Ibid Documentation

Release 0.2.0dev

The Ibid Developers

December 29, 2016

Contents

1 Introduction **1**
1.1 Getting Help with Ibid . . . 1

2 Installation **3**
2.1 Prerequisites . . . 3
2.2 Database Creation . . . 4
2.3 Package Managed Installation . . . 5
2.4 Installation From Source . . . 6

3 Configuration **9**
3.1 The botdir . . . 9
3.2 The configuration file . . . 9
3.3 Permissions . . . 13

4 Plugins **15**
4.1 Factoids . . . 15

5 Ibid Plugin Tutorial **19**
5.1 Getting Started . . . 19
5.2 Ibid Theory . . . 20
5.3 Plugin Writing Time . . . 21
5.4 Configuration . . . 23
5.5 Style . . . 23
5.6 Next Steps . . . 23

6 Contributing **25**
6.1 Bug Reporting . . . 25
6.2 Submitting Patches . . . 25
6.3 Style Guidelines . . . 26
6.4 Bazaar for Ibid Developers . . . 27
6.5 Running a Development Ibid . . . 27

7 Command line utility Reference **29**
7.1 ibid-db . . . 29
7.2 ibid-factpack . . . 30
7.3 ibid-knab-import . . . 31
7.4 ibid-memgraph . . . 32
7.5 ibid-objgraph . . . 32
7.6 ibid-pb-client . . . 33

7.7 ibid-plugin . . . 34
7.8 ibid-setup . . . 35
7.9 ibid . . . 35
7.10 ibid.ini . . . 36

8 API Reference **39**
8.1 `ibid.compat` – Python Version Compatibility . . . 39
8.2 `ibid.config` – Configuration Files . . . 40
8.3 `ibid.core` – Ibid Core . . . 41
8.4 `ibid.event` – Events . . . 43
8.5 `ibid.plugins` – Plugin registration . . . 45
8.6 `ibid.test` – Ibid Testing . . . 48
8.7 `ibid.utils` – Helper functions for plugins . . . 49

9 Changes in Ibid **53**
9.1 Release 0.1.1 “Pimpernel” (2011-02-24) . . . 53
9.2 Release 0.1.0 “Hazel” (2010-03-10) . . . 59

10 Indices and tables **61**

Python Module Index **63**

CHAPTER 1

Introduction

This is the documentation for Ibid. Ibid is a multi-protocol general purpose chat bot written in Python. It uses a natural language interface, and can connect to multiple sources including IRC, Jabber and SILC servers, as well as allowing interaction using SMTP, HTTP and various RPC protocols. It aims to be a useful base to build any kind of chat bot from, and to make plugins and extensions as easy as possible to write.

Ibid is in constant development. The code is hosted on Launchpad.

The most recent version of this documentation is available at http://ibid.omnia.za.net/docs/.

1.1 Getting Help with Ibid

- The Ibid Developers can be found in `#ibid` on `irc.atrum.org`.
- Alternatively, file a Question on Launchpad.

CHAPTER 2

Installation

The preferred method of installing a production Ibid is to use the packages provided by your distro. Ibid is available in Debian (since squeeze) and Ubuntu (since Lucid). Additionally, private backport repositories are available for supported stable releases of:

- Debian (lenny)
- Ubuntu (hardy-karmic)

Installing Ibid *through your distribution's package-management system* should give you the least hassles in the long run [1].

For plugin-development this will be sufficient, but for *interaction with the Ibid development community* (i.e. contributing code) or drastic internal changes, you will probably want to work *from source* instead.

2.1 Prerequisites

2.1.1 Python

You'll need a sane Python environment and a few Python 3rd-party packages, described below.

We attempt to support all Python 2.x releases >= 2.4. However, we'd recommend the most recent stable 2.x release, as Python memory usage has improved and more recent Python releases have been better tested with Ibid. (We have to go out of our way to test with 2.4...) Python 3.x support is off the cards until our dependencies are 3.x capable.

We expect Ibid to work on most operating systems that can provide Python, but only develop and test heavily on Debian and Ubuntu Linux.

2.1.2 Python Libraries:

- Twisted framework (core sources)
- Twisted Words (IRC, XMPP)
- Wokkel. (XMPP)
- SQLAlchemy 0.6 or later.
- ConfigObj >= 4.7.0
- python-dateutil

[1] Your distribution of choice not listed here? That's probably because none of the current Ibid developers use it. Why not *chip in* and help us package Ibid for you.

- SOAPpy [3]
- Setuptools

Many core plugins require the following Web scraping & parsing libraries:

- ElementTree (only needed for Python 2.4)
- html5lib
- BeautifulSoup
- SimpleJSON (only needed for Python < 2.6)

Web source and web services:

- Jinja2

Other sources:

- SILC: pysilc

There are many non-essential plugins that require other libraries or programs, they are described in the *plugin documentation*.

2.1.3 Database

Ibid needs a relational database [2]. MySQL, PostgreSQL, and SQLite are all supported as first-class citizens. Using the database-independent backup & restore tool, it is straightforward to migrate between database engines at any time.

2.2 Database Creation

By default Ibid will use SQLite, which is a perfectly capable database, but if you have a MySQL or PostgreSQL server handy, you may prefer to use it to save some memory or take advantage of the more powerful feature-set.

If you are using SQLite, you can skip over this section, but for MySQL or PostgreSQL, you'll need to create a database and DB user before running `ibid-setup`.

2.2.1 MySQL

Install MySQL. On Debian/Ubuntu:

```
user@box $ sudo aptitude install mysql-server python-mysqldb
```

You'll be prompted to set a root password for your server. You should probably do that.

Create a database for your bot:

```
user@box $ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.
mysql> CREATE DATABASE joebot CHARSET utf8;
Query OK, 1 row affected (0.02 sec)

mysql> GRANT ALL PRIVILEGES ON joebot.* TO joebot@localhost IDENTIFIED BY 'mysecret';
Query OK, 0 rows affected (0.13 sec)
```

[3] SOAPpy can be hard to install, so we have debian packages and eggs to help. `setup.py` knows where to look.
[2] If you don't need user-accounts (and many other features), the database code could be removed. It'd probably be quite a bit of work, though.

```
mysql> quit
Bye
```

In this example, the database is called `joebot`, the user `joebot` and the password is `mysecret`, so the DB URL will be:

```
mysql://joebot:mysecret@localhost/joebot
```

2.2.2 PostgreSQL

Install PostgreSQL. You'll also need the `citext` contributed module. On Debian/Ubuntu:

```
user@box $ sudo aptitude install postgresql postgresql-contrib python-psycopg2
```

Create a database for your bot:

```
user@box $ sudo -u postgres -i
postgres@box $ createuser -D -R -S -P joebot
Enter password for new role:
Enter it again:
postgres@box $ createdb -O joebot joebot
postgres@box $ psql -f /usr/share/postgresql/8.4/contrib/citext.sql joebot
postgres@box $ logout
```

In this example, the database is called `joebot` and the user `joebot` if the password were `mysecret`, the DB URL would be:

```
postgres://joebot:mysecret@localhost/joebot
```

2.3 Package Managed Installation

2.3.1 Add the APT source

These repositories are only necessary if you are using an old Debian / Ubuntu release that doesn't include Ibid.

Debian (lenny):

> `deb http://ibid.omnia.za.net/debian/ lenny-backports main`
> GPG Key: 0x5EB879CE

Ubuntu (pre-lucid):

> `deb http://ppa.launchpad.net/ibid-core/ppa/ubuntu karmic main`
> If you are using a different release to `karmic`, substitute its name.
> GPG Key: 0xFD1C44BA

You can follow these instructions or add it from a terminal like this:

```
user@box $ echo deb http://ppa.launchpad.net/ibid-core/ppa/ubuntu `lsb_release -cs` main | sudo tee /
user@box $ sudo apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xFD1C44BA
user@box $ sudo aptitude update
```

2.3.2 Install Ibid

```
user@box $ sudo aptitude install ibid
```

Now you should probably create a user for your bot to run as. While every effort is made to ensure that your bot won't do naughty things, we can't guarantee that there is no way to exploit it. If you are feeling adventurous, skip down to creating a bot directory:

```
user@box $ sudo adduser --disabled-login ibid
```

Switch to the bot user:

```
user@box $ sudo -u ibid -i
ibid@box $
```

If you are going to be using MySQL or PostgreSQL *set up your database now*.

Then you'll need to create a directory for your bot to live in:

```
ibid@box $ mkdir botdir
ibid@box $ cd botdir
```

Now you can install the bot:

```
ibid@box $ ibid-setup
Couldn't load core plugin: botname
Couldn't load knab plugin: No module named perl
Couldn't load trac plugin: argument of type 'NoneType' is not iterable
What would you like to call your bot? joebot
Please enter the full URL of the database to use, or just press Enter for an SQLite database.
Database URL:
Please enter the details for the primary source. Press Enter for the default option.
Source name (e.g. freenode, atrum, jabber): freenode
Server: irc.freenode.net
Port:
Source type (irc or jabber): irc
Default channels to join (comma separated): #myawesomechannel
Nick/JID: joeuser
Password: [my password]
Account created with admin permissions
```

Note: This will throw out some harmless errors (about plugins that you don't have pre-requisites for).

Load any factpacks you desire (in this case, common greetings):

```
ibid@box $ ibid-factpack greetings.json
```

Now would be the time to configure your bot. But for now, let's just get it running:

```
ibid@box $ twistd -n ibid
```

You should see copious debugging output, and the bot should log into your IRC channel.

2.4 Installation From Source

If you want to do any development, or install from trunk or a specific branch, you'll need Bazaar installed.

Firstly, you need the dependencies *listed above*. We recommend a recent release of Debian/Ubuntu Linux, and the instructions are tailored for such. If you use something else, you'll have to interpolate.

Install the required python modules. You can use another DB, but we default to SQLite. If you are not using Debian/Ubuntu or would prefer to have these dependencies installed in a virtualenv, you can skip this step:

```
user@box $ sudo aptitude install bzr python-configobj python-sqlalchemy \
  python-twisted python-beautifulsoup python-celementtree \
  python-html5lib python-setuptools python-simplejson \
  python-soappy python-jinja2 python-dateutil python-virtualenv
```

Create a user to run your bot as:

```
user@box $ sudo adduser --disabled-login ibid
```

Create a virtualenv to install Ibid to:

```
user@box $ virtualenv ve
```

Note: This isn't strictly necessary as Ibid can run out of a source checkout for development. But for long-term deployments it is sensible to separate the source from the botdir.

Checkout the latest version of Ibid (instead of this, you could extract a source tarball):

```
user@box $ sudo -u ibid -i
ibid@box $ bzr branch lp:ibid
ibid@box $ cd ibid
```

Install Ibid:

```
user@box $ . ~/ve/bin/activate
user@box $ ./setup.py install --no-dependencies
```

Note: If you didn't install the packages listed in the first step, you'll have to remove `--no-dependencies` so setuptools can do its magic.

If you are going to be using MySQL or PostgreSQL *set up your database now*.

Then you'll need to create a directory for your bot to live in:

```
ibid@box $ mkdir ~/botdir
ibid@box $ cd ~/botdir
```

Set up your bot:

```
ibid@box $ ibid-setup
```

Note: This will throw out some harmless errors (about plugins that you don't have pre-requisites for).

If you haven't created a configuration file, it will ask you to give the bot a name, and describe the first source. A source is an IRC network, jabber, or SILC network.

It'll ask you to enter the details of the first administrative account. Assuming you will be connecting the bot to an IRC server, enter your nick, the network's name, and a password (e.g. "joebloggs", "freenode", "s3cr3tpass").

Load any factpacks you desire (in this case, common greetings):

```
ibid@box $ ibid-factpack ~/ibid/factpack/greetings.json
```

Run your bot:

```
ibid@box $ twistd -n ibid
```

CHAPTER 3

Configuration

3.1 The botdir

Every Ibid lives in a directory, the botdir. This holds the configuration file, logs, caches, the SQLite database if you are using it, and plugins you've written.

The botdir should be your current directory whenever you run an ibid-related script, so it can find the configuration file.

All non-absolute paths in the ibid configuration are relative to the botdir.

Note: The botdir is added to the front of `sys.path`, so any python package that you put in the botdir will be available to the bot, and take precedence over other versions of the same package.

3.2 The configuration file

Ibid's configuration is stored in `ibid.ini`, created when you install Ibid. You can edit it at any time and tell the bot to `reread config` or edit it online with the *config* feature.

A simple example ibid.ini:

```
botname = joebot
logging = logging.ini

[auth]
    methods = password,
    timeout = 300
    permissions = +factoid, +karma, +sendmemo, +recvmemo, +feeds, +publicresponse

[sources]
    [[telnet]]
    [[timer]]
    [[http]]
        url = http://joebot.example.com
    [[smtp]]
    [[pb]]
    [[atrum]]
        channels = "#ibid",
        nick = $botname
```

```
        type = irc
        auth = hostmask, nickserv
        server = irc.atrum.org

[plugins]
    cachedir = /tmp/ibid
    [[core]]
        names = $botname, bot, ant
        ignore = ,

[databases]
    ibid = sqlite:///ibid.db
```

This shows the main sections of the file. It is in configobj format, an ini-variant with nested sections. Whitespace is ignored, all values belong to the most recently defined section.

Lines can be commented out by prefixing them with `#`.

3.2.1 Top Level Options

`botname:`
String: The name of the bot. It will respond to this name.

`logging:`
String: The location of the `logging` configuration file.

Default: `logging.ini`

`mysql_engine:`
String: The engine that MySQL tables will be created in.

Default: `InnoDB`

3.2.2 Auth

This section is for configuring Ibid's user-authentication. Permissions that are granted `...when authed` require users to authenticate themselves to the bot before the permission can be invoked. Some sources have special ways of authenticating users (e.g. the `nickserv` authentication method on IRC) or guarantee that their users are always authenticated via the `implicit` authentication method (e.g. jabber).

`methods:`
List: Authentication methods that can be used on all sources.

`timeout:`
Number: Time in seconds that authentication should be cached for before requiring re-authentication.

`permissions:`
List: Permissions that are granted to everyone. Although they can be overridden for specific users, using the online grant function.

The name of the permission can be prefixed with a + to indicate that this permission is granted without requiring authentication. Or a − to revoke a permission granted to all users of a source.

See *the list of permissions*.

3.2.3 Sources

Sources are the way that Ibid connects to users. Every IRC/SILC/DC server is it's own source as are connections to other services.

The configuration parameters that applies to all sources are:

disabled:
Boolean: Every source can be disabled from auto-starting by setting this to `True` in the source's configuration.

type:
String: The driver that this source should use. This allows you to have more than one IRC source, for example.

Default: The name of the source. If you specify a type, you are free to name the source anything you want to.

permissions:
List: This lets you grant and revoke permissions to all users on the source. They can be overridden for specific users, using the online grant function.

The name of the permission can be prefixed with a + to indicate that this permission is granted without requiring authentication. Or a − to revoke a permission granted to all users of a source.

See *the list of permissions*.

IRC Source

Some of the IRC functionality (i.e. NickServ authentication and joining/parting channels) is handled by the `irc` plugin.

server:
Reqired String: The hostname of the IRC server to connect to.

Ibid does not currently support falling back to alternate servers, so you may want to use a round-robin hostname.

port:
Number: The port to connect to.

Default: `6667`

ssl:
Boolean: Use SSL-encrypted connection to the network.

Default: `False`

nick:
String: The nickname for the bot to use on this network.

Default: The `botname`

modes:
String: The IRC modes to set. Some servers require bots to set mode `B`.

Default: nothing

channels:
List: Channels to join on connection to the network.

> **Warning:** You must include the leading `#`, but unless you quote each channel, Ibid will see the rest of the config line as a comment.
> So use quotes around each channel name like this: `"#ibid", "#fun"`

nickserv_password:
String: The password identifying your bot to NickServ. If set, the bot will respond to authentication requests from NickServ.

Default: Nothing

nickserv_mask:
String: The NickServ's hostmask on this network. You can set this to ensure that you don't accidentally give your NickServ password to an imposter, should the network's services module go down.

You can use glob wildcards, i.e. `*` and `?`.

Default: `*`

nickserv_nick:
String: The NickServ's nickname on this network. You probably won't need to change it.

Default: `NickServ`

ping_interval:
Number: How many seconds in between each keep-alive PING sent to the server.

Default: `60`

pong_timeout:
Number: How long to wait for PONGs before giving up and reconnecting.

Default: `300`

Jabber Source

jid:
Required String: The jabber ID that the bot will connect with. (This looks like an e-mail address)

password:
Required String: The password for the supplied JID.

rooms:
List: MUC chatrooms to join on connection.

Default: Nothing

accept_domains:
List: Domains that the bot will accept messages from. If this isn't set, it'll accept messages from anyone.

Default: Nothing (i.e. no restriction)

server:
String: The hostname of the server to connect to.

Default: Determined automatically from the jabber ID.

port:
Number: The port to connect to.

Default: `5222` or `5223` if `ssl` is `True`

ssl:
Boolean: Use old port 5223-style SSL connection instead of opportunistic TLS on port 5222.

Default: `False`

nick:
String: The nickname for the bot to use on this server when in MUC chatrooms.

Default: The `botname`

max_public_message_length:
Number: The bot will limit public (i.e. MUC) messages to this length (in bytes) to avoid flooding the channel with long messages.

Default: `512`

3.3 Permissions

The following permissions are used in Ibid core:

accounts Alter user's accounts.

admin Grant and revoke permissions. Shut down the bot.

config Alter configuration values online. (Rewrites the configuration file)

core Reload Ibid core components.

plugins Load and unload plugins.

sources Start and stop sources. Join and leave channels.

Other permissions used in plugins:

chairmeeting Start meeting minute-taking.

eval Execute arbitrary Python code.

factoid Set factoids and modify factoids that you have set yourself.

factoidadmin Delete / modify factoids that you didn't set in the first place.

feeds Configure RSS/Atom feeds

karma Promote or demote karma for things.

karmaadmin Delete karma items.

recvmemo Receive memos.

saydo Use the bot as a puppet.

sendmemo Send memos.

summon Summon a user via another source.

CHAPTER 4

Plugins

Plugins are at the heart of Ibid, they provide all the features. There are core plugins that provide features such as account management, and plugin management. Then there are feature plugins, such as factoids, RSS/Atom feeds, dictionary lookup, etc.

4.1 Factoids

Factoids are one of the most important Ibid plugins. They are what give a bot most of its personality, and after a few years on IRC, it can expect to pick up a few thousand of them from its users.

While it can store simple factoids like:

```
<Alice> Ibid is an awesome bot written in Python
<ibid> If you say so
<Alice> Ibid
<ibid> Ibid is an awesome bot written in Python
```

It can also allow for some basic behaviour programming:

```
<Alice> slap $arg is <action>slaps $1 and runs for his life
<ibid> Got it
<Alice> slap Bob
* ibid slaps Bob and runs for his life
```

4.1.1 Basics

Factoids start out pretty basic, as a way of the bot remembering things, such as:

```
<Alice> ibid: Ibid is an awesome bot framework
<ibid> Alice: One learns a new thing every day
```

Now I can ask him about that:

```
<Alice> ibid: what is ibid?
<ibid> Alice: Ibid is an awesome bot framework
```

Or even:

```
<Alice> ibid: Ibid
<ibid> Alice: Ibid is an awesome bot framework
```

They can also store multiple values. On retrieval, a value is picked at random:

```
<Alice> ibid: Ibid is also your father
<ibid> Alice: I'll remember that
<Alice> ibid: Ibid
<ibid> Alice: ibid is your father
<Alice> ibid: Ibid
<ibid> Alice: ibid is an awesome bot framework
```

You can request a specific one:

```
<Alice> ibid: Ibid /frame/
<ibid> Alice: ibid is an awesome bot framework
<Alice> ibid: literal ibid
<ibid> Alice: 1: is an awesome bot framework, 2: is your father
<Alice> ibid: Ibid #2
<ibid> Alice: ibid is your father
```

4.1.2 Fancy factoids

Factoids don't just have to be like a dictionary, they can have some basic behaviour attached. This example also shows how you can trivially replace factoids (assuming you have permission to do so):

```
<Alice> ibid: hi is <reply>Hi there
<ibid> Alice: One learns a new thing every day
<Alice> ibid: hi
<ibid> Hi there
<Alice> ibid: no, hi is <action>waves
<ibid> Alice: Got it
<Alice> ibid: hi
 * ibid waves
<Alice> ibid: no, hi is <reply>Hello $who
<ibid> Alice: If you say so
<Alice> ibid: hi
<ibid> Hello Alice
```

The `$who` substitutes the speaker's name. Other substitutions possible are:

- `$channel`, `$source`
- `$year`, `$month`, `$mon`, `$month1`, `$month2`, `$day`, `$day2`, `$hour`, `$minute`, `$second`, `$date`, `$time`, `$weekday`, `$unixtime`

`$month`, `$mon`, `$month1` and `$month2` substitute, respectively, the month name, abbreviated month name, month number and zero-padded month number; `$day2` substitutes the zero-padded day number.

And factoids can have multiple names. Our greeting only responds to "hi", it should respond to more greetings:

```
<Alice> ibid: hello is the same as hi
<ibid> Alice: Yessir
<Alice> ibid: good day is the same as hi
<ibid> Alice: Alrighty
<Alice> ibid: good day
<ibid> Hello Alice
```

Finally, factoids can take basic arguments:

```
<Alice> ibid: slap $arg is <action>slaps $1
<ibid> Alice: Got it
<Alice> ibid: slap Alice
```

```
 * ibid slaps Alice
<Alice> ibid: run from $arg to $arg is <action>starts running at $1 and stops at $2
<ibid> Alice: One learns a new thing every day
<Alice> ibid: run from here to there
 * ibid starts running at here and stops at there
```

4.1.3 Searching

Factoids aren't very useful if we can't remember what they are called:

```
<Alice> ibid: search slap
<ibid> Alice: slap $arg [1]
<Alice> ibid: search father
<ibid> Alice: Ibid [2]
```

The first example tells us about a factoid mentioning slap. The second shows what happens when a multi-valued factoid is found. The result we are after may be one of two values, so we can't simply say "`ibid: ibid`", we should rather use say:

```
<Alice> ibid: ibid /father/
<ibid> Alice: Ibid is your father
```

You can specify which part of the factoid you are searching in:

```
<Alice> ibid: search for values containing awesome
<ibid> Alice: Ibid [2]
<Alice> ibid: search for names containing awesome
<ibid> Alice: I couldn't find anything with that name
```

4.1.4 Modification

Factoids aren't set in stone. While they can be replaced with "`no, name is newvalue`", sometimes you just want to make a small change:

```
<Alice> ibid: ibid
<ibid> Alice: Ibid is an awesome bot framework
<Alice> ibid: ibid += , written in python
<ibid> Alice: Pattern matches multiple factoids, please be more specific
<Alice> ibid: ibid /awesome/ += , written in python
<ibid> Alice: Done
<Alice> ibid: ibid /awesome/
<ibid> Alice: Ibid is an awesome bot framework, written in python
<Alice> ibid: ibid /python/ ~= s/awesome //
<ibid> Alice: Yessir
<Alice> ibid: ibid /python/
<ibid> Alice: Ibid is an bot framework, written in python
```

Note: The replacement is a simple search & replace. If you want to use a regular expression, you must add an `r` flag. E.g. s/fo+/bar/r

The `i` and `g` flags are also supported, as is the `y/x/y/` operation.

And finally, removal:

```
<Alice> ibid: forget ibid
<ibid> Alice: Sure
<Alice> ibid: ibid
<ibid> Alice: What?
```

Or only one of the values:

```
<Alice> ibid: literal ibid
<ibid> Alice: 0: is your father, 1: is an awesome bot framework
<Alice> ibid: forget ibid /framework/
<ibid> Alice: Sure
<Alice> ibid: literal ibid
<ibid> Alice: 0: is your father
```

CHAPTER 5

Ibid Plugin Tutorial

This will guide you through the process of creating an Ibid plugin so you can add your own features.

5.1 Getting Started

5.1.1 Install an Ibid

Before we can write a plugin, we need a *working base Ibid install*.

5.1.2 The Testing Environment

Rather than working on a live bot and having to reload modules a lot, when developing for Ibid, we usually use `ibid-plugin`, a minimal, fast, testing environment.

It looks like this:

```
user@box ~/botdir $ ibid-plugin
... Messages about loading plugins
Query: hello
Response: Huh?
Query: help
Response: I can help you with: looking things up.
Ask me "help me with ..." for more details.
Query: help me with looking things up
Response: I use the following features for looking things up: help
Ask me "how do I use ..." for more details.
```

To exit, press `Control-C` or `Control-D`.

As you can see, there is almost nothing loaded. It can't even respond to "hello", the code for that is in the *factoid* module. If you want to load the *factoid* module, you can tell `ibid-plugin` to load it on startup, by adding it as a parameter:

```
user@box ~/botdir $ ibid-plugin factoid
... Messages about loading plugins
Query: hi
Response: good morning
```

Well, actually there are some administrative functions, which don't show up in the overall help. You could have asked the bot to "load factoid":

```
Query: help admin
Response: I use the following features for administrative functions:
config, core, die, help, plugins, sources and version
Ask me "how do I use ..." for more details.
Query: how do I use plugins
Response: Lists, loads and unloads plugins. You can use it like this:
  list plugins
  (load|unload|reload) <plugin|processor>
Query: load factoid
DEBUG:core.reloader:Loading Processor: factoid.Forget
DEBUG:core.reloader:Loading Processor: factoid.Get
DEBUG:core.reloader:Loading Processor: factoid.Modify
DEBUG:core.reloader:Loading Processor: factoid.Search
DEBUG:core.reloader:Loading Processor: factoid.Set
DEBUG:core.reloader:Loading Processor: factoid.StaticFactoid
DEBUG:core.reloader:Loading Processor: factoid.Utils
DEBUG:core.reloader:Loaded factoid plugin
Response: factoid reloaded
Query: hi
Response: howsit
```

Try talking to it too fast, it'll start ignoring you. This makes sense for a real chat channel, but not debugging. You can tell the *ignorer* not to load by adding it as a parameter followed by `-`:

```
user@box ~/botdir $ ibid-plugin factoid core.Ignore-
```

If you want all the normal modules loaded, you can add the `-c` option, but it'll take quite a bit longer to start up:

```
user@box ~/botdir $ ibid-plugin -c
... Screenfulls of messages
Query: hello
Response: sup
```

Play with that a bit. It isn't exactly the same as a full bot, there are a few things that won't work, but it's good enough for testing. Some examples:

- Karma, because it's disabled for private conversations by default. You can switch to public mode with `-p`.
- The games, because they require some advanced Twisted functionality (as well as other channel members).

5.2 Ibid Theory

Ibid is divided into two main parts (excluding the Ibid core code): *Sources* and *Plugins*.

The sources speak IRC, Jabber, e-Mail, etc. There is one source for each network that the bot is connected to. When someone says something in an IRC channel, the IRC source for that network will create an *Event*. The event is passed to the plugins, which each take a turn to look at it and decide if they want to do anything. If a plugin decides to reply, the event is sent back to the source to dispatch the reply.

Events are also used for, private messages from users to the bot, people joining and leaving channels, etc. but most plugins don't need to deal with anything except message events, directed to the bot.

Ibid comes with some plugins for pre- and post-processing of events (such as logging), and some for features.

5.3 Plugin Writing Time

5.3.1 Processors and Handlers

Let's see what that looks like in practice. Here's a simple hello world plugin. Create a directory called `ibid/plugins` in the botdir. In that directory, create a file called `tutorial.py` with the following contents:

```
from ibid.plugins import Processor, handler

class HelloWorld(Processor):
   @handler
   def hello(self, event):
      event.addresponse(u'Hello World!')
```

A plugin can contain multiple *Processor*s. Each one is a self-contained part of the event handling chain. It can register an interest in certain types of event, or a specific place in the chain, but for most plugins the defaults are sufficient.

Inside the processor, any functions decorated with *`@handler`* will get a chance to look at the event. If it choses to add a response to the event, the response will be returned to the user.

Note: Ibid uses unicode strings and to catch mistakes, you'll get a warning if you pass a normal string as a response, so try to get in the habit of using unicode.

Test it out, anything you say to the bot should provoke a "Hello World!" response:

```
user@box ~/botdir $ ibid-plugin tutorial
... Messages about loading plugins
Query: hello
Response: Hello World!
```

Now, you could include code inside your handler to determine if you want to reply to a message or not, but must of the time you are after messages that look like something particular, so we have another decorator, *`@match()`*, to help you:

```
from ibid.plugins import Processor, match

class HelloWorld(Processor):
    @match(r'^hello$')
    def hello(self, event):
        event.addresponse(u'Hello World!')
```

Match takes a regular expression as a parameter, and will only run your handler function if the regex matches the event's message. In this case, it'll only fire if you say "hello". It'll ignore trailing punctuation and whitespace, as that's removed by the `core.Strip` plugin.

5.3.2 Match Groups

Time for a more complex example, a multiple dice roller, you can add it as another Processor in your tutorial plugin:

```
from random import randint

from ibid.plugins import Processor, match
from ibid.utils import human_join

class Dice(Processor):
```

```
    @match(r'^roll\s+(\d+)\s+dic?e$')
    def multithrow(self, event, number):
        number = int(number)
        throws = [unicode(randint(1, 6)) for i in range(number)]
        event.addresponse(u'I threw %s', human_join(throws))
```

If you still have an `ibid-plugin` open you can "reload tutorial" to reload your plugin.

Any match groups you put in the regex will be passed to the handler as arguments, in this case the number of dice to throw. If you want brackets without creating a match group, you can use the non-grouping syntax `(?:  )`.

`ibid.utils` contains many handy helper functions. *`human_join()`* is the equivalent of `u', '.join()`, with an "and" before the last item.

`addresponse()` takes a second argument for string substitution. If you want to substitute multiple items, use the dict syntax:

```
event.addresponse(u'Nobody %(verb)s the %(noun)s!', {
    'verb': u'expects',
    'noun': u'Spanish Inquisition',
})
```

5.3.3 Documentation

At the moment you'll see that your plugin doesn't appear in the help system. You can fix that with a little more code:

```
from random import randint

from ibid.plugins import Processor, match
from ibid.utils import human_join

features = {}

features['dice'] = {
    'description': u'Throws multiple dice',
    'categories': ('fun',),
}

class Dice(Processor):
    usage = u'roll <number> dice'

    features = ('dice',)

    @match(r'^roll\s+(\d+)\s+dic?e$')
    def multithrow(self, event, number):
        number = int(number)
        throws = [unicode(randint(1, 6)) for i in range(number)]
        event.addresponse(u'I threw %s', human_join(throws))
```

The module-level `features` dict specifies descriptions for features (given with the usage description) and categories to place the feature in. You can find a list of available categories in `ibid.categories` and if necessary add a category to it from your module.

The Processor can be linked to a feature by specifying it in the *features* attribute. Usage for the Processor's functions (in BNF) goes in a *usage* attribute. "reload tutorial" and you should see "dice" appear in the features for "fun stuff".

5.4 Configuration

Ibid has a configuration system that may be useful for your plugin. Configuration values can be set at runtime or by editing `ibid.ini`.

Let's make the number of dice sides be configurable:

```
from random import randint

from ibid.config import IntOption
from ibid.plugins import Processor, match
from ibid.utils import human_join

class Dice(Processor):
    sides = IntOption('sides', 'Number of sides to each die', 6)

    @match(r'^roll\s+(\d+)\s+dic?e$')
    def multithrow(self, event, number):
        number = int(number)
        throws = [unicode(randint(1, self.sides)) for i in range(number)]
        event.addresponse(u'I threw %s', human_join(throws))
```

IntOption() creates a configuration value called `plugins.tutorial.sides` with a default value of 6. There are also configuration helpers for other data types.

If you merge the following into your `ibid.ini`, you can change to 21 sided dice:

```
[plugins]
   [[tutorial]]
      sides = 21
```

5.5 Style

Now that you've got all the basics, here are some other things you should know about writing Ibid plugins.

5.5.1 Error Handling

You might have noticed that we haven't said anything about error handling. That was intentional. All exceptions in plugins are caught at the dispatcher level, and an appropriate response will be returned to the user, as well as tracebacks logged. The only time you should worry about handling errors is if you can recover gracefully or you want to return a specific response (such as an explanation).

5.5.2 Responses

The general Ibid style is that the bot should be something people can relate to, not too mechanical. So many Ibid responses are playful and maybe a little snarky. Also, many responses aren't static, but rather chosen from a list of 3 or 4 at random (`random.choice()` is good for that).

5.6 Next Steps

That's it, you are now more than able to write your own Ibid plugins. Please *send us* anything you write, it may be useful for other people too.

We wished there was more documentation we could point you at, to help you, but it hasn't been written yet. So, read some modules to see what's there, and stick your nose in our IRC channel for help.

CHAPTER 6

Contributing

6.1 Bug Reporting

Please report any bugs in the Launchpad tracker. (Oh, and check for existing ones that match your problem first.)

Good bug reports describe the problem, include the message to the bot that caused the bug, and any logging information / exceptions from `ibid.log`.

6.2 Submitting Patches

Want to go one step further, and fix your bug or add a new feature. We welcome contributions from everyone. The best way to get a patch merged quickly is to follow the same development process as the Ibid developers:

1. If you don't have one, create a Launchpad account and configure Bazaar to use it.
2. If there isn't a bug in the tracker for this change, file one. It motivates the change.
3. Mark the bug as *In Progress*, assigned to you.
4. Take a branch of Ibid trunk (See *Bazaar for Ibid Developers* if you are new to Bazaar):

```
user@box $ bzr branch lp:ibid description-1234
```

   `description` is a two or three-word hyphen-separated description of the branch, `1234` is the Launchpad bug number.

5. Fix your bug in this branch, following the *Style Guidelines*. See also *Running a Development Ibid*.
6. Link the commit that fixes the bug to the launchpad bug:

```
user@box $ bzr commit --fixes lp:1234
```

7. Test that the fix works as expected and doesn't introduce any new bugs. `pyflakes` can find syntax errors you missed.
8. Run the test-cases:

```
user@box $ trial ibid
```

9. Push the branch to Launchpad:

```
user@box $ bzr push lp:~yourname/ibid/description-1234
```

10. Find the branch on Launchpad and propose it for merging into the Ibid trunk.

11. Proposals require approvals by a member of ibid-core and two members of ibid-dev (or simply two members of ibid-core).

    Please join ibid-dev and help out with review.

6.3 Style Guidelines

Writing code that matches the Ibid style will lead to a consistent code base thus happy developers.

- Follow PEP 8, where it makes sense.
- 4 space indentation.
- Single quotes are preferred to double, where sensible.
- Almost all of Ibid should be compatible with Python 2.4+ (but not 3). Compatibility functions, imports, and libraries can be found in *ibid.compat*.
- There is more on good style in Code Like a Pythonista: Idiomatic Python.

6.3.1 Naming Conventions

- Features should either go into an existing plugin, or if large enough into a plugin of the same name as the feature (singular).
- Database table names are plural.

6.3.2 Sources

- Follow Twisted style.

6.3.3 Plugins

- All features should have help and usage strings.
- Try to code for the general case, rather than your specific problem. `Option` configurables are handy for this, but don't bother making things that will never be changed configurable (i.e. static API endpoints).
- Use `event.addresponse`'s string formatting abilities where possible. This will aid in future translation.
- Any changes to database schema should have upgrade-rules included for painless upgrade by users.
- Write tests for your code at `ibid/test/plugins/test_pluginname.py`. The *ibid.test* module provides an API for testing plugins.

6.3.4 Bot Responses

- While there are exceptions, a well behaved Ibid only speaks when spoken to.
- URLs should be surrounded with whitespace to help clients detect them.

6.4 Bazaar for Ibid Developers

You'll want a non-ancient version (>=1.16) of Bazaar (check your distribution's backport repository), and a Launchpad account.

If you've never used Bazaar before, read Bazaar in five minutes.

Configure Bazaar to know who you are:

```
~ $ bzr whoami "Arthur Pewtey <apewtey@example.com>"
~ $ bzr launchpad-login apewtey
```

Make a Bazaar shared repository to contain all your Ibid branches:

```
~ $ mkdir ~/code/ibid
~ $ cd ~/code/ibid
~/code/ibid $ bzr init-repo --2a .
```

Check out Ibid trunk:

```
~/code/ibid $ bzr checkout lp:ibid trunk
```

When you wish to create a new branch:

```
~/code/ibid $ bzr update trunk
~/code/ibid $ bzr branch trunk feature-1234
```

If you want to easily push this to Launchpad, create a `~/.bazaar/locations.conf` with the following contents:

```
[/home/apewtey/code/ibid]
pull_location = lp:~apewtey/ibid/
pull_location:policy = appendpath
push_location = lp:~apewtey/ibid/
push_location:policy = appendpath
public_branch = lp:~apewtey/ibid/
public_branch:policy = appendpath
```

That will allow you to push your branch to `lp:~apewtey/ibid/feature-1234` by typing:

```
~/code/ibid/feature-1234 $ bzr push
```

To delete a branch, you can simply `rm -r` it.

See also:

- Launchpad code hosting documentation
- Using Bazaar with Launchpad
- Bazaar User Guide
- Bazaar Reference

6.5 Running a Development Ibid

A full-blown Ibid install is overkill for development and debugging cycles.

Ibid source contains a developer-oriented `ibid.ini` in the root directory. This uses SQLite and connects to a South African IRC server. If you wish to change it, either remember not to commit this file to your branch, or override settings in `local.ini`, which is ignored by Bazaar.

Ibid can be simply run out of a checkout directory:

```
~/code/ibid/feature-1234 $ scripts/ibid-setup
```

If you won't need an administrative account, you can hit `^D` and avoid setting one up.

Test a specific plugin:

```
~/code/ibid/feature-1234 $ scripts/ibid-plugin pluginname
```

Test with all plugins loaded:

```
~/code/ibid/feature-1234 $ scripts/ibid-plugin -c
```

Note: Not all plugin features will work in the `ibid-plugin` environment. In particular, anything relying on source-interaction or timed callbacks (such as many of the games). Also, all permissions are granted.

If `ibid-plugin` isn't sufficient for your debugging needs, you can launch a normal Ibid by running:

```
~/code/ibid/feature-1234 $ twistd -n ibid
```

CHAPTER 7

Command line utility Reference

7.1 ibid-db

7.1.1 SYNOPSIS

`ibid-db` *command* [*options...*]

7.1.2 DESCRIPTION

This utility is for offline management of your Ibid bot's database. Used for import, export, and upgrades.

The export format is DBMS-agnostic and can be used to migrate between different databases.

7.1.3 COMMANDS

-e FILE, --export=FILE Export DB contents to *FILE*. Export format is JSON. *FILE* can be - for *stdout* or can end in `.gz` for automatic gzip compression.

-i FILE, --import=FILE Import DB contents from *FILE* as exported by this utility. *FILE* can be - for *stdin* or can end in `.gz` for automatic gzip compression.

Note: The DB must be empty first.

-u, --upgrade Upgrade DB schema to the latest version. You need to run this after upgrading your bot.

Note: You should backup first.

7.1.4 OPTIONS

--version Show the program's version and exit.

-h, --help Show a help message and exit.

-v, --verbose Turn on debugging output to stderr.

7.1.5 FILES

ibid.ini Locates the database to act upon by looking for the [**databases**].**ibid** value in the bot configuration file in the current directory.

7.1.6 SEE ALSO

`ibid`(1), `ibid.ini`(5), `ibid-setup`(1), http://ibid.omnia.za.net/

7.2 ibid-factpack

7.2.1 SYNOPSIS

`ibid-factpack` [`-s`] *factpack-file*
`ibid-factpack -r` [`-f`] *factpack-name*
`ibid-factpack -h`

7.2.2 DESCRIPTION

This utility is for adding and removing sets of packaged factoids, known as factpacks, from your Ibid's factoid database.

The default mode is factpack loading. The *factpack-file* specified is loaded into the bot's database. Should the pack contain any facts with the same name as an existing fact in the bot's database, loading will be aborted, unless the `-s` option is supplied.

Factpacks can be gzipped if the filename ends with `.gz`.

When invoked with the `-r` option, the named factpack (original import filename minus the extension) will be removed from the bot. If any of the facts contained in that pack were modified while loaded in the bot, unloading will be aborted, unless the `-f` option is supplied.

7.2.3 OPTIONS

-r, --remove	Remove the named factpack, rather than importing.
-f, --force	Force removal, if facts in the factpack were modified by users.
-s, --skip	Skip facts that clash with existing factoids, during import.
-h, --help	Show a help message and exit.

7.2.4 FACTPACKS

Factpacks are JSON-encoded text files containing a list of facts. Each fact is a tuple of two lists: fact-names, fact-values. The same substitutions are available as in normal online Factoids.

Example:

```
[
 [["Hello", "Hi"], ["<reply> Hi There", "<action> waves"]],
 [["Bye"], ["<reply> kbye $who", "<reply> Cheers"]]
]
```

7.2.5 FILES

ibid.ini Locates the database to act upon by looking for the [**databases**].**ibid** value in the bot configuration file in the current directory.

7.2.6 SEE ALSO

`ibid`(1), `ibid.ini`(5), http://ibid.omnia.za.net/

7.3 ibid-knab-import

7.3.1 SYNOPSIS

`ibid-knab-import` *knab-sa-url source* [*config-file*]

7.3.2 DESCRIPTION

This utility imports users, last seen information, factoids, and URLs from a Knab bot's database into an Ibid.

For best results, import directly into a brand new, clean Ibid install.

On import, strings are converted to Unicode, guessing UTF-8 and falling back to detection.

7.3.3 OPTIONS

knab-sa-url The SQLAlchemy URI for the Knab's database. The format is `mysql://`*user*`:`*pass*`@`*hostname*`/`*dbname*

source The name in the Ibid bot for the source that the Knab was previously connected to.

config-file If supplied, this is configuration file is used for locating the Ibid's database rather than `ibid.ini`.

7.3.4 FILES

ibid.ini Locates the database to act upon by looking for the [**databases**].**ibid** value in the bot configuration file in the current directory.

7.3.5 SEE ALSO

`ibid`(1), `ibid.ini`(5), `ibid-setup`(1), http://ibid.omnia.za.net/, http://knab.omnia.za.net/

7.4 ibid-memgraph

7.4.1 SYNOPSIS

`ibid-memgraph` [*options...*] *logfile*
`ibid-memgraph -h`

7.4.2 DESCRIPTION

This utility is for graphing memory usage from an Ibid bot configured to log such usage.

Matplotlib is required for graphing.

7.4.3 OPTIONS

-o FILE, --output=FILE Output to *FILE* instead of displaying interactive graph GUI. *FILE* can be any format supported by Matplotlib, detected by the file extension.

-d DPI, --dpi=DPI When outputting in raster formats, use *DPI* output DPI.

-h, --help Show a help message and exit.

7.4.4 FILES

logfile A log file generated by loading the **memory** plugin into Ibid, which will periodically log memory usage. It can be gzip compressed, if the filename ends in `.gz`.

7.4.5 SEE ALSO

`ibid`(1), `ibid-objgraph`(1), http://ibid.omnia.za.net/

7.5 ibid-objgraph

7.5.1 SYNOPSIS

`ibid-objgraph` [*options...*] *logfile type...*
`ibid-objgraph` [*options...*] `-e` *TIME logfile*
`ibid-objgraph -h`

7.5.2 DESCRIPTION

This utility is for graphing object-type usage from an Ibid bot configured to log such usage.

Matplotlib is required for graphing.

7.5.3 OPTIONS

-e TIME, --examine=TIME Examine the object usage at time *TIME*, and print a sorted list of type statistics at that time. This function can be useful in determining which types to graph, when chasing down a detected leak.

-o FILE, --output=FILE Output to *FILE* instead of displaying interactive graph GUI. *FILE* can be any format supported by Matplotlib, detected by the file extension.

-d DPI, --dpi=DPI When outputting in raster formats, use *DPI* output DPI.

-h, --help Show a help message and exit.

7.5.4 FILES

logfile A log file generated by loading the **memory** plugin into Ibid, which will periodically log object usage. It can be gzip compressed, if the filename ends in `.gz`.

7.5.5 SEE ALSO

`ibid`(1), `ibid-memgraph`(1), http://ibid.omnia.za.net/

7.6 ibid-pb-client

7.6.1 SYNOPSIS

`ibid-pb-client` [*options*...] `message` *message*
`ibid-pb-client` [*options*...] `plugin` *feature method* [*parameter*...]
`ibid-pb-client -h`

7.6.2 DESCRIPTION

This utility is for passing events to a running Ibid bot, or executing RPC-exposed functions remotely.

It communicates with the **pb** source on the Ibid.

message is a text message as could be sent to the bot by an IM source. The message is processed normally by the bot.

feature is the name of the feature to invoke exposed method *method* on, directly. *parameter*s are passed directly to the method. They can be specified positionally or by key, using the same syntax as Python: *key=value*. They may be encoded in JSON, if not valid JSON they will be treated as strings.

The output is a JSON-encoded response.

7.6.3 OPTIONS

-s SERVER, --server=SERVER Connect to the Ibid running on *SERVER*, by default it connects to *localhost*.

-p PORT, --port=PORT Connect to the **pb** source running on port *PORT*, by default 8789.

-h, --help Show a help message and exit.

7.6.4 SEE ALSO

`ibid`(1), http://ibid.omnia.za.net/

7.7 ibid-plugin

7.7.1 SYNOPSIS

`ibid-plugin` [*options*...] [*plugin*[−]|*plugin*.*Processor*[−]...]

7.7.2 DESCRIPTION

This utility is for testing Ibid plugins without the full bot environment. This means testing can be performed offline and without loading all the available plugins.

This should be run in a configured Ibid bot directory.

All the listed plugins and Processors will be loaded on start-up. Naming a plugin loads the complete plugin. Suffixing a − to the name, ignores that plugin or Processor instead of loading it.

7.7.3 OPTIONS

-c, --configured	Load all configured plugins, instead of only the core and requested plugins.
-o, --only	Don't load the Ibid core plugins, only the plugins requested. Note that without the **core** plugin to pre- and post-process events, most other plugins won't function correctly.
-p, --public	By default, **ibid-plugin** emulates a private conversation with the bot. With this option, the conversation is considered to be public and the bot will have to be addressed to provoke a response.
-v, --verbose	Increase verbosity. The final form of each *Event* object will be displayed before any responses.
-h, --help	Show a help message and exit.

7.7.4 FILES

ibid.ini Locates the database to act upon by looking for the [**databases**].**ibid** value in the bot configuration file in the current directory.

7.7.5 BUGS

ibid-plugin doesn't emulate a complete Ibid environment, and will ignore all of the following:

- Delayed and periodically executed functions.
- Messages to alternate sources.
- Messages directly dispatched, rather than added to responses.
- Permissions. All permissions are granted to the user.

7.7.6 SEE ALSO

`ibid`(1), `ibid.ini`(5), `ibid-setup`(1), http://ibid.omnia.za.net/

7.8 ibid-setup

7.8.1 SYNOPSIS

ibid-setup

7.8.2 DESCRIPTION

This program sets up everything that a new Ibid bot needs before it can run. It asks a series of questions about the new bot, and writes out a basic configuration file - `ibid.ini`(5) - to the current directory. It also creates a database for the bot, by default a SQLite database in the current directory.

This should be run in the directory which will become the new Ibid bot's base.

Should there be an existing `ibid.ini` in the current directory, it will be used, and the only questions asked will be for adding an administrative user. These can safely be skipped with a `^C`.

7.8.3 FILES

ibid.ini The Ibid bot's configuration file, will be created if it doesn't exist.

7.8.4 SEE ALSO

`ibid`(1), `ibid.ini`(5), http://ibid.omnia.za.net/

7.9 ibid

7.9.1 SYNOPSIS

`ibid` [*config-file*]

7.9.2 DESCRIPTION

This runs an Ibid bot in the foreground.

There should there be an existing `ibid.ini` (created by `ibid-setup`(1)) in the current directory or one should be provided on the command line.

Where possible, you should run `twistd -n ibid` instead of this program, as otherwise some classes of errors go unreported. See **BUGS**.

7.9.3 BUGS

Exceptions in twisted callbacks can go unnoticed in this program. That has no harmful effects, but the developers may miss out on some good bug reports.

7.9.4 FILES

ibid.ini The Ibid bot's configuration file, will be created if it doesn't exist.

7.9.5 SEE ALSO

`ibid.ini`(5), `ibid-setup`(1), `twistd`(1), http://ibid.omnia.za.net/

7.10 ibid.ini

7.10.1 NAME

7.10.2 DESCRIPTION

`ibid.ini` contains all the configuration for an Ibid bot.

A complete description of the contents of this file is out of the scope of this manpage. For more details see the Ibid HTML documentation: http://ibid.omnia.za.net/docs/

Lines beginning with `#` are considered to be comments and ignored. To use a `#` symbol in an option (e.g. an IRC channel name), quote the option with double-quotes, e.g. `channels = "#ibid",`

This file will be written to by the bot when configuration settings are altered online. It can also be edited manually and a running bot told to `reload config`. Manual edits and comments will be preserved when the bot modifies its own configuration, provided that they have not been edited since bot start-up or the last config reload.

7.10.3 SECTIONS

auth

Settings related to permissions and authentication. Permissions listed in **auth**.**permissions** are granted to all users unless revoked by source or account.

sources

Sources are Ibid connections to an IM service. They range from IRC networks to the bot's built-in HTTP server.

Each source is configured in a section named after the source. The source name will define the driver that the source should use, unless a `type` option is provided.

Sources can be disabled by setting `disabled = True`.

plugins

Plugin configuration. Each plugin is configured within a section named after the plugin.

cachedir The directory that temporary files (such as downloaded data), useful to be the bot but expendable, is stored in.

autoload If `True`, all plugins not explicitly ignored will be loaded. (Note that some plugins mark themselves as non-auto-loadable). Defaults to `True`.

load The list of plugins (or **plugin**.**Processor**s) to load.

noload The list of plugins (or **plugin**.**Processor**s) to ignore and not load.

core.names The names that the bot should respond to.

core.ignore Nicks that the bot should completely ignore (e.g. other bots).

7.10.4 EXAMPLE

```
botname = joebot
logging = logging.ini

[auth]
    methods = password,
    timeout = 300
    permissions = +factoid, +karma, +sendmemo, +recvmemo, +feeds, +publicresponse

[sources]
    [[telnet]]
    [[timer]]
    [[http]]
        url = http://joebot.example.com
    [[smtp]]
    [[pb]]
    [[atrum]]
        channels = "#ibid",
        nick = $botname
        type = irc
        auth = hostmask, nickserv
        server = irc.atrum.org

[plugins]
    cachedir = /tmp/ibid
    [[core]]
        names = $botname, bot, ant
        ignore = ,

[databases]
    ibid = sqlite:///ibid.db
```

7.10.5 FILES

logging.ini A standard Python **logging.config** configuration file describing loggers, handlers, and formatters for log messages. See http://docs.python.org/library/logging.html

7.10.6 SEE ALSO

`ibid`(1), http://ibid.omnia.za.net/

CHAPTER 8

API Reference

Note that the API reference is still incomplete.

8.1 `ibid.compat` – Python Version Compatibility

This module provides compatibility for older Python versions back to 2.4, allowing the use of some newer features.

The following modules and functions are available, and should be imported from *`ibid.compat`* rather than elsewhere.

8.1.1 Modules

`email_utils`
Standard Python `email.utils`.

Functions for parsing and formatting e-Mail headers.

`hashlib`
Standard Python `hashlib`.

Cryptographic hash functions.

On Python 2.4 it won't support the SHA-2 functions: `hashlib.sha224()`, `hashlib.sha384()` and `hashlib.sha512()` – these will all return `'Not Supported'`.

`json`
Standard Python `json`, using SimpleJSON on older versions.

JSON serialisation and parsing library.

`ElementTree`
Standard Python `xml.etree.cElementTree`, using ElementTree on older versions.

8.1.2 Classes

class `ibid.compat.`**`defaultdict`**([*default_factory*[, ...]])
Standard Python `collections.defaultdict`.

Returns a dict-like-object where all unset values contain the value returned by *default_factory()*.

8.1.3 Functions

ibid.compat.**all**(*iterable*)
Standard Python *all()*.

Return `True` if every item in *iterable* is `True`.

ibid.compat.**any**(*iterable*)
Standard Python *any()*.

Return `True` if any single item in *iterable* is `True`.

ibid.compat.**strptime**(*date_string*, *format*)
Standard Python `datetime.datetime.strptime()`.

Return a `datetime` corrosponding to *date_string*, according to *format*.

ibid.compat.**factorial**(*x*)
Standard Python `math.factorial()`.

Return the factorial of *x*.

8.2 ibid.config – Configuration Files

This module handles Ibid's configuration file and provides helper functions for accessing configuration from sources and plugins.

8.2.1 Helper Functions

class ibid.config.**Option**(*name*, *description*, *default=None*)
A unicode string configuration item.

Parameters

- **name** – The configuration key
- **description** – A human-readable description for the configuration item.
- **default** – The default value, if not specified in the configuration file.

If created as an attribute to a *ibid.plugins.Processor* or `ibid.source.IbidSourceFactory`, it will retrieve the value of `plugins`.*plugin.name* or `sources`.*source.name*.

This is also the base class for other Options.

Example:

```
class Secret(Processor):
    password = Option('password', 'Secret Password', 's3cr3t')
```

Assuming that processor is in the `secret` plugin, the configuration item could be provided as:

```
[plugins]
  [[secret]]
    password = blue
```

class ibid.config.**BoolOption**(*name*, *description*, *default=None*)
A boolean configuration item.

Usage is identical to *Option*.

class ibid.config.**IntOption**(*name*, *description*, *default=None*)
A integer configuration item.

Usage is identical to *Option*.

class ibid.config.**FloatOption**(*name*, *description*, *default=None*)
A floating-point configuration item.

Usage is identical to *Option*.

class ibid.config.**ListOption**(*name*, *description*, *default=None*)
A list configuration item. Values will be unicode strings.

Usage is identical to *Option*.

class ibid.config.**DictOption**(*name*, *description*, *default=None*)
A dictionary configuration item. Keys and values will be unicode strings.

Usage is identical to *Option*.

8.2.2 Core Functions

ibid.config.**FileConfig**(*filename*)
Parses *filename* and returns a configuration tree.

8.3 ibid.core – Ibid Core

This module contains Ibid's startup code, plugin, source, config, and DB loading as well as the Event dispatcher.

8.3.1 Dispatcher

ibid.core.**process**(*event*, *log*)
This function takes *event* and passes it to the *process()* function in each processor, in order of increasing *priority*. Messages are logged on the logger *log*.

After each *Processor*, any unclean SQLAlchemy sessions are committed and exceptions logged.

class ibid.core.**Dispatcher**
The Ibid *Event* dispatcher.

call_later(*delay*, *callable*, *oldevent*, **args*, ***kwargs*)
Run *callable* after *delay* seconds, passing it *oldevent* and **args* and **kwargs*.

Returns a twisted.internet.base.DelayedCall.

Can be used in plugins instead of blocking in sleep.

Internal Functions

ibid.core.**_process**(*event*)
The core of the dispatcher, must be called from a worker thread.

This function takes *event* and passes it to the *process()* function. Any responses attached to *event* are then dispatched to their destination sources.

ibid.core.**send**(*response*)
Dispatches *response* to the appropriate source.

ibid.core.**dispatch**(*event*)
Called by sources to dispatch *event*. Calls *_process()*, deferred to a thread, and returns the twisted.internet.defer.Deferred.

ibid.core.**delayed_call**(*callable*, *event*, **args*, ***kwargs*)
The method called by call_later(), in a thread, to call *callable*, then *_process()* on *event*.

ibid.core.**delayed_response**(*event*)
Dispatches responses from *delayed_call()*.

8.3.2 Reloader

class ibid.core.**Reloader**
The center of Ibid's bootstrap process, the reloader loads plugins and processors. They can be reloaded at any time.

run()
Boostrap Ibid and run the reactor.

reload_dispatcher()
Reload the Ibid dispatcher.

load_source(*name*[, *service*])
Load source of name *name*, setting the service parent to *service*.

load_sources([*service*])
Load all enabled sources, setting the service parents to *service*.

Sources can be disabled by setting the configuration key *service*.''disabled = True''.

unload_source(*name*)
Unload source of name *name*.

reload_source(*name*)
Re-load source of name *name*.

load_processors([*load*, *noload*, *autoload*])
Load all enabled processors, according to the rules in load_processor().

load specifies the plugins to force loading, *noload* plugins to skip loading, and *autoload* whether to load everything by default. If these parameters are not supplied or are None, they will be looked up as configuration keys in the plugins block.

load_processor(name, [noload, load, load_all=False, noload_all=False])
Load the plugin of name *name*. Individual Processors can be disabled by listing them in *noload*. If they are marked with *autoload* = False, then they are skipped unless listed in *load* or *load_all* is True.

unload_processor(name).
Unload plugin of name *name*.

reload_databases()
Reload the Databases.

reload_auth()
Reload the ibid.auth.

reload_config()
Notify all processors of a configuration reload, by calling *setup()*.

8.3.3 Databases

ibid.core.**regexp**(*pattern*, *item*)
Regular Expression function for SQLite.

ibid.core.**sqlite_creator**(*database*)
Connect to a SQLite database, with regular expression support, thanks to *regexp()*.

class ibid.core.**DatabaseManager**(*check_schema_versions=True*)
The DatabaseManager is responsible for loading databases (usually only one, 'ibid'), and is a dict of database to sqlalchemy.orm.scoping.ScopedSessions.

load(*name*)
Load the database of name *name*.

Echoing is configured by debugging.sqlalchemy_echo.

Databases are configured as sanely as possible:

•All databases are brought up in a UTF-8 mode, with UTC timezone.

•MySQL has the default engine set to InnoDB and ANSI mode enabled.

8.4 ibid.event – Events

class ibid.event.**Event**(*source*, *type*)
Events are at the core of Ibid's workings. Every join/part/message seen by a source is dispatched as an *ibid.event.Event* to all the plugins to process. Then responses are extracted from it and returned to the source.

Parameters

- **source** – The name of the source that this event relates to.
- **type** – The type of the event, a string of one of the following values:

  'message' A normal message

  'action' An action, the result of a /me or /describe

  'notice' An IRC notice

  'state' A state change, such as join, part, online, offline

Event inherits from dict so properties can be get and set either as attributes or keys.

source
The *source* string specified at creation.

type
The *type* string specified at creation.

responses
A list of responses that should be returned.

Note: Rather than appending to this directly, you should use the *addresponse()* function.

sender
The sender of the event, a dict with the following keys:

'nick' The user's nickname, as should be used for addressing him/her.

'id' The unique identifier for the user. I.e. jabber address or SILC user key hash. Used for opening a conversation with a user.

'connection' The unique identifier of connection that the user spoke on. Used for addressing the reply to the correct client.

complain
A string, that if present says the `Complain` processor should return an error message to the sender.

If set to `'notauthed'`, the complaint will be about insufficient authorisation.

If set to `'exception'`, the complaint will be about the bot not feeling very well.

processed
A boolean flag indicating that the event has been processed and other *Processors* don't need to look at it.

session
A SQLAlchemy `sqlalchemy.orm.session.Session` that can be used by a plugin for making queries.

It will be automatically committed by the dispatcher, but you are free to commit in a plugin so you can log a successful commit.

addresponse(*response*, *params={}*, *processed=True*, ***kwargs*)
Add a response to an event.

An event can contain more than one response, they'll be sent as separate messages.

Parameters

- **response** – The unicode response to add, can contain string substitutions, which will be provided by *params*.
- **params** – Parameters to substitute into *response*. Can either be a single unicode string or a dict of named substitutions.
- **processed** – Set *processed* `True` if `True`. Default: `True`.
- **source** – The source name to direct this reply to. Default: *source*.
- **target** – The user to direct this reply to. Default: *sender['connection']*.
- **address** – Boolean flag indicating if the user should be addressed when delivering this reply. Default: `True`.
- **action** – Boolean flag for whether the reply is a message or an action. Default: `False`.
- **notice** – Boolean flag for whether the reply is a message or an notice. Default: `False`.

Most commonly *addresponse()* is called with a unicode parameter for *response* and either a single substitution in *params* or multiple, named substitutions. However, you can also pass a Boolean value as *response* in which case the bot will emit a generic positive or negative response.

Examples (in public IRC):

```
event.addresponse(True)
# Sends something like u'user: Okay'
event.addresponse(False)
# Sends something like u"user: Shan't"
event.addresponse(u'Sure')
# Sends u"user: Sure"
event.addresponse(u'Jo said "%s"', message)
# Sends u'user: Jo said "hello"' if message was u'hello'
event.addresponse(u'%(key)s is %(value)s', {
```

```
            'key': u'Spiny Norman',
            'value': u'a Hedgehog',
        })
        # Sends u'user: Spiny Norman is a Hedgehog'
        event.addresponse(u'Look at me', address=False)
        # Sends u'Look at me'
        event.addresponse(u'dances', action=True)
        # Is the equivalent of '/me dances'
```

8.5 ibid.plugins – Plugin registration

Plugins are added to Ibid by placing a module inside *ibid.plugins*.

To do anything, the plugin must contain classes extending *Processor*.

class ibid.plugins.**Processor**

Base class for Ibid plugins. Processors receive *Events* and (optionally) do things with them. Plugins extend Processor to implement features. Each Processor occupies a single slot in the event processing queue, and can request specific types of events through the class attributes.

The Processor doesn't need to be instantiated, Ibid discovers all Processors defined in a plugin at load time.

usage

String: each line should be a BNF description of a function in the Processor. Leading and trailing whitespace in each line is ignored, as are empty lines.

Default: None

features

List: Strings naming each feature that this Processor is part of. Used in locating "usage" strings for on-line documentation.

The "description" string located inside a module-level *features* dict maps feature names to help strings.

Default: empty

permission

String: The name of the permission that *authorise()* methods in this Processor require.

Default: None

permissions

List of Strings: The names of permissions that *authorise()* methods in this Processor check directly (using *auth_responses()*).

Default: empty

event_types

A tuple of *Event* type strings that the Processor wishes to receive.

Default is only messages: ('message',)

addressed

Boolean flag: Whether to only receive events where the bot is addressed (i.e. private chat or addressed in a channel).

Default: True

processed

Boolean flag: Whether to receive events that are already marked as having been processed.

Default: `False`

priority
Integer: The weight of a Processor. Negative numbers put a Processor earlier in the queue, positive later.

Values in the range of -1000 to 1900 are sane, but outside of those, events will not behave normally, as pre-processing occurs between -2000 and -1000 and logging happens at 1900.

Default: 0 unless *processed* is `True`, then 1500

autoload
Boolean flag: Whether to load the plugin or not.

Default: `True`

setup(*self*)
Runs once on startup and on every configuration reload. Use it for setting up your Processor.

If you implement it, call `super()`.

shutdown(*self*)
Runs once on shutdown. Use it for cleaning up.

process(*self*, *event*)
This is the core of a Processor, where events get dispatched.

event is the *ibid.event.Event* to process.

Note: Don't override this, instead register handlers via *@handler* or *@match()*.

8.5.1 Decorators

ibid.plugins.**handler**()
Decorator that makes a method receive all events.

First parameter to the wrapped method will be the event object:

```
@handler
def handle(self, event):
    event.addresponse(u'Did you see that? I handled an event')
```

ibid.plugins.**match**(*regex*, *version='clean'*, *simple=True*)
Decorator that makes a method receive message events matching regular expression string *regex*.

The *regex* will be matched, with `re.I`, `re.UNICODE` and `re.DOTALL` modes. You should anchor both sides of it. If *simple* is true (default), the regex will be modified to match the whole string (`^` and `$` are added), a space in the regex will match any sequence of whitespace, and the following shortcuts are available for common regex fragments (which are captured as arguments):

Selector	expands to
{alpha}	[a-zA-Z]+
{any}	.+
{chunk}	\S+
{digits}	\d+
{number}	\d*\.?\d+
{url}	*url_regex()*
{word}	\w+

Using simple regexes where they are applicable can make them much more readable.

Any match groups or selectors in the regex will be passed as parameters to the decorated method, after the event object:

```
@match(r'(?:foo|bar) {chunk}')
def foo(self, event, parameter):
   event.addresponse(u'Foo: %s', parameter)
```

The above match is equivalent to this non-simple version:

```
@match(r'^(?:foo|bar)\s+(\S+)$', simple=False)
```

Optionally you can specify keyword arguments, which will get passed on to the function. The syntax is `{keyword:selector}`, where the keywords will get passed onto the decorated method:

```
@match(r'I am {nick:chunk} on {network:chunk}')
def register(self, event, network, nick):
   pass
```

The above match is equivalent to this non-simple version:

```
@match(r'^I\s+am\s+(?P<nick>\S+)\s+on\s+(?P<network>\S+)$', simple=False)
```

Note that you cannot mix positional and keyword arguments. All your selectors must either be named, or all be unnamed.

version can be set to one of:

'clean' The default, and almost always what you want. The bot name and intervening punctuation are removed from the front of the message, if the bot was addressed. Trailing punctuation and surrounding whitespace is stripped.

'raw' The message as the bot saw it.

'deaddressed' The bot name and intervening punctuation are removed from the front of the message, if the bot was addressed.

'stripped' Trailing punctuation and surrounding whitespace is stripped.

	De-address	Don't de-address
Strip	`'clean'`	`'stripped'`
Don't strip	`'deaddressed'`	`'raw'`

ibid.plugins.**authorise**(*fallthrough=True*)

Decorator that requires *Processor.permission* for the user that would trigger this method.

fallthrough sets the failure mode. If `True`, the next Procesor will be called in the hope of finding another one that'll handle it. If one is never found or *fallthrough* is `False`, an error message will be returned by `ibid.plugins.core.Complain`:

```
permission = 'awesome'

@authorise()
@match(r'^do\s+awesome\s+things$')
def method(self, event):
   event.addresponse(u'Yes sir, you are awesome!')
```

ibid.plugins.**periodic**([*interval=0*, *config_key=None*, *initial_delay=60*])

Decorator that runs the method every *interval* seconds, from timer events. The method won't be called until *initial_delay* seconds have passed since startup.

If *config_key* is set to a string, the *IntOption* of that name will be used to set `interval`. This is done in *Processor.setup()* so if you override that, be sure to call super.

8.5.2 Other Functions

ibid.plugins.**auth_responses**(*event*, *permission*)
If the event sender has the *permission* permission, return `True`.

If not, the event will be marked as having failed authorisation. If no other Processor processes the event, an error message will be returned by `ibid.plugins.core.Complain`.

This is used internally by *`@authorise()`*, but you can call it directly if you need more complex permission handling than *`@authorise()`* allows for.

When you use this, you should ensure that *permission* is listed in *`Processor.permission`* or *`Processor.permissions`*.

8.5.3 RPC

class ibid.plugins.**RPC**
All methods named with the prefix `remote_` will be exposed via Ibid's various RPC mechanisms (including the web interface).

It is common to extend both *`Processor`* and RPC in the same class. The handlers can then wrap around the `remote_` methods, to provide the same features over IM and RPC.

Note: The RPC code is still experimental and not widely used. Don't be surprised if it doesn't work.

8.6 `ibid.test` – Ibid Testing

8.6.1 End-to-end testing

class ibid.test.**PluginTestCase**
A subclass of Twisted Trial's unittest.TestCase. It sets up an environment much like a running Ibid including a clean SQLite database, and loads the specified plugins,

load
List: strings naming plugins to be loaded before running tests.

Default: empty

noload
List: strings naming plugins *not* to be loaded.

Default: empty

load_base
Boolean: whether to load a small set of base plugins (currently, just core).

Default: `True`

load_configured
Boolean: load all configured modules (excluding *`noload`*).

Default: if *`load`* is empty, `True`; otherwise, `False`.

username
String: the default username/nick in events created by the *`make_event()`* method.

Default: `u'user'`

public
Boolean: whether or not the events created by *make_event()* are public.

Default: `False`

network
Boolean: whether or not the test uses the external network. Used to skip tests in networkless environments (where the environment variable `IBID_NETWORKLESS_TEST` is defined).

Default: `False`

setUp()
If you override this method, make sure you call *PluginTestCase.setUp()*.

tearDown()
If you override this method, make sure you call *PluginTestCase.tearDown()*.

make_event(*message=None*, *type=u'message'*)
Create and return an event on the test source, from the test user, of type *type*.

responseMatches(*event*, *regex*)
Process *event* (either an event or a string to be treated as a message from the test user on the test source), and return a 3-tuple of (result, event, responses); result is a bool indicating whether the response matches *regex* (either a regex string or a compiled regex).

The other two elements give more information: *event* is the processed event; *responses* is a single matching response if result is `True`, or a list of all responses otherwise.

assertResponseMatches(*event*, *regex*)
Assert that *responseMatches()* returns true.

failIfResponseMatches(*event*, *regex*)
The opposite of *assertResponseMatches()*.

assertSucceeds(*event*)
Process *event* (either an event or a string to be treated as a message from the test user on the test source), and check that it is processed by some *Processor* and no complaint is set.

assertFails(*event*)
The opposite of *assertSucceeds()*.

8.7 `ibid.utils` – Helper functions for plugins

This module contains common functions that many Ibid plugins can use.

8.7.1 String Functions

ibid.utils.**ago**(*delta*[, *units*])
Return a string representation of *delta*, a `datetime.timedelta`.

If *units*, an integer, is specified then only that many units of time will be used.

```
>>> ago(datetime.utcnow() - datetime(1970, 1, 1))
'39 years, 6 months, 12 days, 9 hours, 59 minutes and 14 seconds'
>>> ago(datetime.utcnow() - datetime(1970, 1, 1), 2)
'39 years and 6 months'
```

ibid.utils.**format_date**(*timestamp*[, *length='datetime'*, *tolocaltime=True*])
Convert `datetime.datetime` *timestamp* to the local timezone (*timestamp* is assumed to be UTC if it has no *tzinfo*) and return a unicode string representation.

length can be one of `'datetime'`, `'date'` and `'time'`, specifying what to include in the output.

If *tolocaltime* is `False`, the time will left in the original time zone.

The format is specified in the `plugins.length` configuration subtree, as three values: `datetime_format`, `date_format` and `time_format`.

Example:

```
>>> format_date(datetime.utcnow())
u'2009-12-14 12:41:55 SAST'
```

ibid.utils.**parse_timestamp**(*timestamp*)
Parse string *timestamp*, convert it to UTC, and strip the timezone.

This function is good at parsing machine timestamps, but can't handle "human" times very well. (It uses `dateutil.parser`)

Return a naive `datetime.datetime`.

ibid.utils.**human_join**(*items*[, *separator=u', '*, *conjunction=u'and'*])
Turn iterable *items* into a unicode list in the format `a, b, c and d`.

separator separates all values except the last two, separated by *conjunction*.

Example:

```
>>> human_join(['a', 'b', 'c', 'd'])
u'a, b, c and d'
```

ibid.utils.**plural**(*count*, *singular*, *plural*)
If *count* is 1, return *singular*, otherwise *plural*.

It's recommended to use complete words for *singular* and *plural* rather than suffixes.

ibid.utils.**indefinite_article**(*phrase*)
Use heuristics to determine whether the pronunciation of *phrase* starts with a vowel or consonant (assuming it is English) and return 'an' or 'a' respectively.

ibid.utils.**decode_htmlentities**(*text*)
Return *text* with all HTML entities removed, both numeric and string-style.

ibid.utils.**file_in_path**(*program*)
Returns a boolean indicating whether the program of name *program* can be found, using the `PATH` environment variable.

Similar to `which` on the command line.

ibid.utils.**get_process_output**(*command*, *input=None*)
Runs *command*, a list of arguments where the first argument is the process to run (as in `subprocess.Popen`). The command will be fed *input* on standard input, and *`get_process_output()`* will block until the command exits.

Returns a tuple of (*output*, *error*, *code*): standard output, standard error, and exit code.

ibid.utils.**unicode_output**(*output*[, *errors='strict'*])
Decodes *output* a string, to unicode, using the character set specified in the `LANG` environment variable. *errors* has the same behaviour as the builtin `unicode()`.

Useful for parsing program output.

ibid.utils.**ibid_version**()
Return the current Ibid version or `None` if no version can be determined.

ibid.utils.**locate_resource**(*path*, *filename*)
Locate a resource shipped with Ibid. *path* is specified as a python package (e.g. `'ibid'`). *filename* is the relative path within the package (e.g. `'data/something.txt'`)

Returns the filename to the resource.

ibid.utils.**get_country_codes**()
Retrieve and decode a list of ISO-3166-1 country codes.

Returns a dict of code -> country_name. The codes are capitalised.

ibid.utils.**identity_name**(*event*, *identity*)
Refer to *identity* naturally in response to *event*.

8.7.2 URL Functions

ibid.utils.**url_regex**()
Returns a regular expression string (not a `re.RegexObject`) for matching a URL.

ibid.utils.**is_url**(*url*)
Is *url* a valid URL? (according to *url_regex()*)

ibid.utils.**iri_to_uri**(*iri*)
Convert a unicode *iri* to punycode host and UTF-8 path. This allows IRIs to be opened with `urllib`.

8.7.3 Web Service Functions

ibid.utils.**cacheable_download**(*url*, *cachefile*[, *headers*, *timeout=60*])
Useful for data files that you don't want to keep re-downloading, but do occasionally change.

url is a URL to download, to a file named *cachefile*. *cachefile* should be in the form of `pluginname/filename`. It will be stored in the configured `plugins.cachedir` and the full filename returned. Extra HTTP headers in *headers* can be supplied, if necessary.

If *cachefile* already exists, *cacheable_download()* will do an *If-Modified-Since* HTTP request. It handles HTTP-compression.

Example:

```
filename = cacheable_download(
   'http://www.iso.org/iso/country_codes/iso_3166_code_lists/iso-3166-1_decoding_table.htm',
   'lookup/iso-3166-1_decoding_table.htm')
```

ibid.utils.**generic_webservice**(*url*[, *params*, *headers*])
Request *url*, with optional dicts of parameters *params* and headers *headers*, and return the data.

ibid.utils.**json_webservice**(*url*[, *params*, *headers*])
Request *url*, with optional dicts of parameters *params* and headers *headers*, and parse as JSON.

JSONException will be raised if the returned data isn't valid JSON.

exception ibid.utils.**JSONException**(*Exception*)
Raised by *json_webservice()* if invalid JSON is returned.

8.7.4 ibid.utils.html – HTML Parsing

ibid.utils.html.**get_html_parse_tree**(*url*[, *data*, *headers*, *treetype='beautifulsoup*])
Request *url*, and return a parse-tree of type *treetype*. *data* and *headers* are optionally used in the request.

treetype can be any type supported by `html5lib`, most commonly `'etree'` or `'beautifulsoup'`.

ContentTypeException will be raised if the returned data isn't HTML.

exception ibid.utils.html.**ContentTypeException**(*Exception*)
Raised by *get_html_parse_tree()* if the content type isn't HTML.

CHAPTER 9

Changes in Ibid

9.1 Release 0.1.1 "Pimpernel" (2011-02-24)

Bug fix release, including a couple of security issues.

Several plugins that consume Web services or scrape Web sites have been updated to cope with changes since the last release.

There were no DB schema changes between 0.1.0 and 0.1.1.

9.1.1 Resolved Security Issues

Remote Execution

LP: #705860: Permissions were ignored for handlers not using `@match`. This allowed users to perform actions they were not authorised to.

However, no included plugins were exposed by this, all access-restricted handlers had match patterns.

Information Disclosure

LP: #567576: Occasionally insecure permissions on log files. When the bot spoke first (creating a new log file), the log file would be publicly readable, even if the message was sent in private.

Example: If the bot delivered a *privmsg* memo to a user at the beginning of the month, it would create the logfile with public readable permissions. If the logfile directory was published by a web server, this would make this private conversation log accessible to the public.

Resolution: Now channels must be explicitly configured to have publicly readable logs.

LP: #649383: If someone received a private message from the bot during a public meeting, the message could appear in the meeting minutes.

Example: a *privmsg* memo received during a meeting would appear in the minutes.

9.1.2 Major User Visible Changes

- New configuration option `plugins.log.public_logs`, a list of `source:channel` globs of channels to log in files with publically readable permissions.

- New configuration option `plugins.ascii.preferred_fonts`, a list of figlet fonts, the first one found is the default.
- Currency exchange now uses Yahoo instead of XE.com.

9.1.3 API Changes

- New Function: *ibid.utils.parse_timestamp()* for parsing well-formatted timestamps.
- New Function: *ibid.utils.generic_webservice()* for retrieving arbitrary data from a web-service.
- New Function: `ibid.db.get_regexp_op()` which returns a REGEXP SqlAlchemy operator for the DBMS in use.

9.1.4 All Changes

2011-02-23 Stefano Rivera <stefano@rivera.za.net>

> Remove MyLifeIsG.com support from MyLifeIsAverage Processor. The site has been down for around a year.
>
> Fixes: LP: #722675.

2011-02-22 Stefano Rivera <stefano@rivera.za.net>

> Use bounded_matches when returning search results.
>
> Fixes: LP: #722655.

2011-02-22 Stefano Rivera <stefano@rivera.za.net>

> Typo: parse_timestmap -> parse_timestamp (and remove an unnecessary import).
>
> Fixes: LP: #722682.

2011-02-22 Stefano Rivera <stefano@rivera.za.net>

> Allow addresponse() to take the param 0.
>
> Fixes: LP: #723132.

2011-02-21 Stefano Rivera <stefano@rivera.za.net>

> Added CHANGES and tool for generating changelog entries. Set version to 0.1.1.

2011-02-20 Stefano Rivera <stefano@rivera.za.net>

> Filter out empty definitions in gdefine.
>
> Fixes: LP: #719851.

2011-02-20 Stefano Rivera <stefano@rivera.za.net>

> We don't support SQLAlchemy 0.6 yet.
>
> Fixes: LP: #651992.

2011-02-20 Marco Gallotta <marco@gallotta.co.za>

> Only append .com for url's like "example".
>
> Fixes: LP: #702062.

2011-02-20 Stefano Rivera <stefano@rivera.za.net>

Use escape=# for LIKEs. Perform literal queries on all non-get Factoid operations. Return useful error if start index is too high. Substitute $arg for _% in search.

Fixes: LP: #544493.

2011-02-20 Stefano Rivera <stefano@rivera.za.net>

HTTPErrors should result in using url as title, not abandoning the grab.

Fixes: LP: #702798.

2011-02-20 Stefano Rivera <stefano@rivera.za.net>

Catch ImportErrors for packages we don't require in setup.py.

Fixes: LP: #651990.

2011-02-20 Stefano Rivera <stefano@rivera.za.net>

pysqlite is only necessary on ancient Pythons.

Fixes: LP: #708302.

2011-01-25 Stefano Rivera <stefano@rivera.za.net>

Add function get_regexp_op to ibid.db that returns a REGEXP op that works on Postgres too.

Fixes: LP: #595423.

2011-01-22 Keegan Carruthers-Smith <keegan.csmith@gmail.com>

Use correct plurality in pending memos message.

Fixes LP: #634257.

2011-01-22 Stefano Rivera <stefano@rivera.za.net>

Add parse_timestamp function to ibid.utils, use for parsing timestamps from Twitter.

Fixes LP: #702815.

2011-01-22 Stefano Rivera <stefano@rivera.za.net>

URLErrors have reasons, but there are other HTTPErrors

Fixes LP: #670855.

2011-01-21 Max Rabkin <max.rabkin@gmail.com>

Enforce permissions on non-@match handlers.

Fixes LP: #705860.

2011-01-19 Stefano Rivera <stefano@rivera.za.net>

Handle non-500 error codes from twitter.

Fixes LP: #670855.

2011-01-19 Stefano Rivera <stefano@rivera.za.net>

Strip tags from gcalc response.

Fixes LP: #702371.

2011-01-19 Max Rabkin <max.rabkin@gmail.com>

Check content_type is set before checking its value.

Fixes LP: #701900.

2011-01-19 Max Rabkin <max.rabkin@gmail.com>

Catch exceptions when polling feeds so that one broken feed doesn't stop all feeds.

Fixes LP: #578396.

2011-01-19 Max Rabkin <max.rabkin@gmail.com>

Use new OEIS API at oeis.org

Fixes LP: #700475.

2010-12-25 Stefano Rivera <stefano@rivera.za.net>

Fix for the logging open file cache: Logs in logs might not be in recent_logs.

Fixes LP: #655645.

2010-12-25 Stefano Rivera <stefano@rivera.za.net>

Support toilet fonts, correctly decode utf-8 figlet output, handle font choice edge cases.

Fixes LP: #607743.

2010-12-24 Stefano Rivera <stefano@rivera.za.net>

Follow redirects in "is it up"

Fixes LP: #599410.

2010-12-24 Stefano Rivera <stefano@rivera.za.net>

Rework nickserv auth to allow simultaneous authentications for the same nick (although Nickserv will only be bothered once).

Fixes LP: #655647.

2010-12-24 Stefano Rivera <stefano@rivera.za.net>

Use absolute imports to import SILC correctly

Fixes LP: #654202.

2010-12-20 Stefano Rivera <stefano@rivera.za.net>

Country Code list parsing: Check for ; in a line before splitting by semi-colons.

Fixes LP: #692347.

2010-12-20 Max Rabkin <max.rabkin@gmail.com>

Don't treat feeds with no messages as errors.

Fixes LP: #661187.

2010-11-08 Stefano Rivera <stefano@rivera.za.net>

Correctly handle state events that have no channel.

Fixes LP: #656349.

2010-11-07 Stefano Rivera <stefano@rivera.za.net>

Port google scrape search to get_html_parse_tree + etree. Handle superscript in gcalc.

Fixes LP: #580696.

2010-11-07 Stefano Rivera <stefano@rivera.za.net>

Put periodic lock-using code in a try...finally block.

2010-10-15 Stefano Rivera <stefano@rivera.za.net>

Support twitter's new AJAX URLs.

Fixes LP: #654535.

2010-10-15 Stefano Rivera <stefano@rivera.za.net>

Always respond to memo sending with confirmation of recipient. Allow memos to begin with "on ..." when not naming a known source.

Fixes LP: #634253.

2010-10-04 Stefano Rivera <stefano@rivera.za.net>

Disallow empty factoid names.

Fixes LP: #606065.

2010-10-05 Guy Halse

Allow bot to identify with zanet.net's NickServ.

Fixes LP: #652000.

2010-10-03 Stefano Rivera <stefano@rivera.za.net>

[SECURITY] Add a configuration glob-list of channels which should have public logs, rather than attempting to guess.

Fixes LP: #567576.

2010-09-30 Stefano Rivera <stefano@rivera.za.net>

Docs: Be clear that ibid is in Debian & Ubuntu.

2010-09-30 Stefano Rivera <stefano@rivera.za.net>

Handle 0 tweets in Twitter Atom feed parsing, correctly handle it elsewhere instead of treating it as no such twit.

Fixes LP: #646989.

2010-09-29 Max Rabkin <max.rabkin@gmail.com>

[SECURITY] Don't leak private messages to meeting logs.

Fixes LP: #649383.

2010-08-14 Stefano Rivera <stefano@rivera.za.net>

Don't try to process() events without a message in meeting.

Fixes LP: #598094.

2010-07-10 Stefano Rivera <stefano@rivera.za.net>

Correct abbreviated cross-ref format, shown up by Sphinx 1.0b1.

2010-07-04 Stefano Rivera <stefano@rivera.za.net>

Display latest tweets from retweeting-twits instead of thinking they don't exist.

Fixes LP: #554906.

2010-06-13 Michael Gorven <michael@gorven.za.net>

Fix real JID detection when more than one 'x' element is received.

2010-06-07 Stefano Rivera <stefano@rivera.za.net>

NickServ support for rizon.

2010-06-04 Stefano Rivera <stefano@rivera.za.net>

Switch from XE.com to Yahoo for currency conversions.

2010-06-04 Stefano Rivera <stefano@rivera.za.net>

Limit the size of the file-descriptor pool in log.

Fixes LP: #567571.

2010-05-12 Stefano Rivera <stefano@rivera.za.net>

Google is also a calculator.

Fixes LP: #574300.

2010-05-12 Stefano Rivera <stefano@rivera.za.net>

Use explicit lower() on each side of LIKE so factoids with arguments can be case-insensitive on Postgres.

Fixes LP: #574427.

2010-05-05 Max Rabkin <max.rabkin@gmail.com>

Escape query url in google scrape.

Fixes LP: #572308.

2010-05-05 Stefano Rivera <stefano@rivera.za.net>

Incorrect substitution in SQLite indexing warning.

2010-05-05 Stefano Rivera <stefano@rivera.za.net>

Change administrative user & identity linking syntax to be less problematically broad.

Fixes LP: #567510.

2010-04-26 Stefano Rivera <stefano@rivera.za.net>

Increase default HTTP GET size from 500 bytes to 2kiB.

Fixes LP: #563928.

2010-04-13 Stefano Rivera <stefano@rivera.za.net>

Update youtube plugin to cope with site redesign.

Fixes LP: #561684.

2010-04-13 Max Rabkin <max.rabkin@gmail.com>

Allow digits in Unicode character names.

2010-04-13 Stefano Rivera <stefano@rivera.za.net>

HTTP GET: Don't assume everything is utf-8, decode according to provided charset, fall back to utf-8 for text, and guess with chardet if either of those was wrong.

Fixes LP: #560973.

2010-04-09 Max Rabkin <max.rabkin@gmail.com>

Use unicode case-insensitive matching in factoid.

Fixes LP: #542707.

2010-03-27 Michael Gorven <michael@gorven.za.net>

Treat the Processor's first feature as the primary feature in RPC.

Fixes LP: #545168.

2010-03-27 Max Rabkin <max.rabkin@gmail.com>

Respond with unicode in bible error handlers.

Fixes LP: #544260.

2010-03-26 Marco Gallotta <marco@gallotta.co.za>

Allow trailing punctuation in tea-style addressing.

Fixes LP: #545186.

2010-03-23 Marco Gallotta <marco@gallotta.co.za>

Add username=ibid parameter to geonames API calls. Some calls now require it.

Fixes LP: #543989.

2010-03-23 Max Rabkin <max.rabkin@gmail.com>

Python 2.5 compatibility update for unicode lookup. Exception for unknown character changed in 2.6.

Fixes LP: #542593.

2010-03-10 Stefano Rivera <stefano@rivera.za.net>

Add placeholder to force ibid/static to be distributed.

9.2 Release 0.1.0 "Hazel" (2010-03-10)

- First public release.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

i

ibid.compat, 39
ibid.config, 40
ibid.core, 41
ibid.event, 43
ibid.plugins, 45
ibid.test, 48
ibid.utils, 49
ibid.utils.html, 52

Symbols

_process() (in module ibid.core), 41

A

addresponse() (ibid.event.Event method), 44
addressed (ibid.plugins.Processor attribute), 45
ago() (in module ibid.utils), 49
all() (in module ibid.compat), 40
any() (in module ibid.compat), 40
assertFails() (ibid.test.PluginTestCase method), 49
assertResponseMatches() (ibid.test.PluginTestCase method), 49
assertSucceeds() (ibid.test.PluginTestCase method), 49
auth_responses() (in module ibid.plugins), 48
authorise() (in module ibid.plugins), 47
autoload (ibid.plugins.Processor attribute), 46

B

BoolOption (class in ibid.config), 40

C

cacheable_download() (in module ibid.utils), 51
call_later() (ibid.core.Dispatcher method), 41
complain (ibid.event.Event attribute), 44
ContentTypeException, 52

D

DatabaseManager (class in ibid.core), 43
decode_htmlentities() (in module ibid.utils), 50
defaultdict (class in ibid.compat), 39
delayed_call() (in module ibid.core), 42
delayed_response() (in module ibid.core), 42
DictOption (class in ibid.config), 41
dispatch() (in module ibid.core), 41
Dispatcher (class in ibid.core), 41

E

environment variable
    IBID_NETWORKLESS_TEST, 49
Event (class in ibid.event), 43
event_types (ibid.plugins.Processor attribute), 45

F

factorial() (in module ibid.compat), 40
failIfResponseMatches() (ibid.test.PluginTestCase method), 49
features (ibid.plugins.Processor attribute), 45
file_in_path() (in module ibid.utils), 50
FileConfig() (in module ibid.config), 41
FloatOption (class in ibid.config), 41
format_date() (in module ibid.utils), 49

G

generic_webservice() (in module ibid.utils), 51
get_country_codes() (in module ibid.utils), 51
get_html_parse_tree() (in module ibid.utils.html), 52
get_process_output() (in module ibid.utils), 50

H

handler() (in module ibid.plugins), 46
human_join() (in module ibid.utils), 50

I

ibid.compat (module), 39
ibid.config (module), 40
ibid.core (module), 41
ibid.event (module), 43
ibid.plugins (module), 45
ibid.test (module), 48
ibid.utils (module), 49
ibid.utils.html (module), 52
IBID_NETWORKLESS_TEST, 49
ibid_version() (in module ibid.utils), 50
identity_name() (in module ibid.utils), 51
indefinite_article() (in module ibid.utils), 50
IntOption (class in ibid.config), 40
iri_to_uri() (in module ibid.utils), 51
is_url() (in module ibid.utils), 51

J

json_webservice() (in module ibid.utils), 51

JSONException, 51

L

ListOption (class in ibid.config), 41
load (ibid.test.PluginTestCase attribute), 48
load() (ibid.core.DatabaseManager method), 43
load_base (ibid.test.PluginTestCase attribute), 48
load_configured (ibid.test.PluginTestCase attribute), 48
load_processors() (ibid.core.Reloader method), 42
load_source() (ibid.core.Reloader method), 42
load_sources() (ibid.core.Reloader method), 42
locate_resource() (in module ibid.utils), 51

M

make_event() (ibid.test.PluginTestCase method), 49
match() (in module ibid.plugins), 46

N

network (ibid.test.PluginTestCase attribute), 49
noload (ibid.test.PluginTestCase attribute), 48

O

Option (class in ibid.config), 40

P

parse_timestamp() (in module ibid.utils), 50
periodic() (in module ibid.plugins), 47
permission (ibid.plugins.Processor attribute), 45
permissions (ibid.plugins.Processor attribute), 45
PluginTestCase (class in ibid.test), 48
plural() (in module ibid.utils), 50
priority (ibid.plugins.Processor attribute), 46
process() (ibid.plugins.Processor method), 46
process() (in module ibid.core), 41
processed (ibid.event.Event attribute), 44
processed (ibid.plugins.Processor attribute), 45
Processor (class in ibid.plugins), 45
public (ibid.test.PluginTestCase attribute), 48

R

regexp() (in module ibid.core), 43
reload_auth() (ibid.core.Reloader method), 42
reload_config() (ibid.core.Reloader method), 42
reload_databases() (ibid.core.Reloader method), 42
reload_dispatcher() (ibid.core.Reloader method), 42
reload_source() (ibid.core.Reloader method), 42
Reloader (class in ibid.core), 42
responseMatches() (ibid.test.PluginTestCase method), 49
responses (ibid.event.Event attribute), 43
RPC (class in ibid.plugins), 48
run() (ibid.core.Reloader method), 42

S

send() (in module ibid.core), 41
sender (ibid.event.Event attribute), 43
session (ibid.event.Event attribute), 44
setup() (ibid.plugins.Processor method), 46
setUp() (ibid.test.PluginTestCase method), 49
shutdown() (ibid.plugins.Processor method), 46
source (ibid.event.Event attribute), 43
sqlite_creator() (in module ibid.core), 43
strptime() (in module ibid.compat), 40

T

tearDown() (ibid.test.PluginTestCase method), 49
type (ibid.event.Event attribute), 43

U

unicode_output() (in module ibid.utils), 50
unload_source() (ibid.core.Reloader method), 42
url_regex() (in module ibid.utils), 51
usage (ibid.plugins.Processor attribute), 45
username (ibid.test.PluginTestCase attribute), 48