MetaDoc Documentation (MDOC)

Bjørnar Grip Fjær

June 18, 2010

1 XML document

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

1.1 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.

1.2 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 `rfc3330_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.

1.3 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 **id**s 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**.

2 Extensibility

MetaDoc is made so that extending the data sent between client and server should be as easy as possible.

2.1 Client

2.1.1 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.

3 MetaDoc API description

3.1 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 `<reciept>` element, which will contain `<r_entry>` elements for each element recieved. The `<r_entry>` element should return a code from table 1 for each element. See section 4 for more information on errors.

3.1.1 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.

3.1.2 Server HTTP responses

The server makes use of HTTP status codes.

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

4 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 availible. The `<r_entry>` tag might also contain text with a longer message, if more information is needed about the error.

4.1 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), reffering to the document itself.

5 Caching

The client will cache any information that is not accepted by the server, *unless* the server returns a reciept 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.

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