whois

API Documentation

March 31, 2007

Contents

1 Package whois

```
whois Python module
-------------------

Guess host location from their IP, based on whois services.

2 method are implemented:

* IP searched in a local database
* IP searched on an Internet whois service

Search results are stored in a local database to speedup further identical
searches. Use of former results can be disabled. Internet whois search can
be disabled too.

Local databases can be downloaded/updated from:
   http://software77.net/cgi-bin/ip-country/geo-ip.pl
   http://ip-to-country.webhosting.info/node/view/6

Full Internet whois queries may also be accessed.

Usage
-----

Import whois module:

 >>> import whois
 > loaded 79552 IP ranges from /usr/lib/.../whois/db/IpToCountry.csv.db

Query host country:

 >>> print whois.guessIPcountry('194.109.137.218')
 NETHERLANDS

Full Internet whois query:

 >>> Whois = whois.WhoisConsumer('194.109.137.218')
 >>> whois.WhoisRequest(Whois, whois.WhoisServer)
 <whois._whois.WhoisRequest at -0x4a62ce74>
 >>> whois.asyncore.loop()
 >>> print Whois.text

 OrgName:    RIPE Network Coordination Centre
 OrgID:      RIPE
 Address:    P.O. Box 10096
 City:       Amsterdam
 StateProv:
```

```
 PostalCode: 1001EB
 Country:    NL

 ReferralServer: whois://whois.ripe.net:43

 NetRange:   194.0.0.0 - 194.255.255.255
 CIDR:       194.0.0.0/8
 NetName:    RIPE-CBLK2
 NetHandle:  NET-194-0-0-0-1
 Parent:
 NetType:    Allocated to RIPE NCC
 NameServer: NS-PRI.RIPE.NET
 NameServer: NS3.NIC.FR
 NameServer: SUNIC.SUNET.SE
 NameServer: NS-EXT.ISC.ORG
 NameServer: SEC1.APNIC.NET
 NameServer: SEC3.APNIC.NET
 NameServer: TINNIE.ARIN.NET
 Comment:    These addresses have been further assigned to users in
 Comment:    the RIPE NCC region. Contact information can be found in
 Comment:    the RIPE database at http://www.ripe.net/whois
 RegDate:    1993-07-21
 Updated:    2005-08-03

 # ARIN WHOIS database, last updated 2007-03-24 19:10
 # Enter ? for additional hints on searching ARIN's WHOIS database.
```

Querying coordinates of a single country:

```
 >>> whois.geomap.getLongLat('SPAIN')
 (40.0, -4.0)
```

Drawing the world map with circles at given countries:

```
 >>> from pylab import show
 >>> whois.geomap.world({'CANADA': 48, 'FRANCE': 4, 'INDIA': 570})
 >>> show()
```

1.1 Modules

- **geomap**: whois Python module ————————-
  Geographical localisation of countries thanks to a longitude and latitude database.
  (Section 2, p. 6)
- **requester**: Internet whois client module.
  (Section 3, p. 8)

1.2 Functions

addDatabase(*IP*, *CountryName*)

Records a searched IP into searched IP database.
`>>> addDatabase(IP, CountryName)`
Input:
- IP = integer number as returned by IP2num
- CountryName = full country name

guessIPcountry(*IP*, *noInternet*=`False`, *recompute*=`'none'`)

Guesses the IP country using local database and eventually an Internet whois service. Local database is always queried before any Internet service. Internet services are queried if local database cannot answer the query.
`>>> CountryName = guessIPcountry(IP, noInternet=False, recompute='none')`
Input:
- IP = IP string
- noInternet = never use Internet whois service
- recompute = 'none', 'all' or 'unknown'

Output:
- CountryName = full country name, '?' if not found in databases

Meaning of the recompute values:
- 'none' = never search IP in whois databases if the IP was previously searched (searched IP database)
- 'all' = always search IP in whois databases
- 'unknown' = as 'none' but IP that was previously unknown in databases (CountryName = '?') are searched again

IP2num(*IP*)

Converts a string IP (xx.xx.xx.xx) into an integer number.
`>>> IPnum = IP2num(IP)`
Input:
- IP = IP string

Output:
- IPnum = IP equivalent integer number

loadDatabase()

Loads the database of previously searched IP.
`>>> loadDatabase()`

num2IP(*IPnum*)

Converts a string IP (xx.xx.xx.xx) into an integer number.
>>> `IP = num2IP(IPnum)`
Input:
- IPnum = IP equivalent integer number

Output:
- IP = IP string

readIPCountryFile(*verbose*=`True`)

Reads an IP/country database file.
>>> `readIPCountryFile(verbose=True)`
Input:
- verbose = display some information

saveDatabase()

Saves the searched IP database.
>>> `Data = saveDatabase()`
Output:
- Data = dictionary Data[IP] = country

NB: IP is an integer number as returned by IP2num

1.3 Variables

Name	Description
__author__	**Value:** `'jean-michel.philippe@libertysurf.fr'` *(type=**str**)*
__ModulePath__	**Value:** `'/home/jm/dev/lib/whois'` *(type=**str**)*
__url__	**Value:** `'http://philipjm.free.fr/blog/'` *(type=**str**)*
__version__	**Value:** `'0.2.0'` *(type=**str**)*
WhoisServer	**Value:** `'whois.arin.net'` *(type=**str**)*

2 Module whois.geomap

```
whois Python module
-------------------

Geographical localisation of countries thanks to a longitude and latitude database.

Querying coordinates of a single country:

>>> whois.geomap.getLongLat('SPAIN')
(40.0, -4.0)

Drawing the world map with circles at given countries:

>>> from pylab import show
>>> whois.geomap.world({'CANADA': 48, 'FRANCE': 4, 'INDIA': 570})
>>> show()
```

2.1 Functions

getLongLat(*Country*)

Returns the country longitude and latitude, 'None' if country not found.
`>>> Coords = getLongLat(Country)`
Input:
- Country = country name, *uppercase* only

Output:
- Coords = tuple (longitude, latitude) or 'None'

loadCountryLocations()

Loads the file of country longitudes and latitudes.
`>>> loadCountryLocations()`

loadCountryTranslation()

Loads the file of whois country to geos country translation.
`>>> loadCountryTranslation()`

strCoord2Num(*StrValue*)

Converts a string longitude or latitude to a float number.
>>> `Coord = strCoord2Num(StrValue)`
Input:
- Value = string coordinate as 'DD MM O', DD degrees, MM minutes, O orientation (= 'E', 'W', 'N' or 'S')

Output:
- Coord = float coordinate (degrees)

world(*CountryStats*, *smin*=`200.0`, *coeff*=`1.0`, *func*=`None`, *noticks*=`True`, *c*=`'b'`, *marker*=`'o'`, *cmap*=`None`, *norm*=`None`, *vmin*=`None`, *vmax*=`None`, *alpha*=`1.0`, *linewidths*=`None`, *faceted*=`True`, ***kwargs*)

Draws the world map with circles at country location. The circles surface is proportional to a function of input data for each country (see below).
>>> `ScatterObj = world(CountryStats, smin=200.0, coeff=1.0, func=None, noticks=True, c='b', marker='o`
Important note: saving a map as PNG significantly decreases circle surface.
Input:
- CountryStats = dictionary of country stats, CountryStats[Country] = integer
- smin = minimum circle surface (pixels, world = 2048x1024)
- coeff = circle sizing coefficient
- func = input data function
- noticks = do not draw axis ticks
- other args = see matplotlib 'scatter' function documention

Output:
- ScatterObj = the 'scatter' draw object

Example:
>>> `whois.geomap.world({'CANADA': 50, 'FRANCE': 34, 'INDIA': 67})`
The circle surface is computed this way:
S = map(func, coeff * array(CountryStats.values()) + smin)

2.2 Variables

Name	Description
__ModulePath__	**Value:** `'/home/jm/dev/lib/whois'` *(type=`str`)*

3 Module whois.requester

Internet whois client module.

Typical use:

```
>>> Whois = whois.WhoisConsumer('194.109.137.218')
>>> whois.WhoisRequest(Whois, whois.WhoisServer)
<whois._whois.WhoisRequest at -0x4a62ce74>
>>> whois.asyncore.loop()
>>> print Whois.text
[...]
```

3.1 Class WhoisConsumer

3.1.1 Methods

__init__(*self*, *host*)

Defines a consumer IP.

```
>>> consumer = WhoisConsumer(host)
```

Input:

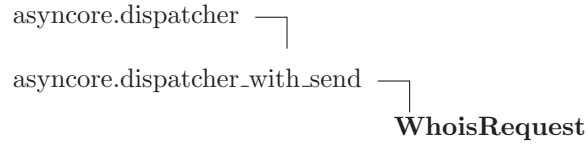
- host = host IP string

abort(*self*)

close(*self*)

feed(*self*, *text*)

3.2 Class WhoisRequest



Simple Internet whois requestor.

3.2.1 Methods

__init__(*self*, *consumer*, *host*, *port*=43)

Queries a whois Internet service.
>>> `WhoisRequest(consumer, host)`
Input:

- consumer = a WhoisConsumer object instance
- host = the whois server host IP string

Overrides: asyncore.dispatcher_with_send.__init__

handle_close(*self*)
Overrides: asyncore.dispatcher.handle_close

handle_connect(*self*)
Overrides: asyncore.dispatcher.handle_connect

handle_expt(*self*)
Overrides: asyncore.dispatcher.handle_expt

handle_read(*self*)
Overrides: asyncore.dispatcher.handle_read

Inherited from dispatcher: __getattr__, __repr__, accept, add_channel, bind, close, connect, create_socket, del_channel, handle_accept, handle_error, handle_expt_event, handle_read_event, handle_write_event, listen, log, log_info, readable, recv, set_reuse_addr, set_socket
Inherited from dispatcher_with_send: handle_write, initiate_send, send, writable

3.2.2 Class Variables

Name	Description
Inherited from dispatcher: accepting *(p. ??)*, addr *(p. ??)*, closing *(p. ??)*, connected *(p. ??)*, debug *(p. ??)*	

Index