Dataset Documentation

September 28, 2011

CONTENTS

The following is a description of a set of general classes for representing networks of entities and the communications between them in Python as well as in a database (using MongoDB). Also described are a couple of examples extending these models to represent the Enron Email corpus and conversation threads on online discussion boards.

The goal of these classes is to maximize interoperability of analytical tools across corpora by establishing a uniform terminology and format for representing their common features. Doing so should minimize the need for one to familiarize one's self with the idiosyncracies of a particular dataset before being able to perform useful analysis on it.

Aiding in this goal is Arbre, a Python framework designed to serve as an intermediary between users and the databases in which they store their datasets. Arbre contains general models like the ones described below (which have not yet been incorporated into the official program) which can be inherited from and extended for a wide variety of applications while providing the benefits of a consistent, common foundation. It also provide tools for general functions like graph generation and analysis and database operations.

Important Currently, the database creates "hot" backups on Tuesday, Thursday, and Saturday mornings (midnight). The database also *shuts down* for a period of at most 10 minutes on Sunday mornings (also at midnight) in order to do a complete backup. Be aware of this if you are planning on running a long job that needs access to the database.

CHAPTER

ONE

GENERALMODELS.PY

The generalModels.py module provides general Python classes for representing a dataset in Python and MongoDB. The classes define basic attributes belonging to a wide variety of objects. They are intended to be augmented and extended into more specific models to fit records to before inserting them into MongoDB, providing a consistent format for records in the database and allowing general analytical functions to be applied to a wide variety of datasets stored using models extended from these basic ones. Specifically, these models, along with those defined in the general models.py module in Arbre , are meant to be augmented into more specific models to which data should be fit, defined in the models.py file of an "arbreapp". Read more on Arbre and this framework for working with data here.

These do not try to be exhaustive, but instead to provide a consistent syntax for representing features of data. For example, all models inherit from the CommonDoc model, which has an optional uid field. This establishes the convention that if an object has an associated value that uniquely identifies it in a collection, this value will be stored in a field called uid. As another example, the generalModels.py file defines an Emailer class with just one field: email_address. Regardless of whether an Emailer is a single user, an account sending emails on behalf of an organization, or an automated mailing system, one can expect an object of any class which includes this class in its lineage to have an email_address field. Why go through the trouble of defining a class for one field when one can query a MongoDB database for all documents with an "email_address" field? One reason is that doing so and incorporating it in any class of objects with email addresses enforces the convention of referring to this field as email_address and not email or emailAddress or some other alternative. Therefore, when querying a database for all the entities who have the Emailer class among their _types (see next paragraph), one can expect that this field will be present. In addition, if it is later discovered in application that there are additional attributes that should be common to all Emailers, they can be added to this class as well.

The DictShield Python package, used to define all of the classes in generalModels.py, provides a framework for defining models to which data should be fit and ensuring that all instances of a model are in compliance with this format, as well as automatically providing all instances of a class with _cls and _types fields, allowing one to quickly see what class an object is instantiated from and what classes that object inherited from. In additon, an object of a class defined using this framework has a validate() function to ensure it properly fits the structure of its class before being inserted into MongoDB. This enables one to impose a consistent format on data in stored in "NoSQL" databases, whose flexibility and ability to represent objects in a more natural way than traditional RDBMS's prevents them for inforcing these strict guidelines themselves. Read more about DictShield at its GitHub repository.

The generalModels.py module defines the following classes:

class generalModels.**CommonDoc**(***values*)

An extension of the DictShield document that adds a for_mongo() method, to prepare the document for insertion into MongoDB

All of the classes in the generalModels.py module inherit the following method from this class:

uid

A field that contains a unique idenitifier of this object (if available). This may correspond to a unique identifier provided by the data set (ex. a vBulletin post_id), or a unique integer assigned to the object at a

later date. For the sake of generality, this assumed to be a String

for_mongo()
Method that returns a Python dictionary representation of this document that is ready for insertion into MongoDB.

1.1 Graph Models

class generalModels.**Edge**(**values*)
A representation of a graph edge, without respect for direction.

An Edge (or subclass of one) defines the following attributes:

ffrom
A field containing the uid of one of the Nodes to which this edge is connected. Since this model represents an *undirected* edge, this Node is interchangeable with the one whose uid is stored in the to field. The name "ffrom" is used because "from" is a reserved word in Python.

to
A field containing the uid of one of the Nodes to which this edge is connected. Since this is an undirected edge, this Node is interchangeable with the one whose uid is stored in the ffrom field.

weight
A field for storing the numerical "weight" of an edge. For the sake of generality, this assumed to be stored as a Python "float"

class generalModels.**DirectedEdge**(**values*)
Contains the same fields as an Edge, but direction is to be respected.

ffrom
Field containing the uid of the Node from which this edge originates.

to
Field containing the uid of the Node to which this edge points.

weight
A field for storing the numerical "weight" of an edge. For the sake of generality, this assumed to be stored as a Python "float"

class generalModels.**Node**(**values*)
A simple base class for representing a uniquely identifiable node in a graph.

uid
A value that uniquely identifies this Node in a collection. For the sake of generality, this class defines this field as a "String", but subclasses may define this to be an integer or other type.

class generalModels.**AdjacencyListNode**(**values*)
An extenstion of the Node class intended to be used in adjacency list graph represntations.

adjacent_vertices
A field containing a list of the :attr':*uid*s of the Nodes adjacent to this one in a graph.

class generalModels.**IncidenceListNode**(**values*)
An extension of the Node class intended to be used in an incidence list representation of a graph.

incident_edges
A field containing a list of Edges which point to, or originate from, this node.

1.2 Modeling People

class generalModels.**Person**(**values*)
A class defining the attributes of a person.

These include:

full_name

first_name

last_name

date_of_birth
Represented as a Python datetime object.

gender
Represented as a single 'M' or 'F' character

class generalModels.**Employee**(**values*)

class generalModels.**Emailer**(**values*)
Extremely basic class providing the email_address field, which one can be sure any entity (human or automated) participating in email communication will have

1.3 Messages

class generalModels.**Message**(**values*)
Model for a general representation of a message as its own entity, storing references to senders and recipients, as well as the message's text content

The Message class defines the following attributes:

uid
A integer uniquely identifying the message. For example, this field corresponds to the "post id" assigned to posts on discussion boards hosted using vBulletin or phpBB. In some cases, like in an email corpus, generation of such an integer is useful for uniquely identifying a message.

subject
The subject line (if available) of a message.

body
The text consituting the body of the message.

sender
A field containing the uid of the sender of the message. For the sake of generality, this is assumed to be a string.

recipients
A field containing the uids (assumed to be strings) of the intended recipients of the message. For the sake of generality, this field is assumed to be a list, since a message may have multiple intended recipients (as is often the case with emails). In the case of there being only one intended recipient, it is encouraged to still create a list with one entry for this field, in order to keep a consistent format. If this is the case, the presence of a to field indicates the intention for a principal recipient.

to
A field containing the uid (assumed to be a string) of the principal or sole recipient of the message, if available.

date_time
A Python `datetime` object representing the time the message was sent.

annotations
A field containing a list of instances of the `Annotation` class. This provides a uniform format for providing additional information about a message (such as indicating a `MessageQuote`)

class `generalModels.`**`Annotation`**(***values*)
Base class for specifying a kind of text annotation

annotation_type
A string indicating what kind of annotation this is. Indicates the type/format of annotation. This will typically be a "class constant".

for_mongo()
Returns a python dictionary describing the annotation and ready for insertion into MongoDB. Deletes the `_cls` and `_types` attributes provided by default by `DictShield`, since there may be many `annotations` for a given message and these attributes are not particularly helpful. The `annotation_type` field serves the purpose of these fields.

class `generalModels.`**`OriginalMessageContentAnnotation`**(***values*)
An instance of an `Annotation` used to indicate a portion of a `Message` which constitutes content original to the message (i.e. not quoted material)

start_index

An integer indicating the character index of the `Message` `body` at which the content starts

end_index
An integer indicating the character index of the `Message` `body` at which the content ends

class `generalModels.`**`MessageQuoteAnnotation`**(***values*)
An instance of an `Annotation` used to indicate a portion of a `Message` that is quoted from another message. It indicates the original author and `uid` (if available) of the quoted message.

start_index
An integer indicating the character index of the `Message` `body` at which the quote starts

end_index
An integer indicating the character index of the `Message` `body` at which the quote ends

author_id
The `uid` of the author of the quoted message. Defined as a String value for this general class, can be subclassed to include the type of unique identifier applicable to the specific corpus. For example, email authors may be assigned a unique integer id, while a `username` String uniquely identifies a poster on an online discussion board.

message_id
`uid` of the quoted `Message`

1.4 Threads

class `generalModels.`**`Thread`**(***values*)
Basic layout model for describing a thread of communication

Threads instantiated using this model are intended to be stored in their own collection, and reference a collection of `Message` s in a database. This model can be inherited from and extended for a given data set.

A Thread is decomposed into `ThreadNode`s, which correspond to `Message`s in the message collection. These nodes contain edges to other nodes in the thread (e.g. the node corresponding to a reply will have an edge to the

node corresponding to the replied-to message in the thread). A "thread" is the graph represented by these sets of nodes and edges. The graph is stored using an adjacency list representation. See `ThreadNode` for more details on how this graph is represented.

There are several advantages for storing these "thread nodes" within a `Thread` object instead of storing the messages themselves inside the thread object:

- For those uninterested in the concept of threads, messages can be accessed directly in the `Messages` collection instead of having to access them through `Thread` objects
- Only the information relative to a message's participation in a given `Thread` are stored in a corresponding `ThreadNode`. A message can be considered part of multiple threads without redundant storage of information. For example, even within the same "thread" in a message board, some posts are direct responses to one another, or discuss a somewhat different subject than other posts, and these can be grouped into their own `Thread`
- Some `ThreadNodes` may correspond to a message which is unavailable. For example, a forum post may have been deleted or an email may have been removed from a data set.

The class attributes are:

`title`
Title of the thread, corresponds to the subject line of an email thread or thread title in a message board

`uid`
A unique identifier of the thread

`originator`
This is the `uid` of the "thread starter" in the `Communicant` collection

`root_node_id`
`uid` of the `ThreadNode` that is the root of this thread

`time_created`

`thread_nodes`
A list of the `ThreadNode` objects that consititute this thread

class `generalModels.`**`ThreadNode`**(***values*)
Model for representating a message's participation in a `Thread`

`ThreadNode` objects are intended to be part of a set which constitutes a graph called a `Thread`. This graph is stored in an adjacency list representation, with each node having a `incident_edges` array attribute, storing a list of the (directed) edges corresponding to this Node.

The class attributes are:

`message_id`
The `uid` of the `Message` in the the corresponding *Messages* collection to which this `ThreadNode` corresponds.

`uid`
A unique identifier of this thread_node.

`node_depth`
Depth of this node in the thread. E.g. a node corresponding to a direct reply to the original message in the thread will have a `node_depth` value of 1, while the node corresponding to the original message will have a `node_depth` value of 0.

CHAPTER

TWO

ENRON EMAIL DATABASE

Important

Currently, the database creates "hot" backups on Tuesday, Thursday, and Saturday mornings (midnight). The database also *shuts down* for a period of at most 10 minutes on Sunday mornings (also at midnight) in order to do a complete backup. Be aware of this if you are planning on running a long job that needs access to the database.

The Enron email database is broken into three collections: *entities*, *emails*, and *threads*.

2.1 Entities

class EnronEntity

Base model for representing an entity in the Enron organization, ranging in specificity from a division like "Government Affairs" to individual employees.

The `EnronEntity` class describes the following attribute common to many `EnronUnit`s and `EnronEmployee`s:

position_nodes

An array of the `PositionNode` objects describing this entity's position in the organizational hierarchy at Enron.

This class inherits from: `generalModels.IncidenceListNode`

class EnronUnit

Model for representing organizational units within Enron like "Global Corp Affairs". This model contains all the attributes of an `EnronEntity`, but is defined in order to allow one to quickly distinguish between an organizational unit and an employee in the *entities* collection.

This class inherits from: `EnronUnit`

class EnronEmployee

Model for representing employees at Enron.

This model contains the following attributes of an `Emailer`

email_address

email_names

email_addresses

This model also contains the following attribute from the `Employee` class defined in the `generalModels.py` module:

position
A string describing this employee's highest position in the org chart

As well as:

position_id
This is an alpha numeric string associated with every position in the Enron org chart file. This field is not defined in the general `Employee` model in the `generalModels.py` module.

Since some employees in the org chart have multiple `PostionNodes` associated with them, `EnronEmployees` also have
the following field:

position_nodes
A list of all the `PostionNodes` associated with this entity in the org chart.

This class inherits from `EnronEmailer` and `generalModels.Employee`

class `Emailer`
This model contains the following attributes of an `Emailer`

email_address
A single, canonical email_address belonging to this emailer (if one exists). If an emailer has multiple email_address and it is not clear which single address should be chosen, then this field will not exist. On the other hand, the `email_addresses` field will *always* exist, even if the emailer has just a single email address.

email_names
Names by which an email participant has been referred to in the corpus. There is a many-to-one relationship of names to email participant. "[W]e made some effort to recognize different names that referred to the same person. Chiefly this was by the different names that appeared in the "From" field in a single person's sent mail items. Sometimes this led to, e.g. an executive and an assistant being assigned the same uid".

For example the string 'From: "William Williams" <bwillia5@enron.com>'may occur in one or more emails, associa
the name "William Williams" with the owner of the email address bwillia5@enron.com.

email_addresses
An array of email_addresses associated with this emailer

This class inherits from `generalModels.Emailer`

class `PositionNode`
A model associated with an `EnronEntity` describing its position in the organizational hierarchy at Enron(as charted in the orgcharts file) as a node in a graph, where an edge from this node to another indicates this node is in a higher position, and vice-versa.

A `PositionNode` is an extension of an `IncidenceListNode` as defined in the `generalModels.py` module. That is, it has a unique id, and a list of edges (specifically, of type `PositionEdge`) connecting this node to other nodes. Each `PositionNode` is associated with (and stored as a property of) an `EnronEntity`, representing that entity's position within the organization. For example, a supervisor will have a `PositionNode` with a position string of "Supervisor", and edges from his position node to the position nodes of those he supervises to indicate that he is at a higher level than these employees in the organizational hierarchy.

Also, an organizational unit like "US Natural Gas" has a `PositionNode` associated with it, representing this unit's position in Enron. This position node's `incident_edges` field contains `PositionEdges` from it to all the employees it contains, and has a `PositionEdge` pointing to it from the `PositionNode` associated with Mr. Steffes, the director of the unit.

A `PositionNode` has the following attributes:

position
A string giving the title of the position

uid
A string which serves as a unique identifier of this PositionNode. PositionEdges pointing to or from this node will reference this value.

incident_edges
An array containing the PositionEdges originating from, or pointing to this position in the organizational hierarchy.

This class inherits from generalModels.IncidenceListNode

class PositionEdge
Extension of the DirectedEdge class defined in generalModels.py module to add the "relationship" property specific to directed edges between PositionNodes in our representation of the Enron hierarchy. The to and ffrom fields in this edge store the uids of the PositionNodes this edge is connecting.

A PositionEdge has the following attributes:

to
A field storing the uid of the "child" of the PositionNode from which this edge is pointing. This indicates that the position being pointed to is below the position this edge comes from in the organizational hierarchy.

ffrom
A field storing the uid of the "parent" of the PositionNode to which this edge is pointing. This indicates that the position being pointed to is above the position this edge points to in the organizational hierarchy.

relationship
A string describing the nature of the relationship between the positions being connected. For example, a positon may "manage" another employee or entity

class EnronEmailer
Class for describing entities which have an email address that ends in @enron.com, but who are not in our orgchart and cannot be confirmed to be employees (ex. there are many mailing lists that come from an @enron.com address

This class inherits from EnronEntity and Emailer

2.2 Emails

class EnronEmail
Model for representing a unique message appearing in the enron email data set

This model inherits the following attributes from the general Message class in generalModels.py :

uid
A integer uniquely identifying the message.

subject
The subject line (if available) of a message.

body
The text consituting the body of the message.

sender
A field containing the uid of the sender of the message. For the sake of generality, this is assumed to be a string.

from
A field containing the uid of the sender of the message. The difference between this field and sender is that for emails in the MySQL database that did not indicate a sender, the "sender" field does not exist, while the "from" field exists with a null value.

recipients
A field containing the uids of the intended recipients of the message. For the sake of generality, this field is assumed to be a list, since a message may have multiple intended recipients (as is often the case with emails). In the case of there being only one intended recipient, it is encouraged to still create a list with one entry for this field, in order to keep a consistent format.

to
A list containing the uids of the people to whom this email was addressed. Even when the header info describing who the email was sent to is missing, an empty array is stored for this field.

cc
A list containing the uids of those who have been cc'd/bcc'd on this email. In the case that there were no such people, this field will simply contain an empty list.

date_time
A Python datetime object representing the time the message was sent.

annotations
A list of objects containing additional information about a message (such as indicating a MessageQuote)

In addition, the following fields unique to this dataset are defined:

is_bubble
This is a boolean field where a value of *true* indicates that this message was missing in the corpus, but was reconstructed from quotations in other messages.

header_info
This is a list of HeaderInfo objects, each of which has the uid of a sender or recipient of the message, and the corresponding information about this person that appeared in the header of the email (ex. the email address that was used)

attachments
This is a list of filenames of attachments to this message

corresponding_files
This is a list of FileInfo objects describing instances of the email appearing in the Enron dataset (ex. an email may appear both in one person's "sent" box and another's inbox)

text_chunks

A list of the TextChunk objects which comprise this email. For example, a quoted message within the email will be represented as its own TextChunk object. I think a more natural representation of the information provided by splitting a message into "text chunks" is to store the message as one, contiguous string, and indicate which ranges are quotes and other information using MessageQuoteAnnotation and SplitterTextAnnotation objects

This model inherits from generalModels.Message

class HeaderInfo
An object describing the appearance of an Emailer in an email's header, it contains the following attributes:

uid
The uid of the participating entity

email_name
The name by which the entity is referred to in the header field of the email, ex. M Scott Kuehn is reffered to both as M Scott and as Kuehn in separate emails.

email_address
The email_address with which this entity participates in this email (ex. the email was sent to or from this email address)

role
The role of this entity's participation in the email (ex. this entity was cc'd or, the email was sent from this entity). This field can take any one of the following values: *to*, *from*, *cc*, *bcc*, *sender*, *reply-to*, *to-box*, *from-box*, *bubble-to*, *bubble-from*

header_source
Describes the source of the header information. The most frequent sources of header information were that stored in an email using the Microsoft Exchange protocol (field has value "Exchange v1.0), and that stored following the general RFC format ("RFCv1.0). An additional possible value for this field is "mailbox-tagger". This value indicates that no suitable headers were found for the email, and that an attempt was made to recover the header information for the email from the folder the message was found in.

class FileInfo
This is a model for describing an instance of an email appearing in the Enron email data set. For example, the same message may be found in both the "sent" items folder of the sender and "inbox" folder of the receiver

It describes the following properties of this instance of the email:

file_path
A string indicating where in the archive of emails the file containing this instance of the message is located.

file_name
A string indicating the name of the file in which this instance of the message is located

mailbox_name
A string giving the name of the "mailbox" in which this message was found (one of the 158 released mailboxes)

sdoc_no
This is a value given to each document in the collection in the FERC dataset provided online (from which this dataset was scraped)

class OriginalMessageContentAnnotation
This class is exactly the same as the generalModels.OriginalMessageContentAnnotation class. It indicates a region of content original to this message (not quoted). It contains the following attributes:

start_index

end_index

annotation_type
The string "OriginalMessageContent" is stored here, to make the purpose of this annotation immediately apparent to someone browsing the database.

class MessageQuoteAnnotation
This class is exactly the same as the generalModels.MessageQuoteAnnotation class, with the following additional field:

relative_depth
An integer that describes the position of this text chunk in the email. Relative depth of 0 means that this is the original text of the email. A directly quoted message in the email will have a relative depth of - 1, and a message quoted by that message will have a value of -2, and so on.

class SplitterTextAnnotation
An instance of an Annotation used to indicate a portion of a Message body that is "splitter text", separating a quote from the remainder of the message.

start_index
An integer indicating the character index of the `Message` `body` at which the splitter text starts

end_index
An integer indicating the character index of the `Message` `body` at which the splitter text ends

author_id
The `uid` of the author of the quoted message.

message_id
`uid` of the quoted message (if available)

This class inherits from `generalModels.Annotation`

class TextChunk
Container object for information describing a "chunk of text" (typically a single message) present in an email. I believe that the information represented by these objects is more naturally represented as a combination of a single contiguous `body` string and corresponding `MessageQuoteAnnotation`s and `SplitterTextAnnotation`s.

This class contains the following attributes:

content
A string giving the content of this `TextChunk`

splitter_text
The text (if any) that immediately precedes this chunk of text in the message, typically used to indicate quote.

splitter_type
A code describing the "type" of splitter text that is used.

relative_depth
An integer that describes the position of this text chunk in the email. Relative depth of 0 means that this is the original text of the email. A directly quoted message in the email will have a relative depth of - 1, and a message quoted by that message will have a value of -2, and so on.

2.3 Threads

In this database, `Thread`s are stored in their own collection, titled "*threads*"

class Thread

The form of this class is the same as that defined in `generalModels.Thread` with the following additinal attributes:

dataset
A field present in Aaron's Enron database, it takes one of three values: *train*, *test* or *dev*

annotation_group_id
A number grouping this thread into a smaller subgroup for hand annotation.

This model inherits the rest of its attributes from `generalModels.Thread`

class ThreadNode
`ThreadNode`s are embedded objects stored in `Thread`s to describe a message's participation in the thread, the form of these objects in the Enron database is exactly as described in `generalModels.ThreadNode`

CHAPTER

THREE

FORUMMODELS.PY — MODELS FOR REPRESENTING FORUM POSTS AND THREADS IN PYTHON AND MONGODB

These forumModels.py module provides classes for representing messages in an online discussion forum in Python and in MongoDB. Everything defined in this module is extended from the generalModels.py module. The models outlined here are fit in the VB3Scrape module, which provides an API for scraping posts and threads from online discussion boards hosted using the vBulletin3 software.

The forumModels.py module defines the following classes:

class forumModels.**ForumPoster**(***values*)

A class describing a participant in an online discussion forum

This class defines the following attributes:

username

The login name of the forum poster. On forums hosted using vBulletin and phpBB this string can most often uniquely identify the user.

poster_id

An integer "user id" often assigned to a poster by the discussion board software. This may not always be present.

join_date

A Python datetime object giving when the user created his or her account on the discussion board. On forums hosted using vBulletin, only the month and year of registeration tend to be provided, but some discussion boards provide more detailed information.

location

A string intended to provide the general location of a user. Often posters will provide a nonsensical value.

signature

A message included at the bottom of every post made by the user.

class forumModels.**ForumThread**(***values*)

Model that extends the general :class:'Thread' class for the purpose of representing forum threads.

The ForumThread class is intended to structure information given by forum posts on a discussion board. In addition to the attributes provided by the Thread class, the ForumThread class defines the following attributes:

In addition to providing the structure for storing discussion board forums in a database, this class includes methods for scraping the necessary information from a vBulletin 3 thread.

__init__(*url*, ***kwargs*)
This constructor may be called with either the `url` of the thread or with a dictionary containing the fields and corresponding information for a thread.

Note:

Threads created with just a `url` are "lazily initialized", meaning that any relevant data that is unavailable and needs to be scraped from the thread page is collected when asked for, meaning that the first request for such information from a `Thread` object may result in a delay while the information is fetched from the given `url`

class `forumModels.`**`ForumPost`**(***values*)
Model that extends the general `Message` class for the purpose of representing forum posts

In addition to the attributes inherited from the `Message` class, the `Post` class defines the folowing attributes:

uid
An integer uniquely identifying the post on the discussion board. Both phpBB and vBulletin give each post this unique integer id.

post_num
An integer giving the number of this post in the thread. For example, the original post in the thread will have an `post_num` value of 1, and the first reply in the thread will have a `post_num` value of 2.

class `forumModels.`**`ForumPostQuote`**(***values*)
An instance of an `Annotation` used to annotate a portion of a `ForumPost` that is quoted from another post. Inherits from the more general `MessageQuote` class, providing an `annotation_type` string that indicates what kind of annotation this is, making the purpose of this annotation immediately apparent to one who observes one in a database

CHAPTER

FOUR

VBULLETIN3 — MODELS AND METHODS FOR REPRESENTING VBULLETIN3 POSTS IN PYTHON

The vBulletin3 module defines classes and associated methods for scraping information from discussion boards hosted using the vBulletin version 3.x software, and representing them in Python. This module is designed to be used to as an interface to discussion boards like Political Forum for applications wishing to collect data from them.

The module includes

vBulletin3.**forum_login**(*url*, *username*, *password*)

A function that can be called prior to instantiating a Forum or Thread object. vBulletin3 encodes passwords using an *md5hash* function before sending them to the server if the user's browser has javascript enabled, so this function does the same using the Python md5 module.

Parameters

- **url** – The URL of a forum page from which a user can login. Typically, the supplying the URL of the discussion board indexworks well.
- **username** – Username used to log in to forum.
- **password** – Password used to log in to forum.

class models.**VB3Thread**(*url=None*, ***kwargs*)

A class defined for scraping threads on forums hosted using the vBulletin 3.x forum software and fitting them to the structure of forumModels.ForumThread model.

The only practical difference between this model and the one defined in generalModels.Thread is that this one provides methods for scraping information from threads hosted using vBulletin 3.x and populating the model accordingly.

The following attributes are inherited from the generalModels.Thread class:

title

thread_id

A unique identifier of the thread

originator

This is the unique identifier of the "thread starter" in the collection of ForumPosters

root_node_id

id of the thread_node that is the root of this thread

time_created

thread_nodes
A list of the ThreadNode objects that consititute this thread

Also, the VB3Thread class has the following attributes inherited from generalModels.ForumThread:

url
The URL at which the thread can be found.

num_posts
An integer representing the number of posts contained in the thread

forum
The name of the Forum in which this thread is located

path_to_thread
A string giving the path to the thread in the format typically found on the top of a thread page, including the name of the discussion board, the forum, any subfora, and the name of the thread. E.g. "Political Forum > Political Issues > Warfare / Military > War is bad". This allows one to query a database for all those threads which fall under the topic of "Political Issues" or specifically "Political Issues > Warefare / Military", etc.

thread_type
A string describing the type of thread this is (e.g. "Sticky", "Poll", etc.)

last_post_dt
A Python datetime object representing the time of the most recent post in the thread. Can be useful for determining whether any additional posts need to be scraped when scraping threads and storing them in a database

In addition to providing the structure for storing discussion board forums in a database, this class includes methods for scraping the necessary information from a vBulletin 3 thread.

__init__(*url*, ***kwargs*)
This constructor may be called with either the url of the thread or with a dictionary containing the fields and corresponding information for a thread.

Note:

Threads created with just a url are "lazily initialized", meaning that any relevant data that is unavailable and needs to be scraped from the thread page is collected when asked for, meaning that the first request for such information from a Thread object may result in a delay while the information is fetched from the given url