Guido Engine Web API Documentation

Release 0.50

Grame

January 28, 2014

Contents:

CHAPTER

ONE

OVERVIEW

The HTTP GUIDO Engine is a web server that provides music score layout services, based on the the GUIDO Engine library and on the *GUIDO Music Notation* format. These services are availble using HTTP requests. The corresponding API is inspired by a RESTful design and is close to the C/C++ API.

The semantic of the operations is carried by the HTTP method (POST, GET, HEAD) and their scope is indicated by the URI:

- POST is used to create a new score,
- GET is used to get a score or score attributes
- HEAD is used to read the header of a server response

Typical examples of transactions with the server could be as follows:

- a client POST GMN code to a server (e.g. located at `http://guido.server.org`). When the code is correct, it gets a unique identifier back from the server (let's say 'XXX'), otherwise the server returns an error code.
- in case of success, the client can access the score and its properties with the 'GET' method and simple paths (the part of the URI to the right of the hostname) that start with the identifier previously returned by the server ('XXX' in the following examples):
  - `http://guido.server.org/XXX/` to get an image of the score
  - `http://guido.server.org/XXX/pagecount` to get the count of pages
  - `http://guido.server.org/XXX/gmn` to get the gmn code
  - etc.
- options can be used and combined in conjunction to the GET method e.g.:
  - `http://guido.server.org/XXX?page=2` to get an image of the second page of the score
  - `http://guido.server.org/XXX?page=2&format=png` to get a png image of the second page
  - etc.

1.1 GUIDO Music Notation

The GUIDO Music Notation format is a formal language for score level music representation. It is a plain-text, i.e. readable and platform independent format capable of representing all information contained in conventional musical scores. The basic GUIDO Format is very flexible and can be easily extended and adapted to capture a wide variety of musical features beyond conventional musical notation (CMN). The GUIDO design is strongly influenced by objective oriented programming to facilitate an adequate representation of musical material, from tiny motives up to complex symphonic scores.

1.2 Server options

The following options can be passed to the server on startup:

- **`--port`** portnum : sets the communication port number

  default value: 8000

- **`--daemon`** used with nohup to make this run as a daemon. Daemon-like behavior can be achieved with the options below:
  - `--root`: run the server in the root directory
  - `--closestdin`: close standard in
  - `--closestdout`: close standard out
  - `--closestderr`: close standard error
- **`--logfile`** logfile mode. Use 0 for Apache-like or 1 for XML.

  default value: 0 (Apache-like)

- **`--cachedir`** name of the cache directory

  default value: A directory named `cache` in the directory of the current executable

- **`--svgfontfile`** name of the svg font file.

  default value: A file called guido2.svg in the directory of the current executable

- **`--version`** version of the server and of guido
- **`--help`** display all of these options in a help message

Here's an example of how to make the cachedir in the file `/home/guidofriend/mychachedir`

> ./guidohttpserver –cachedir /home/guidofriend/mycachedir

Here's an example of how to run the server in daemon mode

> sudo nohup ./guidohttpserver –daemon &

CHAPTER

TWO

API

The GUIDO Engine Web API makes only use of the HTTP POST, GET and HEAD methods.

The POST method is used to create a new score and the GET method to get the corresponding score or the score attributes. Unless otherwise specified, the HTTP request entity-body is empty for the GET method or contains GMN code for the POST method. The HEAD method will return the header of a server response. Responses from the server make use of the HTTP header response code to indicate the request status, and the entity-body contains JSON code.

2.1 The POST method

The POST method is used to create a new score.

The HTTP request entity-body is expected to contain **valid GMN code** that is **URL encoded**. Many URL transfer tools do this URL encoding automatically for certain POST methods. When in doubt, use a URL encoding method like, for example, the function `urlencode` in PHP.

Response body

- **in case of success:**

  a unique identifier, formatted as JSON with 'ID' as key.

  Example: `{ "ID" : "a_unique_score_id" }`

- **in case of error:**

  An error message formatted as JSON with 'Error' as key.

  Example: `{ "Error" : "Parse error line 9." }`

Response code

- 201 ("Created") in case of success.
- 400 ("Bad request") in case of non valid GMN code.

Valid GMN code is GUIDO Music Notation code that is successfully parsed.

2.1.1 Examples

The following HTML script passes in Guido Music Notation code to a server:

```
<html>
  <body>
    <form action="http://guido.server.org" method="post">
      <input type="hidden" name="data" value="[ g e c ]" />
      <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

Here is an equivalent call using ajax:

```
$.ajax({
     url: 'http://guido.server.org',
     type: 'POST',
     data: "[ g e c ]",
 });
```

or using jQuery:

```
$.post('http://guido.server.org', "[ g e c ]");
```

or using curl:

```
curl -d "data=[ g e c ]" http://guido.server.org
```

2.1.2 POSTing files

Files can be sent to the server via POST. For example, you can upload the file `foo.gmn` with:

```
curl --data-urlencode "data@foo.gmn" http://guido.grame.fr:8000
```

2.1.3 Internals

You may think of the returned ID as a reference to a GUIDO abstract representation [GAR], that is build using `GuidoParseFile` C/C++ API. Further reference to this ID may be viewed as a reference to this GAR.

2.2 The GET method

The GET method is used to query an existing score or the GUIDO engine itself.
The HTTP request entity-body is ignored and should be empty.
The scope of the request is indicated by the path of the URI.
Replies to GET queries are formatted as JSON and presented as a values of an object which key is the score identifier.

Whenever the request involves a conversion to a graphic instance of the score, options may be used to convey the layout settings. Similarly, options may be used to convert the score to a MIDI file.

2.2.1 Browsing music pages

Requests to get information about a score.
These requests require a valid score ID.

Getting the voices count

Path *ScoreID*/`voicescount`

Response body

- **in case of success:**

  Gives the voices count of the score identified by `ScoreID` formatted as JSON with 'voicescount' as key.
  Example: `{ "ScoreID" : {"voicescount" : 4} }`

- **in case of error:**

  An error message formatted as JSON with 'Error' as key.
  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX/voicescount');
```

Internals

The `voicescount` request correspond to the `GuidoCountVoices` C/C++ API.

Getting the pages count

Path *ScoreID*/`pagescount`

Response body

- **in case of success:**

  Gives the count of pages of the score identified by `ScoreID` formatted as JSON with 'pagescount' as key.
  Example: `{ "ScoreID" : {"pagescount" : 1} }`

- **in case of error:**

  An error message formatted as JSON with 'Error' as key.
  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/linespace');
```

Internals

The `pagescount` request correspond to the `GuidoGetPageCount` C/C++ API.

Getting a score duration

Path *ScoreID*/`duration`

Response body

- **in case of success:**

  Gives the duration of the score identified by `ScoreID` formatted as JSON with 'duration' as key and expressed as a rational value.
  The rational value represents music time, where 1 is the whole note duration.
  Example: `{ "ScoreID" : {"duration" : "32/4"} }`

- **in case of error:**

  An error message formatted as JSON with 'Error' as key.
  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX/duration');
```

Internals

The `duration` request correspond to the `GuidoDuration` C/C++ API.

Getting a page at a given date

Path *ScoreID*/`pageat`

Parameters `date`: a date expressed as a rational value.

Response body

- **in case of success:**

Gives the page number of the score identified by `ScoreID` that contains the given date, formatted as JSON with ‘date’ and ‘page’ as key.

Example:

```
{ "ScoreID" : {
                  "date" : "15/2",
                  "page" : 2
         }
}
```

- **in case of error:**

  An error message formatted as JSON with ‘Error’ as key.

  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 (“Success”)
- 400 (“Bad Request”) when the score doesn’t contains the date.
- 404 (“Not Found”) in case of incorrect score ID.

Example

Using jQuery:

`$.get('http://guido.server.org/XXX/pageat?date="22/8"');`

Internals

The `pageat` request correspond to the `GuidoFindPageAt` C/C++ API.

Getting the date of a given page

Path *ScoreID*/`pagedate`

Parameters `page`: a page number (page numbers start at 1). Default page is 1.

Response body

- **in case of success:**

  Gives the date of a page of the score identified by `ScoreID` formatted as JSON with ‘page’ and ‘date’ as keys. The date is expressed as a rational value.

  Example:

```
{ "ScoreID" : {
                  "page" : 1,
                  "date" : "12/1"
         }
}
```

- **in case of error:**

  An error message formatted as JSON with ‘Error’ as key.

Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 400 ("Bad Request") when there is no such page.
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX/pagedate?page=2');
```

Internals

The `pagedate` request correspond to the `GuidoGetPageDate` C/C++ API.

Internals

The requests above correspond to the Browsing music pages C/C++ API.

2.2.2 Score drawing and page formatting

Drawing a score and getting the corresponding image depends on optionnal layout and formatting parameters.

Score layout options

Score layout options may be used whenever a conversion from the abstract representation to the graphic representation is required (see the C/C++ library documentation

Options

- **`sysDistance:`** a float value to control the distance between systems, expressed in internal units

  default value: 75

- **`sysDistribution:`** a string value to control the systems distribution. Accepted values are:
  - `auto`: automatic systems distribution
  - `always`: force systems distribution
  - `never`: prevent systems distribution

  default value: "auto"

- **`sysDistribLimit:`** a float value to control the maximum distance allowed between two systems for automatic distribution mode. Distance is relative to the height of the inner page.

  default value: 0.25 (that is: 1/4 of the page height)

- **`force:`** a float value to indicate the force value of the Space-Force function. Typical values range from 400 to 1500.

  default value: 750

- **`spring:`** a float value for the spring parameter. Typical values range from 1 to 5.

  default value: 1.1

- **`neighborhoodSpacing:`** a boolean string ("yes"|"no") to tell the engine to use the neighborhood spacing algorithm or not.

  default value: "no"

- **`optimalPageFill:`** a boolean string ("yes"|"no") to tell the engine to use the optimal page fill algorithm or not.

  default value: "yes"

Example:

Using jQuery:

```
$.get('http://guido.server.org/XXX?optimalPageFill="no"&sysDistance=100');
```

Internals

The layout options correspond to the GuidoLayoutSettings data structure.

Formatting options

Score drawing options may be used when a graphic instance of the score is required.

Options

- **`page:`** a page number

  default value: 1

- **`width:`** the drawing area width in centimeters.

  default value: 15

- **`height:`** the drawing area height in centimeters.

  default value: 5

- **`marginleft:`** the left margin in centimeters.

  default value: 1

- **`marginright:`** the right margin in centimeters.

  default value: 1

- **`margintop:`** the top margin in centimeters.

  default value: 0.5

- **`marginbottom:`** the bottom margin in centimeters.

  default value: 0.5

- **`format:`** a string to request a specific image format. Accepted values are:

  - `png`: to produce a png output
  - `jpg`: to produce a jpeg output

  - `svg`: to produce a svg output

    default value: "png"

- **`resize:`** a boolean string ("yes"|"no") to resize the page size to the music size.

  default value: "yes"

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX?width=20&height=10&format=svg');
```

Path `ScoreID`: an identifier previously retrived using the POST method

Response body

- **in case of success:**

  an image of the score which format and mime/type depends on the optional formatting parameters.

- **in case of error:**

  An error message formatted as JSON with 'Error' as key.
  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX/?page=2');
```

2.2.3 MIDI rendering

MIDI file export is available from the `midi` path and may use optional export parameters.

MIDI File options

MIDI File options may be used with the `midi` path of a score.

Options

- **`ticks:`** ticks per quarternote

  default value: 960

- **`chan:`** the MIDI channel

  default value: 1

- **`intens:`** a floating point value in the range [0.0 ... 1.0]

  default value: 0.8

- **`accent:`** a floating point value used for accents as `intens` multiplicator

  default value: 1.1
- **`marcato:`** a floating point value used for marcato as `intens` multiplicator

  default value: 1.2
- **`dur:`** a floating point value in the range [0.0 ... 1.0] used as duration factor.

  default value: 0.8
- **`stacc:`** a floating point value used for staccato as `dur` multiplicator

  default value: 0.8
- **`slur:`** a floating point value used for slurs as `dur` multiplicator

  default value: 1.0
- **`tenuto:`** a floating point value used for tenuto as `dur` multiplicator

  default value: 0.9
- **`fermata:`** a floating point value used for feramat as `dur` multiplicator

  default value: 2.0

Example:

Using jQuery:

```
$.get('http://guido.server.org/XXX/midi?chan=5&intens=0.5');
```

Internals

The MIDI options correspond to the Guido2MidiParams data structure.

Path *ScoreID*/`midi`

Response body

- **in case of success:**

  a MIDI file.
- **in case of error:**

  An error message formatted as JSON with 'Error' as key.
  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX/midi');
```

2.2.4 GUIDO mappings

For a given score and a given graphic instance of this score, the GUIDO engine provides a description of the relationship between the graphic and the time space.
It provides similar relations between the score time and the performance time (i.e. rolled to unrolled map).

Graphic to time mappings

Gives the mapping between the graphic and the time time. The mapping is given as a set of pairs of graphic rectangles and time intervals.

Note that the relations between the graphic and time space depends on *Score layout options* and *Formatting options*. Thus, and to make sense, layout and formatting parameters could be used with the mapping requests and should be the same than those used to get the graphic score.

Path

ScoreID/`pagemap` : gives the time to graphic mapping at page level.
ScoreID/`staffmap`: gives a staff time to graphic mapping. An optionnal `staff` parameter may be used to indicate the staff number (default vaue is 1).
ScoreID/`voicemap` : gives a voice time to graphic mapping. An optionnal `voice` parameter may be used to indicate the voice number (default vaue is 1).
ScoreID/`systemmap` : gives the time to graphic mapping at system level.

Response body

- **in case of success:**

  a set of pairs indicating the relation between the graphic and the time space, formatted as JSON with the path as key.

  Example:

```
{ "ScoreID" : {
        "staffmap" : [
                        {"graph": {"left": 80, "top": 0, "right": 122, "bottom": 451 },
                         "time": { "start": "0/1", "end": "1/1"} },
                        {"graph": {"left": 122, "top": 0, "right": 243, "bottom": 451 },
                         "time": { "start": "1/1", "end": "2/1"} },
                        {"graph": {"left": 243, "top": 0, "right": 312, "bottom": 451 },
                         "time": { "start": "2/1", "end": "3/1"} }
                ]
        }
}
```

- **in case of error:**

  An error message formatted as JSON with 'Error' as key.
  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX/staffmap?staff=1');
```

Time to time mapping

Gives the mapping between the score and the performance time (i.e. rolled to unrolled time) according to repeat bars and jumps (to coda, da capo, etc.).

Path *ScoreID*/`timemap`

Response body

- **in case of success:**

  a set of pairs indicating the relation between the score and the performance time, formatted as JSON with `timemap` as key.

  Example:

```
{ "ScoreID" : {
        "timemap" : [
                        {"score": {"start": "0/1", "end": "1/1"},
                         "perf":  {"start": "0/1", "end": "1/1"} },
                        {"score": {"start": "1/1", "end": "2/1"},
                          "perf": {"start": "1/1", "end": "2/1"} },
                        {"score": {"start": "2/1", "end": "3/1"},
                         "perf":  {"start": "0/1", "end": "1/1"} },
                        {"score": {"start": "3/1", "end": "4/1"},
                          "perf": {"start": "1/1", "end": "2/1"} }
                ]
        }
}
```

- **in case of error:**

  An error message formatted as JSON with 'Error' as key.
  Example: `{ "ScoreID" : {"Error" : "incorrect score ID."} }`

Response code

- 200 ("Success")
- 404 ("Not Found") in case of incorrect score ID.

Example

Using jQuery:

```
$.get('http://guido.server.org/XXX/timemap');
```

Internals

The mapping requests are based on the GUIDO Mappings as decribed in the C/C++ API.

2.2.5 Miscellaneous

Miscellaneous requests to get information about the GUIDO engine or the GUIDO server.
These requests don't require a score ID.

Getting GUIDO Engine version

Path `version`

Response body

- **in case of success:**

  a version number, formatted as JSON with 'version' as key.
  Example: `{ "version" : "1.5.0" }`

Response code

- 200 ("Success")

Example

Using jQuery:

```
$.get('http://guido.server.org/version');
```

Internals

The `version` request correspond to the `GuidoGetVersionStr` C/C++ API.

Getting GUIDO Server information

Path `server`

Response body

- **in case of success:**

  a version number of the server, formatted as JSON with 'server' as key.
  Example: `{ "server" : "1.0" }`

Response code

- 200 ("Success")

Example

Using jQuery:

```
$.get('http://guido.server.org/server');
```

Getting lines spacing

Path `linespace`

Response body

- **in case of success:**

  Gives the distance between two staff lines formatted as JSON with 'linespace' as key.

  This value is constant (= 50). It does not depend on the context and will probably never change in future versions of the library.

  Example: `{"linespace" : 50}`

Response code

- 200 ("Success")

Example

Using jQuery:

```
$.get('http://guido.server.org/linespace');
```

Internals

The `linespace` request correspond to the `GuidoGetLineSpace` C/C++ API.

Internals

The requests above correspond to the miscellaneous C/C++ API.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *search*