_pricat-slsrpt**
  - Csv to edifact PRICAT.
  - Edifact SLSRPT to csv.
- **demo_preprocessing**
  - This plugin demonstrate preprocessin: the incoming edifact-files start with an number of '#' and '@'. These characters are removed by preprocessing the files.

Contributed plugins at SourceForge

- **alto_seperate_headers_details**
  - input: 2 csv files, one with headers, one with details lines.
  - this is processed into one idoc (so the headers and details are merged).

  - Same technique is also usable for fixed format.

- **x12_837_4010_to_x12_837_5010**
  - converts (physician) insurance claims (x12 837) in the version 4010 to the new, upcoming, 5010 version.
  - The mapping file is rudimentary, but I believe the conversion is OK.
  - I found that removing the one REF file creates a version 5010 file that is accepted and processed properly by Anvicare, the clearing house for my commercial claims.
  - I have included anonymized input edi transactions for Medicare, Blue Shield and commercial insurers.
  - My approach is to try the translation as is, and to make corrections in the mapping file only if I get errors from the clearing house.
  - Medicare and Blue Shield provide comprehensive error checking function. However, they do not yet accept 5010 transactions, even for testing purposes.
  - The clearing house accepts 5010 transactions, and they work.
- **x12_fixed_2_810**
  - converts fixed inhouse to x12 810 including calculation of invoice totals etc and partner specific seperators.

Overviews

Purpose of the chapter of this wiki is to give some global overview for bots open source edi translator.

An overview of features is on sourceforge.

Performance

- Most edi files are just a few kilobyes. An edi file of 5Mb is very large (edifact, x12). If you encounter larger ones: please let me know.
- AFAIK there are no issues with performance. If you run into this: please inform me eg via mailing list.

Get More Performance

- (Bots >= 2.2.0) Cdecimals (Extern library) speeds up bots. This library will be included in python 3.3, but can be installed in earlier python versions.
- (bots >= 3.1) Do not use `get_checklevel=2` in `config/bots.ini`. This does extended checking of mpath's, use this during development only.
- (bots >= 3.0) Schedule bots-engine via the job queue server.
- For xml: check if the c-version of python's elementtree is installed and used.
- Enough memory is important for performance: disk-memory swapping is slow. Actual memory usage depends on size of edi-files.
- Check mappings for slow/inefficient algorithms.
- Bots works with pypy, see below on this page.
- Use SSD for faster reading/writing. In `config/bots.ini` the `botssys-directory` can be set, in `config/settings.py` the place of the SQLite-database.

Strategy for bigger edi volumes

- Best strategy is to *schedule* bots-engine more often.
- (bots >= 3.0) Schedule bots-engine via the *jobqueue server*.
- Routes can be *scheduled independently*.
- Set-up good scheduling, keeping volumes in mind.
- EDI in the real world has often large peaks.
- Dome edi transactions are time critical (eg orders), others not so much (eg invoices)
- Check where the large volumes are (size and number of edi-transactions)
- Look at the sending pattern of your customers. Often edi is send in night jobs, so you might receive lot of volume early in the morning.
- Check where you send large volumes. Send this at a time that does not interfere with other flows.
- **Incoming volumes can be limited per run. This way the time bots-engine runs is predictable.**
  - The max time a channel fetches incoming files is a parameter for each channel.
  - This is dependent upon the communication type used; eg file system I/O is much faster than SFTP.
  - Files "left behind" will be fetched on subsequent runs.
- (Bots >= 3.0) Limit for max file-size (set in bots.ini). If an incoming file is larger, bots will give error. This is to prevent **accidents**.

Performance/throughput testing

1. Tests are done using file system I/O (no testing of communication performance).
2. Tests done in one run of bots-engine.
3. Test system: Intel Q9400 2.66GHz; 4Gb memory; ubuntu 10.04(lucid); python 2.7; default SQLite database.
4. Please note that these tests are **artificial**: if you have such high volumes and big files look at good scheduling.
5. Tests are with edifact; x12 performance is the same.

Description	File Count	Total Size	Message Count	Time(bots2.0)	Speed(bots2.0)	Time(bots3.0)	Speed(bots3.0)	Time(bots3.2)	Speed(bots3.2)
01 edifact2fixed	32	305Mb	32	1:01:39	82 kb/s	0:50:57	100 kb/s	0:44:01	115 kb/s
02 edifact2fixed	116	300Mb	116	1:14:18	68 kb/s	0:36:20	137 kb/s	0:37:25	133 kb/s
03 edifact2fixed	94048	295Mb	141072	0:47:21	104 kb/s	0:39:54	125 kb/s	0:42:30	115 kb/s
04 fixed2edifact	14244	300Mb	78342	1:04:11	78 kb/s	0:33:21	150 kb/s	0:32:40	153 kb/s
05 xml2edifact	17424	300Mb	17424	0:41:24	121 kb/s	0:35:48	139 kb/s	0:35:20	141 kb/s
06 edifact2xml(1to1)	14609	300Mb	74919	1:23:03	60 kb/s	0:58:19	85 kb/s	0:44:38	112 kb/s

Conclusions of performance measurements:

1. Memory usage is stable (no leakage).
2. Memory usage is directly related to the size of the edi-files. In test 01 (edifact files of 9.5Mb) bots-engine uses 1.5Gb memory.

3. Tested with edifact files of 120Mb; memory usage is stable at 4.5Gb.
4. Performance is reasonably independent from the size of edi-files/messages.
5. An edifact file of 9.5Mb takes about 85sec to be processed.
6. For outgoing edi files: writing to one file or multiple file does not significantly affect performance.

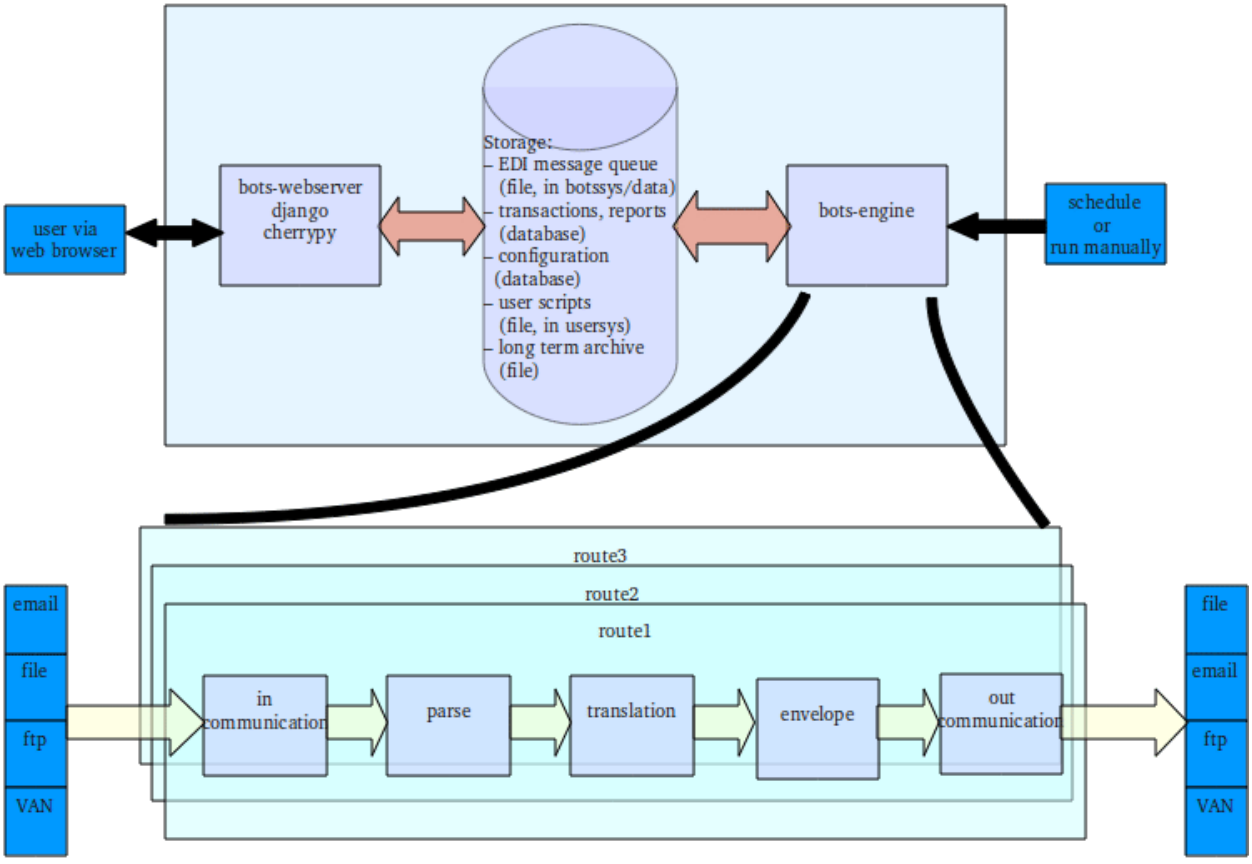
Testing with pypy

pypy s a python implementation that is faster by using a JIT. Results of the first tests with pypy (beta-versions of pypy 2.0):

- Bots works with pypy.
- Comparing some stress tests: much faster, 2-3 times faster.
- Did not run all test-sets. Probably I will do that with definitive version of pypy 2.0 and/or release of bots 3.1.
- **Problem might be that not all libraries/dependencies work with pypy.**
  - SQLite3 database connector: OK
  - MySQL database connector: version 1.2.5 does. Note that bots 3.1.0 gave am error with this version, a patch was easy.
  - paramiko (for SFTP/SSH): no, dependency pycrypto is not supported.

This looks like a very interesting development.

Technical Overview



Directories and Files

Where can I find bots on my system?

- **Windows (depends upon python version used):**
  - eg. `C:\python27\Lib\site-packages\bots`
- **Linux, Unix:**
  - `/usr/lib/python2.7/site-packages/bots`
  - `/usr/local/lib/python2.7/dist-packages/bots (Debian, Ubuntu)`

Note: In the main screen (home) of bots-monitor (GUI) you can see where the bots-directories are.

Where are the programs/executable scripts?

- **Where installed**
  - Windows (depends upon python version used): `C:\python27\Scripts`
  - Linux, Unix eg in `/usr/bin/` or `/usr/local/bin/`
- **Overview of executables:**
  - `bots-webserver.py` - serves the html-pages for the bots-monitor.
  - `bots-engine.py` - does the actual edi communication and translation.
  - `bots-jobqueueserver.py` - the jobqueue-server maintains a queue with (job-engine) jobs and fires these asap.
  - `bots-job2queue.py` - to send jobs to the jobqueueserver
  - `bots-dirmonitor.py` - monitors/watches directoriess; if a file is placed in a directory, bots-engine stats (via jobqueueserver)
  - `bots-grammarcheck.py` - utility to check a grammar.
  - `bots-xml2botsgrammar.py` - utility to generate an xml-grammar from an xml file.
  - `bots-updatedb.py` - for version migrations: updates the database to new definition/schema
  - `bots-plugoutindex.py` - for version control systems: generate a file with the configuration (routes, channels, etc). (configuration as in the database).

Note: For the usage instructions of the executables use `help`, eg:

```
bots-engine.py --help
```

Where are the edi files?

- **Incoming and outgoing**
  - Set this per channel, using 'path' and 'filename'.
  - Relative path is relative to bots directory, absolute paths are OK.
  - In the plugins the edi-files are in `bots/botssys/infile`
- **The archive/backup.**
  - each communication channel of bots CAN archive the incoming or outgoing edi messages.

  - Set this per channel, using 'archivepath'.
  - The place of the archive can be 'anywhere'. We mostly use `botssys/archive`.
- **Relative path is relative to bots directory, absolute paths are OK.**
  - Internal storage of edi data.
  - in subdirectories of `bots/botssys/data`.
  - this is where the edi files are fetched from by the bots-monitor (the 'filer' module).
- **Cleanup of old files**
  - bots manages cleanup of archive and internal storage, based on configuration settings for number of days to keep.
  - Incoming files are (usually, unless testing) removed by the channel setting "remove".
  - Outgoing files are never removed by bots

The directory structure within bots directory

- **`bots`: source code (*.py, *.pyc, *.pyo)**
  - **`bots/botssys`: data file, database, etc**
    - `bots/botssys`: when installing a plugin bots makes a backup of configuration here.
    - `bots/botssys/data`: internal storage of edi data.
    - `bots/botssys/sqlitedb`: database file(s) of SQLite.
    - `bots/botssys/logging`: log file(s) for each run of bots-engine.
    - `bots/botssys/infile`: plugins place here example edi data
    - `bots/botssys/outfile`: plugins place here translated edi data
  - `bots/config`: configuration files. See also multiple environments.
  - `bots/install`: contains an empty sqlite database and default configuration files.
  - `bots/installwin`: (windows) python libraries used during installation.
  - `bots/locale`: translation/internationalisation files for use of another language.
  - `bots/media`: static data for bots-webserver (CSS, html, images, JavaScript)
  - `bots/sql`: sql files for initialising a new database.
  - `bots/templates`: templates for bots-webserver
  - `bots/templatetags`: custom template-tags for use in django.
  - **`bots/usersys`: user scripts.**
    - `bots/usersys/charsets`: e.g. edifact uses its own character-sets.
    - `bots/usersys/communicationscripts`: user scripts for communication (inchannel, outchannel).
    - `bots/usersys/envelopescripts`: user scripts for enveloping.
    - `bots/usersys/grammars`: grammars of edi-messages; directories per editype (x12, edifact, tradacoms, etc).
    - `bots/usersys/mappings`: mappings scripts; directories per editype (incoming).

* `bots/usersys/partners`: partner specific syntax for outgoing messages; directories per editype.
* `bots/usersys/routescripts`: user scripts for running (part of) route.
* `bots/usersys/codeconversions`: for conversions of codes (deprecated).
* `bots/usersys/dbconnectors`: user scripts for communication from/to database (old database connector, deprecated).

Configuration Files

- Bots has two configuration files.
- Mostly the defaults will be OK to get started, but you may need to customize these to your own needs.
- Settings are documented in the files.
- Open these files in your *text editor*.
- The configuration files are located in the `bots/config` directory.
- The original default versions can be found in the `bots/install` directory.

bots.ini

- How long to keep edi files and their registration
- Some GUI customisation settings
- Timeouts and time limits
- Logging options
- Debugging options
- Webserver settings (eg port)
- Directory settings

settings.py

- Mail server settings for error reports
- Database settings (eg. to use another database)
- Security / auto-logout
- Localization (time zone)

Changes, migration

- Details about changes in new versions
- How to migrate/upgrade to newer bots versions.
- In general, try to upgrade to new bots version, especially if the version is upward compatible.
- Most of the time migration is easy.
- Grammars and mappings have always worked for different versions.

Bots 3.2.0

- Bots 3.2.0rc was released 2014-03-22.
- Bots 3.2.0rc2 was released 2014-05-27.
- Bots 3.2.0 was released 2014-09-02.

Migration notes

- bots 3.2.0 is suited for django 1.4, 1.5, 1.6 and 1.7. Support for Django 1.3 is dropped.
- In existing settings.py django requires parameter 'ALLOWED_HOSTS'. Add this as eg:

```
ALLOWED_HOSTS = ['*']
```

- For SQLite no database migration is needed. For MySQL and PostgreSQL: in table unique, field 'domein' should be changed to 70 positions (was: 35) if using option to have in-communication connect failures reported after xx times (field 'Max failures' in channels).
- Python 2.5 is not supported anymore. use 2.6, preferably 2.7.
- Bots does not support python > 3 (yet). Reason is that python MySQL connector is not suited for python 3 (yet).

Changes

Highlights

1. Added http(s) communication to communication methods (using 'requests' library).
2. Better reporting of partnerID for incoming files with errors.
3. File rename after outcommunication using tmp filenames to avoid 'reading before writing'.
4. Updated edifax (print to html); is much easier to use now (edifax plugin will be updated).

Interface (GUI)

1. In run-screen: show used command for bots-engine, show if acceptance test.
2. Search by filename for incoming and outgoing files.
3. *strange characters* like ëë in routeID, channelID, etc will work now.
4. Option to have automaticretrycommunication in menu.
5. Option to add only routes that are not in default run to menu.
6. Added bulk delete for persists data.
7. Use 'rereceive' and 'resend' in screens (instead of 'retransmit').
8. Simpler making bots suited for touchscreen.
9. Filter routes for (not) in default run.
10. Improved edifact indenting in file viewer.
11. Div small layout improvements in GUI.

Smaller changes

1. unicode handling is improved. This is about use of unicode in route-names, grammars, correct errors, etc

2. set maxdaysarchive per channel.
3. email validation in django was too strict for certain email addresses.
4. check CC-address when validating incoming email-addresses.
5. improve reporting on in-communication failures (give process-error after x time failure).
6. suited for django 1.4, 1.5, 1.6and 1.7. Django 1.3 support is dropped.
7. add incoming filename in email-report.
8. option to use user script entries at start/end of a run and command.
9. added option 'parse and passthrough' for routes. Useful for eg generating 997/CONTROL.
10. added explicit enveloping indication (this indication can be set in mapping).
11. making of plugin could take take a long time (better performance for large plugins).
12. improve processing of repeat separator in ISA version above 00403.
13. xml2botsgrammar: order of elements is preserved now; all xml entities as records.
14. xml2botsgrammar: added option to have all xml entities as 'record'.
15. added option to have function 'transform.partnerlookup' return 'None' if not found.
16. improve xmlrpc communication.
17. for persist: change timestamp on update.
18. used new definitions for UNOA and UNOB. Definitions are clearer and easier to change, no change in functionality.
19. better performance for one-on-one translations.
20. bots-engine uses less memory.
21. better reporting for import errors.

Bug fixes

1. preprocessing not working for rereceive.
2. fixed bug in windows installer (occurred when when UAC is disabled).
3. routeid did not always show up correct in error.
4. rights problem: viewer can delete incoming files.
5. show filesize for passthrough files.
6. MySQLdb version 1.2.5 gave error.
7. change password for non-staff user did not work.
8. found query's in GUI not using index (better performance).
9. error could not be displayed: <unprintable MessageError? object>.
10. multiple email-addresses were not used in sending emails.
11. transform.concat erroneously added spaces between elements.
12. callable in QUERIES should have 'node' as parameter.
13. when received x12 message can not be parsed correctly, GS08 might not be found.
14. routeid did not show up correct in some errors.

15. incoming tab-delimited file not parsed correct (for nobotsID & first field Conditional).
16. via incoming/outgoing screen: button 'confirm (same selection)' did not work.
17. in forms.py: reference was an..35, should be an..70

Bots 3.1.0

- Bots 3.1.0RC was released 2013-07-17.
- No database migration is needed.
- Some small changes in x12 grammar might be needed, see details below.

Migration notes (from 3.0.0)

Editype x12: in ISA definition 'ISA11' has to be conditional

- Bots 3.1.0 supports the repeating character (for ISA-versions >= 00403).
- **If you use this, ISA11 has to be conditional in definition; file `usersys/grammars/x12/envelope.py`, was:**
  `['ISA11','M',(1,1),'AN'],`
- **becomes in 3.1.0:** `['ISA11','C',(1,1),'AN'],`
- This change also works for older ISA-version. Advised is to use this.

Editype x12, edifact, tradacoms: different handling of syntax parameters

- **Bots handles the syntax parameters differently. In bots 3.0 and earlier this was somewhat 'fuzzy'. This works now li**
  - default syntax parameters are overruled by envelope parameters,
  - are overruled by message parameters,
  - are overruled by frompartner parameters,
  - are overruled by topartner parameters.
- **Most common problem will be for x12: message grammars had by default in syntax parameters:**
  `'version' : '00403',`
- Formerly this was not used; now it is used.
- This might lead to sending another version of ISA-envelope, this is probably not what you want.
- Solution: delete (or uncomment) the `version` syntax parameter for message grammar.

Note: The ISA-version you send now is probably in `usersys/grammars/x12/x12.py`

Editype fixed and idoc: 'startrecordID' and 'endrecordID' not longer used

- Bots calculates this automatically now.
- Bots also checks for all used records if this is used the same over all records.
- This might lead to errors.
- **Solution**: BOTSID should have correct length in all records and be at correct position.

Envelope scripts

- **was:** `self._openoutenvelope(self.ta_info['editype'],self.ta_info['envelope'])`
- **becomes in 3.1.0:** `self._openoutenvelope()`

Route scripting

Function transform.translate is changed:parameters startstatus,endstatus,idroute,rootidta have to be explicitly indicated (no more defaults).

Changes

Interface (GUI)

1. **Improved and simplified many screens. Eg only filename is displayed (takes less space), a pop-up show full patl**
   (a) incoming
   (b) outgoing
   (c) detail
   (d) document; split up to incoming and outgoing screens;
   (e) confirm
2. Show in configuration screen if there are routescripts, communicationscripts, mappingscripts, grammars. Routescripts etc can be viewed.
3. Partners and partnergroups are split up (via menu and screens).
4. Added a cancel button in configuration editing.
5. Added a choice list for routes in Configuration-Confirm.
6. Display edifact/x12: display per segment for better readability.
7. Show correct number of messages for resends.
8. Email error report is extended with information about errors.
9. Improved view/edit counters screen.
10. In errors the correct name of eg grammars is shown. This was confusing (using sometimes '.' instead of '' etc).

Highlights

1. Extra debug option: check all get/getloop if OK with grammars.
2. Support for repeating elements is added (x12/edifact).
3. Simplified logic of syntax reading: default syntax is overridden by envelope syntax is overridden by message sytax is overridden by partner-syntax.
4. Added: fixed records can now be 'nobotsid': one record type, split to messages via field.
5. Generating 997's can be done/manipulated via user script .

Smaller changes

1. Skip empty json elements in incoming files.
2. StartrecordID and endrecordID are not needed anymore for in grammars for fixed message/idoc, bots calculates this now.

3. Small improvements and bug fixes in XML reading/writing.
4. QUERIES now also support callables.
5. Xml2botsgrammar: sort fields in recorddefs, use empty elements in grammar.
6. If 'alt' transaltion is not found, use default translation.
7. More consistent handling of exceptions and logging (coding only).
8. Fixed problems starting bots-engine from webserver in less common situations.
9. Get correct incoming filename for re-receive.
10. Removed code for old database connector and code-conversion via file.
11. Automaticretry: first run only initialization (to avoid sending moch older files).
12. Explicitly set for outgoing file: no automatic retry.
13. Plugins: for different environments, path and testpath in channels are also relocated .
14. Plugins: handling of unicode-characters is now correct.
15. Add mapping function: getdecimal(). Returns a python decimal; if not found or non-valid input: returns decimal 0.
16. For csv and fixed with 'noBOTSID': nextmessageblock can check for multiple fields, eg: `nextmessageblock = ([{'BOTSID':'lin','field1':None}, {'BOTSID':'lin','field2':None}])`
17. When deleting configuration items via 'bulk delete': make a backup plugin first.

Bug fixes

1. There was a missing import in `xml2botsgrammar`
2. Logging of mapping debug did not work in 3.0
3. Correct handling of resends/rereceives for already resend/received files
4. Fixed bug in automaticretrycommunication
5. Confirmation can now be asked via channel-rule.
6. if multiple commands in run: reports etc are based on timestamp. This messed up the relation between runs and eg incoming files.

Bots 3.0.0

- Bots 3.0.0 was released 2013-02-04
- Bigger changes, and a database migration is needed.

Migration Notes

Introduction

Version 3.0 is a bigger update for bots, view all changes in next section. Overview:

1. Django 1.1 and 1.2 are not supported anymore. Supported are django 1.3 and 1.4.
2. `Settings.py` is changed. Advised: use the new settings.py, and do your customization in the new setting.py (eg for error reports, maybe database and timezone).

3. The database has changed. A script is included to change the database. I had no database issues while testing this migration.
4. lots of changes in bots.ini. The 'old' bots.ini is OK, but it is advised to use new bots.ini, and do your customizations there.
5. Excel input: does work; but is now an incoming messagetype and not via 'preprocessing'.
6. Most user scripts will work; many user scripts are not needed anymore because the functionality is provided by bots now.
7. **Some functions that may be used in user scripting have changed:**
    - `botslib.change()` ->`botslib.changeq()`. This function is used to in processing incoming 997's and CONTRL
    - botsengine routescripts: for 'new' runs in routescript botsengine.py called function postnewrun(routestorun) -> postnew(routestorun)
    - `communication.run(idchannel,idroute)` -> `communication.run(idchannel,command,idroute)`. *Command* is one of: 'new','automaticretrycommunication','resend','rereceive'.
    - `transform.run(idchannel,idroute)` -> `transform.run(idchannel,command,idroute)`. *Command* is one of: 'new','automaticretrycommunication','resend','rereceive'.

Note: My experiences: after changing settings.py and database migration, all works (except for and issues mentioned above).

Summary of procedure

1. make a backup!
2. rename existing installation.
3. do a fresh install.
4. copy old data to new installation.
5. change settings
6. update the database.

Note:

- It is critical to change the settings before updating the database. Bots finds the right database via the settings.
- If you use MySQL or PostGreSQL: same procedure. The bots-updatedb script also updates MySQL or PostGreSQL.
- Tested this for migration from bots2.2.1 -> 3.0.0, but works for all bots2.* installations.

Windows procedure

1. make a backup!
2. **rename existing installation**
    - existing bots-installation is in `C:\Python27\Lib\site-packages`

- renamed bots directory to bots221
- also renamed existing directories for cherrypy, django, genshi.

3. do a fresh install of bots3.0.0 installer (bots-3.0.0.win32.exe)
4. **copy old data to new installation.**
   - in `C:\Python27\Lib\site-packages` new directories have been installed (bots, django, cherrypy, genshi)
   - copy directories botssys and usersys from bots221-directory to bots directories. Everything can be overwritten.
5. **change settings**
   - use new `config/bots.ini`, adapt for your own values.
   - use new `config/settings.py`, adapt for your own values. Especially the database settings are important; the format is slightly different (but similar enough to give no problem); critical is using the 'ENGINE'-value of the new settings.py.
6. **update the database.**
   - use command-prompt/dos-box
   - goto directory `C:\Python27\Scripts`
   - command-line: `C:\Python27\python bots-updatedb.py`
   - should report that database is successful changed.

Note: If you use a 64-bits version of windows another option is to use the 64-bits versions of python and bots.

Linux procedure

1. make a backup!
2. **rename existing installation**
   - existing bots-installation is in `/usr/local/lib/python2.7/dist-packages`
   - renamed bots directory to bots221
   - for libraries: check you use at least django 1.3
3. do a fresh install: see *installation procedure*
4. **copy old data to new installation.**
   - in `/usr/local/lib/python2.7/dist-packages` new bots-directory is installed.
   - copy directories botssys and usersys from bots221 directory to bots directories. Everything can be overwritten.
   - mind your rights.
5. **change settings**
   - use new `config/bots.ini`, adapt for your own values.
   - use new `config/settings.py`, adapt for your own values. Especially the database settings are important; the format is slightly different (but similar enough to give no problem); critical is using the 'ENGINE'-value of the new settings.py.

6. **update the database.**
    - command-line: `bots-updatedb.py`
    - should report that database is successful changed.

Changes

Changes in database format since version 3.0.0rc

Alas; the database as used in the 3.0.0rc version has changed. Changed is:

- table channel: field 'testpath' is added
- table report: field 'acceptance' is added
- tabel ccode: field 'rightcode'->70pos
- tabel ccode: field 'attr1'->70pos

Wrapping of user script functionality into GUI

1. For outgoing filenames: can include partnerID, messagetype, botskey, data/time, etc in channel.
2. Pass-through is option in route.
3. Zip and unzip files as option in channel.
4. SSL keyfile and certificate as options in channel.
5. Excel as incoming messagetype (instead of via preprocessing).
6. Option in channel to indicate edi file in email should be in body.
7. Add: communication-out type 'trash' to discard of edi files.

Improved 'run' options

1. Run options in GUI are simpler: new, rereceive, resend. (Automatic recommunication is possible via bots-engine).
2. Communication errors are visible in outgoing-screen and can be 'resend' manually.
3. Add 'resend all' and 'rereceive all' in incoming/outgoing screen; espc. useful in combination with selections.
4. Indicate in screens a file has been resend; Keep track of number of resends of an outgoing file. Filer works 'over' resends now.
5. Dropped 'retry' option. This was not useful and confusing. Use 'rereceive'.
6. Dropped 'retry communication'. Use 'resend', easier and more consistent.
7. Add: use 'channel' in selects (eg useful for resend selects).

Configuration change management :doc:'see wiki <../deployment/change-management>'

1. **Use tools to push changes test-> production**
    (a) Via GUI: write database configuration to usersys in order to compare environments.
    (b) Via GUI: read the changed database configuration in usersys after pushing changes
2. **Isolated acceptation tests: run an acceptance test set without changes.**
    (a) option in bots.ini to active 'acceptance test'
    (b) bots prevents communication to 'outside

(c) use seperate path to read/write ('testpath')

(d) do not change counters etc

(e) results of acceptance tests can be deleted/removed

(f) etc

Many improvements in GUI, eg:

1. 'Detail' screen has a better layout.
2. Improve: show name partner in selectlist (in configuration).
3. Improve: show channel-type in selectlist and routes (in configuration). I find this very convenient.
4. Improve: better sorting in configuration for in django 1.4, eg for routes, codelists, trasnlations.
5. References in a configuration are better guarded. Eg: when deleting partner that is used in partner-specific translation: user is warned, has to confirm.
6. Improve: removed charset from channel. This was not needed.
7. Improve: selections in screens not only for partners but also for partner groups
8. Fix: download an edi-file via filer could give not-correct file.
9. Add: keep track of file-size of incoming/outgoing files.

Extended partner functionality

1. Add: function 'partnerlookup' for use in mapping scripts to look-up/convert partner related functions.
2. Add: extra partner fields: address, user-defined fields.
3. Add: send a message to any bots partner from mapping script (use partners as the email address book).
4. Improve: multiple email addresses in cc field.

Confirmations

1. Fix: bug in confirmation logic for frompartner/topartner (I had them reversed...).
2. Better CONTRL-message.
3. Option to run user exit either to generate CONTRL message or change the generated CONTRL message.
4. messagetype (for incoming) is now similar to other uses of message-type; edi-type is removed (is not needed)
5. Fix: better indexing for reference/botskey. This improves performance for confirmations.

Plugins

1. Always make configuration backup when reading new plugin.
2. Dropped the date-indication for files that are overwritten.
3. Index file in plugin always starts with type of plugin and layout is sorted/predictable.
4. Fix for performance problem when generating plugin for big plugins.
5. Fixed bug for relations in database. Bug only occurs when reading 'hand-changed'-plugins.

Better handling of 'database is locked'/crash recovery

1. First of all: use the jobqueue-server for more complicated scheduling (prevents running multiple engine)
2. Different way of detecting another instance of engine is running via locking of port.
3. If bots-engine finds no other engine is running, but the database is locked this indicates the previous run was ended unexpectedly (eg computer crash).
4. In this case bots will do an automatic 'crash recovery'. A warning is still given in logs or via email. Only ONE email is send.

Technical

1. An 64-bits installer for windows is available.
2. Runs with django 1.4 now ; dropped support for django 1.1, 1.2
3. Database has changed. A script is added to upgrade the database.

Other changes

1. Add: startup script for bots-webserver using apache. Multiple bots-environments can run over one apache server.
2. Add: user exit for cleanup.
3. Improve: use external file name in archive.
4. Add: archive as zip-files.
5. Fix: port was not used in initializing PostgreSQL.
6. Less statuses in processing (simpler, faster).
7. Improve: use unlimited text fields in database for errortext and persist.
8. Improve: if message in incoming edi file has error in mapping script and/or writing outgoing file: process rest of edi file (there is a compatibility option in bots.ini).
9. Reworked errors for edi files (parsing, generating). All errors have numbers now (for referencing).
10. Always read incoming files sorted by name. Reason: predictability. Formerly the read order was not predicable.
11. Fix: better handling of import errors for user scripts. This could lead to confusing errors/situations.
12. Add: access to whole envelope in mapping.
13. Fix: nr of messages was not used correctly when writing multiple UNH in one mapping script.
14. Fix: bug for numerical fields with more than 4 decimal positions.
15. Add: counter(s) per bots-run in mapping script; eg useful for UNH/ST counter per interchange.
16. Fix: enveloping for edifact with keca-character-set.
17. Add: concatenate function for usage in mapping scripts.
18. Improve: updated grammar-handling to allow for UNS-UNS construction.
19. Improve: set explicit namespace when generating xml.
20. Add: determine translation via user scripting.
21. Add: user scripts can easy detect what type of run is done via routedict['command']
22. Dropped: intercommit connector.

Bots 2.2.0

Migration notes

Procedure

- **Update plugin:**
    1. Get the plugin at sourceforge.
    2. Mind there are 2 version of the plugin, depending upon the version of django you use.
    3. Read like a normal plugin (`bots-monitor->systasks->read plugin`).
    4. Bots-monitor will give an error...nasty but upgrade is done.
    5. Stop the web-server.
    6. Edit `bots/config/settings.py` (See below)
    7. Restart bots-webserver.
- **For windows:**
    1. Get the installer at sourceforge.
    2. **Install new version.**
        - Django is upgraded to 1.3.1
        - Database and mappings are not changed
        - Configuration files have not been changed
    3. Edit `bots/config/settings.py` (See below)

For all updates: `settings.py` HAS to be changed. One line has to be added (at end of file):

```
TEMPLATE_CONTEXT_PROCESSORS = (
    "django.core.context_processors.auth",
    "django.core.context_processors.debug",
    "django.core.context_processors.i18n",
    "django.core.context_processors.media",
    "django.core.context_processors.request",
    "bots.bots_context.set_context",    #THIS LINE IS ADDED
)
```

Compatibility

Version 2.2.0 is upward compatible with previous versions in 2.*.*-series:

- no data migration needed
- grammars, translations etc mostly will work as before

Compatibility problems

1. when upgrading from 2.0.* see *migration to 2.1.0*
2. as mentioned above: new line has to be added in `settings.py` in `TEMPLATE_CONTEXT_PROCESSORS`

Deprecated

1. django 1.1.*; use django > 1.2.0, see for *instructions*

2. editype: database (=database connector with SQLalchemy). Use editype db; if wished you can use SQLalchemy in this new database connector.
3. code conversion via file (in bots/usersys/codeconversions). Use codeconversion via ccode table: better, faster, more flexible.
4. editype template (with library 'kid'); use editype template-html (with library Genshi) instead. Genshi is quite simular to kid, see for *instructions*
5. communication via intercommit (type intercommit). If needed, a plugin can be provided.

Changes

Changes in bots.ini

You can use your old bots.ini with no problem, reasonable defaults have been used. New options added in 2.2.0:

```
[webserver]

#settings for logging of bots-webserver
#console logging on (True) or off (False); default is True.
webserver_log_console = True
#webserver_log_console_level: level for logging to console/screen. Values:␣
↪DEBUG,INFO,STARTINFO,WARNING,ERROR or CRITICAL. Default: STARTINFO
#actually useful: WARNING: only start-up text; info gives more info
webserver_log_console_level = STARTINFO

# to customise name of botslogo html file (default: bots/botslogo.html)
botslogo = bots/botslogo.html
# text displayed on right of bots logo. Useful to indicate different␣
↪environments: TEST, PRODUCTION. Default: no text
environment_text =

#when True, the run menu contains entries to run each route indivudually.␣
↪Default: False
menu_all_routes = False

[custommenus]
#it is possible to add a custom menu to the default bots menu. Features
#1. the menuname to appear on the menu bar in bots monitor; Default: Custom.␣
↪Eg:
#menuname = MyMenu
#2. Entries ins the custom menu: all "name: value" entries in this section␣
↪will be added to the custom menu in bots monitor. Eg:
#Incoming = /incoming/?all
#3. Menu divider lines can be added with special value "---". Eg:
#divider1 = ---
# note: sequence of entries is preserved, but case of menu entry is not;␣
↪title case will be applied
```

Bots 2.1.0

- This version is the first version in one-and-a-half year.
- Bots 2.0.2 proved to be quite stable.
- This version adds a lot of new functionality; quite some bugs have been fixed.

- This new verson is very upward compatible with version 2.0.2

Migration Notes

Procedure

- Get the plugin at sourceforge.
- Mind there are 2 version of the plugin, depending upon the version of django you use.
- Read like a normal plugin (bots-monitor->systasks->read plugin).
- Stop the web-server.
- Restart bots-webserver.

Compatibility

Version 2.1.0 is upward compatible with previous versions in 2.*.*-series:

- no data migration needed
- grammars, translations etc mostly will work as before

Compatibility notes

After upgrading, some (eg. older edifact) grammars can give errors. This is due to stricter checking of grammars. The records in a grammar are now checked for unique field-names: the same field name is not allowed in a record. This was never OK, but was not checked). Typical error:

```
GrammarError: Grammar "...usersys/grammars/edifact.ORDERSD96AUNEAN008",␣
↪record "FII": field "C078.3192" appears twice. Field names should be␣
↪unique within a record.
```

The culprit is the file D96Arecords (or similar), the FII segment has an error in it.

- Solution 1: adapt grammar manually; change FII segment:

```
['C078.3192','C',35,'A'],
['C078.3192','C',35,'A'],
```

to

```
['C078.3192','C',35,'A'],
['C078.3192#2','C',35,'A'],
```

- Solution 2: use plugin `update_edifact_recorddefs.zip` (same directory as update-plugins. this plugin only contains edifact records for D93A and D96A).

Changes

Detailed changes in version 2.1.0 are here.

Changes in bots.ini

You can use your old bots.ini with no problem, reasonable defaults have been used. Following are the new options added.

```
[settings]

#adminlimit: number of lines displayed on one screen for configuration items;
↪ default is value of 'limit'
#adminlimit = 30

#for incoming channels: limit the time in-communication is done (in seconds).
↪ Default is 60. This is the global parameter, can also be limited per␣
↪channel (in GUI)
maxsecondsperchannel = 60

#sendreportifprocesserror : do not send a report by mail if only process␣
↪errors occurred. useful if outcommunication often gives error. default=␣
↪True (send if there is a process error)
sendreportifprocesserror = True

#imap4debug: print detailed information about imap4 session(s). Default 0␣
↪(no debug) (can use 0,1,2,3,4,5)
imap4debug=0

[webserver]

#the server_name. Used to distinguish different bots-environments. defaults:␣
↪bots-webserver
name = bots-webserver

#in order to use ssl/https:
#    - indicate here the file for the ssl_certificate and ssl_private_key.␣
↪(both can be in the same file)
#    - uncomment the lines
#(and of course you will have to make the certificate and private key␣
↪yourself)
#self-signed certificates are allowed.
#ssl_certificate = /path/to/filename
#ssl_private_key = /path/to/filename
```

Migrate Kid to Genshi

If you use bots edi type template then you are using kid for those templates. From version 2.1.0 onwards, using Genshi is also supported, using edi type `template-html`. In a future bots version, kid will be removed as it is no longer being developed.

You may have copied templates from a plugin, or developed your own. Each template normally consists of three files:

- `usersys/grammars/template/<messagetype>.py` is the grammar configured in the translation and defines the actual template files used.
- `usersys/grammars/template/templates/<messagetype>.kid` is the template containing html code for the main body of the message.
- `usersys/grammars/template/templates/<messagetype>_envelope.kid` is the envelope template containing header and footer html code.

Note: Your naming standard for the .kid files may differ from the above example

To convert this to a Genshi template, at least the following steps are required. There may be additional changes depending on your template complexity. For more info see Comparing Genshi to Kid.

1. Of course, the Genshi library must be installed for this to work.
2. Copy the three source files from `usersys/template` to `usersys/templatehtml`. The compiled .pyc files are not copied. Keep the same `/templates` sub-directory structure for the templates.
3. Also copy the `__init.py__` files when first creating this new directory structure.
4. Rename the two copied .kid files to .html (ie. just change the file extension)
5. **Edit the `<messagetype>.py` file and change it's `template` and `envelope-template` settings that refer to the .kid f**

```
syntax = {
        'charset':'utf-8',
        'contenttype':'text/html',
        'merge':False,
        'template':'<messagetype>.html',
        'envelope-template':'<messagetype>_envelope.html'}
```

6. Edit the two .html template files and make the following changes:
   - Namespaces should be changed

```
<!-- Kid namespace -->
<html xmlns:py="http://purl.org/kid/ns#">
```

```
<!-- Genshi namespaces (xmlns:xi probably needed only in envelope) -->
<html xmlns:py="http://genshi.edgewall.org/" xmlns:xi="http://www.w3.org/
↪2001/XInclude">
```

   - xi:include should be used instead of py:replace in the envelope template

```
<!-- Kid syntax for replace -->
<div py:strip="True" py:for="message in data">
    <div py:replace="document(message)"/>
</div>
```

```
<!-- Genshi syntax for include -->
<div py:strip="True" py:for="message in data">
    <xi:include href="${message}" />
</div>
```

7. Change your translation: change template to template-html.
8. Last, but not least: test this ;-)

Migrate Django versions

Migrate Django to version 1.3 or greater:

- Remove the 1.1 version of Django
- **Download new version.**
  - Mind: bots 2.2.0 does not support Django 1.4.*.
  - Django version 1.3.1 is tested and recommended.

- Install the new version
- *Restart* the bots webserver
- Be sure to use the correct bots upgrade plugin to match the version of Django you have installed.
- **Bots includes copies of some Django files in it's directory structure. You may need to refresh these from your current Djan**
  - Copy from: `<python dir>\Lib\site-packages\django\contrib\admin\media`
  - Copy to: `<python dir>\Lib\site-packages\bots\media`
  - Include sub-directories: css, img, js

Migrate Database

If you choose to use *another database* rather than SQLite, you may want to migrate some or all of your data. You may also migrate data between development and production environments. There are several approaches to this, depending on your needs.

- Migrate all configuration and transactional data
- Migrate only the configuration, start fresh with transactional data
- Migrate only partial configuration (eg. adding a new route, channels and translation)

Depending on the approach and the amount of data, several methods could be used.

- **Using bots *plugin* mechanism (write plugin, read plugin)**
  - Simple, good for configuration
  - Works independently of underlying database in use
  - **Disadvantage**: very slow (or fails with memory error) for large volumes of transactional or code-conversion data.
- **Using bots *plugin* mechanism, and "editing" the plugin**
  - Plugins are just zip files, you can open them with any zip tool
  - Use a tool that allows you to edit files within the zip and re-save them (eg. IZArc works for this)
  - Alternatively, unzip the whole directory structure, make your changes, then zip it again. Make sure the same structure is kept.
  - Plugin contains all configuration, you can remove files that are not required (eg. usersys files)
  - The file `botsindex.py` contains all of the database configuration. You can edit this file and delete records not required. Be careful to keep linked records (eg. channels used by a route). The layout of this file is not very user-friendly, you will need to use "find" in your editor a lot. Records are grouped by type, do not re-arrange their sequence.
- **Using SQL (dump, insert)**
  - If you have a large volume of data to migrate, this will probably be faster.
  - add details of how to do this here (I am testing this)
  - Additional information here http://www.redmine.org/boards/2/topics/12793.

Tools, tutorials, links, python

Lots of information, not directly related to bots it-selves. Nevertheless this information can be very useful.

Useful Tools

These are programs you can use in bots development and deployment. (**please add more**)

Text Editors

Any text editor you use should have syntax colouring/highlighting. This makes it much easier to spot any mistakes. It is also good to be able to run python to do syntax checks. Also a good search/replace function (eg to add extra CR/LF in edifact or x12 files).

- EditPlus (Windows) is the editor I have used for many years and so continued to use it with Python. There are no python syntax files in the default installation but you can easily download and add them, several versions are available. You can also add a "user tool" option to run a python syntax check directly in the editor, and one to run the bots grammar check.
- Scite (Windows, Linux) provides Python syntax highlighting and python syntax check directly in the editor to check the correctness of Python scripts.
- Geany (Windows, Linux) provides Python syntax highlighting and python syntax check directly in the editor to check the correctness of Python scripts. Quite similar to scite, but more fancy eg with spell checking. Web site calls it an IDE.
- TextWrangler (Mac) provides Python syntax highlighting and python syntax check directly in the editor to check the correctness of Python scripts. Also quite similar to Scite, very interesting access to remote scripts via SSH/SFTP.

IDE (Integrated Development Environment)

These are a **step up** from just using a text editor. Please add more if you have anny experience with these.

- Eclipse with PyDev. You just create a project in the bots directory and edit everything from that folder.

Compare and merge

Tool for comparing files and directories, analyse changes and copy code changes eg between test and production environments.

- Winmerge (Windows)
- Meld (linux)
- Tkdiff (windows,linux)
- KDiff3 (linux)

Database

- SQLite Expert (Windows, Linux) is a database browser tool for SQLite. Useful for advanced troubleshooting or learning more about Bots internal workings. You can use SQL commands to do quick updates or create reports. Note: I use an old version (1.7) as the newer versions are much larger and slower and offer nothing more useful.

- HeidiSQL (Windows) can be used similarly to the above for bots installations using the MySQL database. A lightweight interface for managing MySQL and Microsoft SQL databases. It enables you to browse and edit data, create and edit tables, views, procedures, triggers and scheduled events. Also, you can export structure and data either to SQL file, clipboard or to other servers.
- MySQL Workbench (multiple platforms) is the full blown MySQL management toolset. It provides an integrated tools environment for Database Design & Modeling, SQL Development (replacing MySQL Query Browser) and Database Administration (replacing MySQL Administrator). The Community (OSS) Edition is available from this page under the GPL.

Other tools

- IZArc (Windows) is an archive tool that allows you to open bots plugins (zip files) and easily modify their contents.
- BareTail (Windows) is a free real-time log file monitoring tool. It is useful for watching Bots log files (engine.log, webserver.log etc). Multiple tabbed interface, colour highlighting, single small program, no installer.

New to python?

Short Explanation

- Python is an interpreted language. This means for you: edit a python source file (eg mappingscript, grammar), save it and run again. No compilation, no linking.
- Python source files always have extension '.py'.
- If you see a '.pyc' file: that is an intermediate file. OK, python does some compilation automatically. Never mind, it is not important. Some systems have '.pyo' files instead. Again, not important.
- Python does not use curly braces ('{...}') or 'BEGIN...END' for functions, loops etc. Python uses 'indentation' instead. Some people love it, some hate it. For me: this is what I always did with all programming languages; now everybody uses the same layout ;-))
- Links to more learning about python are in the page with *external links*

Some tips when you are new to python

1. **Use a good text editor. This is VERY important. It saves you a lot of time. See in *tools* page. Three main reasons:**

    - a good editor has a feature called 'syntax highlighting'. This makes it is lot easier to work with eg mapping scripts.
    - a good editor can do a python syntax check on the python source file (a check if it is valid python). This will point you directly to any errors you made.
    - make the editor use spaces instead of tabs. This is an important feature when working with python. Never mix tabs and spaces.

2. **Creating grammars and mapping scripts does not require an "in depth" knowledge of Python, but you need to at least un**

    - variables and expressions
    - functions
    - strings
    - lists, dicts and tuples

3. **Start with a plugin or example**
    - most grammars are "similar" (csv, fixed are simple; edifact, x12 are more complex)
    - most mappings are "similar" (same functions are used, only "mpaths" change)
    - find something close to what you need, see how it works, adapt it.

Installing extra libraries/dependencies

- For linux lots of information about installing libraries is here. Think this is also useful for windows.
- A advanced method is using **virtualenv**

Feedback

Please share your experiences and tips via the mailing list.

External reference links

EDI

- EANCOM, In trade EANCOM is the most used edifact standard. Guides are much better than the edifact general documentation.
- UN/EDIFACT reference, The official source of edifact (the one and only). All other information is derived from this.
- Stylus Studio EDIFACT reference, I have found this page very useful as my main EDIFACT reference. All the versions are there back to D93A and everything is hyperlinked.
- Liason EDI Notepad, EDI editor/validator free for Windows platform. It provides rendering of EDI/EDIFACT/TRADACOMS/X12.

Note: Please note that the official X12 documentation is not free and should be purchased.

Note: Some of these pages are provided by companies promoting their own EDI services or software, but contain some good information.

Python

You don't need to know a lot about Python to get started with bots, but it will help you to do more advanced mapping and user scripts (or even add new features to bots code)

- Beginners Guide, New to programming? Python is free and easy to learn if you know where to start! This guide will help you to get started quickly.
- Simple Programs, Gives around 30 simple working code samples, gradually introducing more advanced concepts.
- String Methods, These are often used to manipulate string data in mapping scripts.

EDI Basics

A high level introduction to EDI - business process focused

High Level EDI Business Flow

Various documents need to be exchanged during the lifecycle of a business transaction. A generic flow is shown below, this focuses on the business processes, not a particular EDI implementation (we'll get to that later).

There may be other communication, for example business process/contracts may dictate that all changes are requested and approved by email prior to the ERP systems being modified.

Example Business Flow

- Buyer (customer) creates a purchase order (PO)
- PO is sent to supplier via EDI
- Supplier loads a Sale Order in response to the PO, with optional changes.
- Supplier acknowledges PO (with any changes)
- Buyer changes quantity of a line (part)
- Buyer-initiated Change Request is sent to supplier via EDI
- Supplier implements or rejects changes
- Supplier sends summary of changes/rejects to buyer via EDI
- Supplier needs to delete a line as they can no longer supply
- Supplier sends supplier-initiated change request to buyer/customer via EDI
- Buyer receives change request and implement or rejects changes
- Supplier ships material
- Supplier sends an Advance Ship Notice (shipping manifest) via EDI
- Supplier sends invoice via EDI

Typical EDI Document Types

- **Purchase Order**

  The Purchase Order, or PO, is sent from buyer to supplier. It is similar to a paper purchase order.

  Typical Contents

  A PO sent to a supplier typically contains:

  - Customer Purchase Order (PO) Number
  - Line level information: part codes, quantities and prices
  - Delivery Schedule (due date)
  - Incoterms
  - Delivery and invoice addresses

  Document Numbers

  - ASC X12: 850
  - UN/EDIFACT: ORDERS
  - TRADACOMS: Order File

- **Purchase Order Acknowledgement**

The Purchase Order Acknowledge is sent from supplier to buyer. It confirms acceptance of the buyer's Purchase Order along with any changes required by the supplier (for example pricing or scheduled delivery dates).

Typical Contents

A PO Acknowledgement sent to a buyer typically contains:

- Customer Purchase Order (PO) Number
- Line level information: part codes, quantities and prices
- Delivery Schedule

Document Numbers

- ASC X12: 855
- UN/EDIFACT: ORDRSP
- TRADACOMS: Acknowledgement of Order File

- **Purchase Order Change Request**

  Purchase Order Change Request documents may be sent from supplier to buyer (supplier initiated) or from buyer to supplier (buyer-initiated).

  Typical Contents

  - Original PO Number
  - Currency
  - Delivery and Invoice Addresses
  - Line(s) Add/Delete/Change code
  - Line part code, price, quantity and Unit of Measure codes

  Document Numbers

  - ASC X12: 860 (buyer initiated) and 865 (supplier initiated)
  - UN/EDIFACT: ORDCHG
  - TRADACOMS: ?

- **Advance Ship Notice(ASN)**

  The Advance Ship Notice, or ASN, electronically communicates the contents of a shipment to a trading partner. It is sent in advance of the shipment arriving at the partner's facility, this gives the receiver time to arrange storage space or onward logistics.

  Typical Contents

  An ASN sent to a customer typically contains:

  - Customer Purchase Order (PO) Number
  - Carrier information, such as name and tracking/Airway Bill number
  - Estimated Time of Arrival
  - Line level information such as part codes and quantities
  - Item or pack level information such as serial numbers

  Document Numbers

  - ASC X12: 856

  - UN/EDIFACT: DESADV
  - TRADACOMS: Delivery Notification File

- **Invoice**

  The electronic Invoice document is similar to a paper invoice.

  Typical Contents

  An Invoice sent to a customer typically contains:

  - Customer Purchase Order (PO) Number
  - Estimated Time of Arrival
  - Line level information such as part codes, quantities, prices and taxes.
  - Untaxed, tax and total amounts
  - Invoice currency
  - Payment due date
  - Payment address

  Document Numbers

  - ASC X12: 810
  - UN/EDIFACT: INVOIC
  - TRADACOMS: Invoice File

Integration with Odoo/OpenERP

Bots can add EDI support to Odoo (previously OpenERP).

Note: The standard Odoo EDI module only transfers data between two Odoo installations and is NOT X12, EDIFACT or similar EDI.

To do:

- High Level Business Flow
- The Odoo XMLRPC interface
- The bots_connector module
- Example script - Sending an X12 810 invoice
- Example script - Creating a sale order from X12 850 purchase order

Glossary of Terms

alt	Used for chained translations.
botskey	Used for business document number, eg order number, invoice number. Used to set up *document views* (view business document level).
channel type	Type of communication used for sending/receiving files like ftp, smtp, file I/O.
chained translation	A chained translations translate one incoming format to multiple outgoing formats.
editype	Type of edi-file like x12, edifact, tradacoms, xml, csv, fixed record files.
mapping script	A python script containing instructions how to get data from incoming edi-message and put it in the outgoing edi-message.
messagetype	eg 850 (an x12 order), ORDERSD96AUN (edifact order) or your inhouse xml-orders. A messagetype is defined in a grammar
route	a workflow for edi-files
translation	convert a message of a certain editype, messagetype to another editype, messagetype using a mapping script.
translation rule	determines what translations are done for incoming files. In `bots-monitor->Configuration->Translations`