

MetaDoc Documentation (MDOC)

Bjørnar Grip Fjær

June 21, 2010

Contents

1	Using the MetaDoc client	2
1.1	Handles	2
1.2	Caching	2
1.3	Log levels	3
1.4	Customizing MetaDoc	3
1.4.1	Sending data	3
1.4.2	Receiving data	3
2	Extending MetaDoc	4
3	XML document	4
3.1	Element attributes	4
3.2	Dates	4
3.3	Special attributes	4
3.4	Client	4
3.4.1	Cleaning attributes	4
4	MetaDoc API description	5
4.1	Server API	5
4.1.1	Differences from REST	5
4.1.2	Server HTTP responses	5
5	Errors	5
5.1	Document errors	6
A	List of errors	8

1 Using the MetaDoc client

Usage of the MetaDoc client is done mainly through the use of `main.py`. `main.py` takes care of sending and retrieving data from the server, as well as caching any data that could not be sent.

When `main.py` should send data to the server, a custom function that should populate the data to be sent is called, so that each site can customize the way data is gathered on the site.

When `main.py` receives data from the server, it calls a custom function based on the data received, where each site can define what should be done with said data.

1.1 Handles

`main.py` takes handles that tell the script what information to send or retrieve to or from the server. All handles can be mixed together *except* for handles that override each other.

`main.py` takes the following handles:

- h, -help** Displays a short help message explaining the handles that may be passed to `main.py`. Overrides any other handles.
- v, -verbose** Verbose mode. Prints information about progress and information sent and received between client and server. Overridden by quiet if passed together.
- q, -quiet** Quiet mode. Prints nothing unless the program fails.
- l <log level>, -log-level=<log level>** Sets the log level for the program. See section 1.3 for more information about what is logged at different levels.
- n, -no-cache** Prevents the client from sending any cached data. For more information about caching, see section 1.2.
- e** Sends event data from client to server.
- c** Sends configuration data from client to server.
- s** Sends software data from client to server.
- u** Retrieves user data from the server.
- a** Retrieves allocation data from server.
- p** Retrieves project data from server.

1.2 Caching

The client will cache any information that is not accepted by the server, *unless* the server returns a receipt for the information that marks the information as invalid or malformed in some way, such that the information will not be accepted if resent at a later date.

Data the client sends may be marked so that it will not resend any cached data when the client is run with the same handle. This is mainly for use for full updates, such as software and configuration, where any cached data would be outdated or duplicates if sent together with a new run.

If the **-n** or **-no-cache** handles are passed, the script will ignore any cached data completely and run as if it didn't exist.

1.3 Log levels

The client has five different logging levels. The list below gives an overview of what is logged at the different levels. The higher items in the list contain everything below as well, so that with a log level set to **error** will also contain **critical** logging.

debug Debugging information, used for development and error checking.

info Information about what is happening during execution, such as items sent or recieved to/from the server.

warning Warnings occouring during execution, mainly problems that will not cause a failure but that should be addressed.

error Errors that cause partial failure of the execution, such as being unable to connect to the server.

critical Critical failures that causes the execution to halt, or errors in the program code itself.

The log level defaults to the lowest possible, so everything will be logged if nothing is set.

1.4 Customizing MetaDoc

1.4.1 Sending data

1.4.2 Recieving data

When the client recieves data from the server, it parses the data and places the parsed data in a `client.custom.abstrac.MetaInput` instance. When the data is ready, the `MetaInput` instance's `process()` function will be called. This function should process the items in `self.items` as needed on the site.

Examples of `MetaInput` classes are given i `doc/examples/custom/update*.py`

2 Extending MetaDoc

To extend MetaDoc to send more data, the following is needed. Each step is explained in more detail below.

- Definition of data to be sent in the MetaDoc DTD.
- A definition file explaining the data on the client.
- An entries file, explaining any entries allowed in the data on the client.
- A `MetaInput` or `MetaOutput` sub-class that should handle data, depending on whether data is recieved or sent to or from the server, respectively.
- Adding a handle to `main.py` that will activate the data type.
- Configuring the server to send or recieve the intended data.

3 XML document

The XML document should follow the form described in the MetaDoc DTD [1].

3.1 Element attributes

All element attributes *must* be strings. This is because the attributes must be placed in the XML document, and without knowing the way to represent the attribute as a string it is not possible to properly use it as one.

After the attributes `clean`-function is run, it will be checked that the attribute value is a **basestring** (**unicode** or **str**). If any attribute is not, the element will not be sent.

3.2 Dates

All dates in the document should be on the form specified by RFC3339 [2]. The `utils` module provides a function `date_to_rfc3339` that takes a `datetime.datetime` object and returns a string on RFC3339 form. It also provides a function `rfc3339_to_date` which will return a `datetime.datetime` object from a proper RFC3339 string, or **False** if the string is not a correct RFC3339 date.

3.3 Special attributes

The **id** attribute of elements have a special function in MetaDoc. This attribute is used to identify the object when recieving receipts from the server whether elements have been added. The attribute is *not* saved in caching to avoid duplicate **ids** when resending cached data together with new data. If you want to give elements a special identifier that should be saved, it must be called something other than **id**.

3.4 Client

3.4.1 Cleaning attributes

When an element is added as a sub-element to `metaelement.MetaElement`, a `clean` function is called for each attribute. The element defenition for sub-elements added may implement a function called `clean.<attribute name>`. This function should validate that the value of the attribute and make sure it returns the string value

of the attribute. This makes it possible to create elements by passing non-string variables, such as `datetime` objects for date fields, then converting them in the clean function.

4 MetaDoc API description

4.1 Server API

The MetaDoc server implements a REST-like API. The server defines several URLs that can be accessed from the client:

baseurl/allocations/ Retrieves a list of allocations relevant to the site

baseurl/users/ Retrieves a list of users for the site

baseurl/projects/ Retrieves a list of projects relevant to the site

baseurl/config/ Sends system configuration to server

baseurl/events/ Sends site events to the server

baseurl/software/ Sends system software to server

When sending information to the MetaDoc Server, only the information relevant to that URL is processed. Any XML data sent that is not relevant for that URL is discarded, e.g. event information sent to **/baseurl/config/** will be discarded by the server. No receipt will be returned for this data.

The server will return a MetaDoc XML document containing a `<receipt>` element, which will contain `<r_entry>` elements for each element received. The `<r_entry>` element should return a code from table 1 for each element. See section 5 for more information on errors.

4.1.1 Differences from REST

There are certain differences in the API compared to the REST specification. The MetaDoc Server API makes use of HTTP POST where HTTP PUT should be used in accordance with REST. This is due to limitations in standard Python libraries.

Because the access the MetaDoc Server API gives to the client is limited, this change does not prohibit any other functionality.

4.1.2 Server HTTP responses

The server makes use of HTTP status codes.

If the client does not send a SSL certificate, sends a certificate unknown to the server, or attempts to get information about sites not identified with the certificate, the server returns a “403 Forbidden” status code.

5 Errors

The server returns a `<receipt>` containing an `<r_entry>` for each element parsed. The `<r_entry>` has the required attributes **id** and **code**, containing the ID of the element and the error code, respectively. It may also contain an attribute **note** with a short note explaining the error if extra information is available. The `<r_entry>` tag might also contain text with a longer message, if more information is needed about the error.

5.1 Document errors

In the special case where there are problems with the document itself, such as XML errors or the document not passing DTD verification, the `<r_entry> id` attribute will be set to 0 (zero), referring to the document itself.

References

- [1] *MetaDoc Document Type Definition*, <http://www.austad.us/metadoc/MetaDoc.dtd>
- [2] *RFC3339*, <http://www.ietf.org/rfc/rfc3339.txt>

A List of errors

Table 1: Error codes recieved from server

Error code	Meaning	Extra notes
1000	No errors	
2000	Error with the XML data	
2001	Missing attribute	Missing attribute should be returned as a note.
5000	Database error	
5001	MySQL database error	Note should contain the MySQL error code, and the message the MySQL error message