Grobid-quantities Documentation

Release 0.5.2

Patrice Lopez <patrice.lopez@science-miner.com>, Luca Foppia

Jan 21, 2022

Contents

Grobid-quantities is a ML-based application for identification, parsing and normalisation of any expressions of measurements (e.g. pressure, temperature, etc.) from text. This work focuses on technical and scientific articles and supports input data from raw text, PDF and XML. Extracted measurements normalised toward the International System of Units (SI).

CHAPTER 1

Introduction

Grobid-quantities is a Java application, based on Grobid (GeneRation Of BIbliographic Data), a machine learning framework for parsing and structuring raw documents such as PDF or plain text. Grobid-quantities is designed for large-scale processing tasks in batch or via a web REST API.

The machine learning engine architecture follows the cascade approach, where each model is specialised in the resolution of a specific task. The models are trained using CRF (Conditional Random Field) algorithm.

quantities are modelled using three different types:

(a) `atomic values` in case of single measurements (e.g., 10 grams),

(b) `interval` (e.g. `from 3 to 5 km`) and `range` (`100 +- 4`) for continuous values, and,

(c) `lists` of discrete values:

units are decomposed and restructured. Complementary information like unit system, type of measurement are attached by lookup in an internal lexicon.

value are parsed, supporting different representations:

(a) numeric (`2`, `1000`)

(b) alphabetic (`tw`, `thousand`),

(c) power of 10 (`1.5 x 10^-5`)

(d) date/time expressions

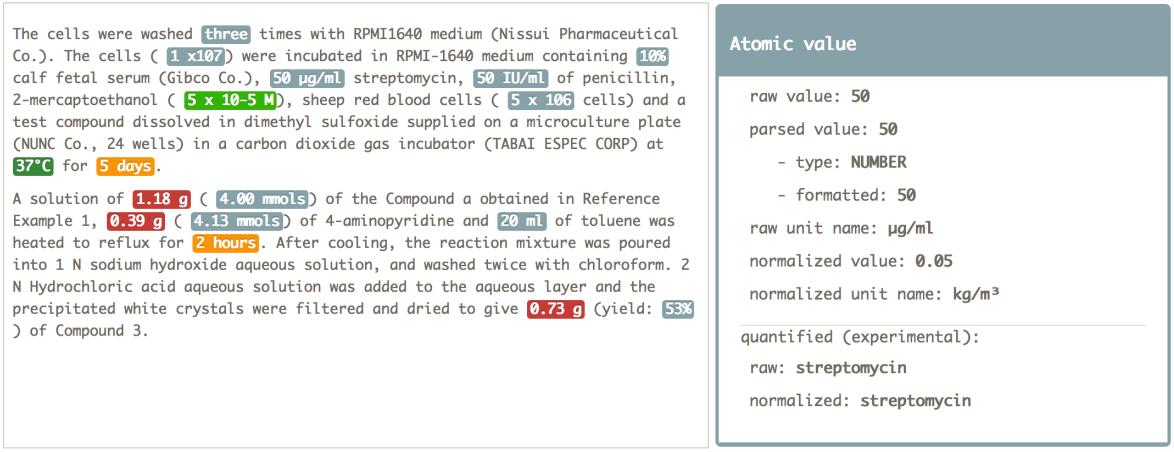
The measurements that are identified are normalised toward the International System of Units (SI) using the java library Units of measurement.

Grobid-quantities also contains a module implementing the identification of the "quantified" object/substance related to the measure. This module is currently *experimental*.

The following screenshot illustrate an example of measurement that is extracted, parsed and normalised, the quantified substance, *streptomycin* is additionally recognised:

1.1 Contacts

Contact: Patrice Lopez (<patrice.lopez@science-miner.com>), Luca Foppiano (<luca@foppiano.org>)



1.2 License

GROBID and grobid-quantities are distributed under Apache 2.0 license. The documentation is distributed under CC-0 license. The annotated data are licenced under CC 4.0 BY.

If you contribute to grobid-quantities, you agree to share your contribution following these licenses.

The References page contains citations, acknowledgement and references resources related to the project.

CHAPTER 2

Getting started

Grobid-quantities requires *JDK 1.8 or greater* and Grobid to be installed.

2.1 Install and build

2.2 Docker containers

The simplest way to run grobid-quantities is via docker containers. To run the container with the default configuration:

```
docker run --rm --init -p 8060:8060 -p 8061:8061  lfoppiano/grobid-quantities:0.7.0
```

To run the container with custom configuration, is possible by providing a configuration file with the parameter `-v` Grobid quantities repository provides already the file *resources/config/config-docker.yml* that contains the correct grobidHome and can be modified to best suits ones's needs:

```
docker run --rm --init -p 8060:8060 -p 8061:8061 -v resources/config/config-docker.
↪yml:/opt/grobid/grobid-quantities/config.yml:ro  lfoppiano/grobid-quantities:0.7.
↪0
```

2.3 Local installation

First install the latest development version of GROBID as explained by the documentation.

Grobid-quantities root directory needs to be placed as sibling sub-project inside Grobid directory:

```
cp -r grobid-quantities grobid/
```

The easier is to clone directly within the Grobid directory.

Then, build everything with:

```
cd PATH-TO-GROBID/grobid-quantities/

./gradlew copyModels
./gradlew clean build
```

You should have the directories of the models `quantities`, `units` and `values` inside `../grobid-home/models`

Run some test:

```
cd PATH-TO-GROBID/grobid-quantities

./gradlew test
```

2.4 Start and use the service

Grobid-quantities can be run with the following command:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar server resources/
↪config/config.yml
```

There is a GUI interface demo accessible at `http://localhost:8060`, and a REST API, reachable under `http://localhost:8060/service` and documented in the *Rest API Documentation*

To test the API, is possible to run a simple text using `curl`:

```
curl -X POST -F "text=I've lost two minutes." localhost:8060/service/
↪processQuantityText
```

Note: The model is designed and trained to work at *paragraph level*. The expected text input to the parser is a paragraph or a text segment of similar size, not a complete document. In case you have a long textual document, it is better either to exploit existing structures (e.g. XML/HTML <p> elements) to initially segment it into paragraphs or sentences, or to apply an automatic paragraph/sentence segmentation. Then send them separately to grobid-quantities to be processed.

2.5 Clients

The easiest way to interact with the server is to use the Python Client. It removes the complexity of dealing with the output data, and managing single or multi-thread processing. More information can be found at the Python client GitHub page.

CHAPTER 3

Rest API Documentation

This page describes the Grobid-quantities REST API.

3.1 Response description

The response is structured following a simple schema composed two attributes: *runtime* and *measurements* representing the request duration server side (in ms) and the list of extracted measurements, respectively.

The basic JSON structure is the following

```
{
   "runtime": "123
   "measurements": [
      {
          "type": ...
          "quantity*": ...
          "quantified": ...
          "pages": ...
      }
   ]
}
```

constituted by the following components:

- *quantity* represents the raw quantity
- *type* describes the measurement nature, in particular it can be `value`, `interval` or `list`. Depending on it, the property related to the quantity will change according to the table below.
- *quantified* contains the quantified object/substance in both raw and normalised expression
- *pages* provides the list of pages when processing a PDF document

Measuremen type	Quantity property name(s)	Object type
value	quantity	quantity object
interval	quantityLeast, quantityMost	quantity objects (2)
list	quantities	list of quantity objects

Note: ranges (`10+-3`) are represented directly as intervals (`7 to 13`) in JSON.

The quantity object follow the schema

```
"quantity": {
  "type": "time",
  "rawValue": "two",
  "rawUnit": {...}
  "parsedValue": {...}
  "normalizedQuantity": 120
  "normalizedUnit": {...}
  "offsetStart": 7,
  "offsetEnd": 10
}
```

which has three main objects:

- rawValue and rawUnit contains information as they appear in input
- parsedValue and parsedUnit contains parsed information (note than parsedUnit is ignored when the normalisation is successfully executed)
- normalisedQuantity and normalisedUnit contains normalisation information

3.2 Process Quantities from Text

Process text and extract and normalise measurements. The access point can be reach by:

```
POST    /service/processQuantityText
```

By processing our classical example `I've lost two minutes`:

```
curl -X POST -F "text=I've lost two minutes." localhost:8060/service/
↪processQuantityText
```

It will returns a JSON response looking like

```
{
  "runtime": 52,
  "measurements": [
      {
          "type": "value",
          "quantity": {
              "type": "time",
              "rawValue": "two",
              "rawUnit": {
                  "name": "minutes",
                  "type": "time",
                  "system": "non SI",
                  "offsetStart": 11,
                  "offsetEnd": 18
              },
              "parsedValue": {
                  "numeric": 2,
                  "structure": {
                      "type": "ALPHABETIC",
                      "formatted": "two"
                  },
                  "parsed": "two"
              },
              "normalizedQuantity": 120,
              "normalizedUnit": {
                  "name": "s",
```

(continues on next page)

(continued from previous page)


```
                    "type": "time",
                    "system": "SI base"
                },
                "offsetStart": 7,
                "offsetEnd": 10
            }
        }
    ]
}
```


Another example of a quantity of type interval looks as below:


```
{
  "runtime": 3,
  "measurements": [
      {
          "type": "interval",
          "quantityLeast": {
              "type": "time",
              "rawValue": "1",
              "rawUnit": {
                  "name": "minutes",
                  "type": "time",
                  "system": "non SI",
                  "offsetStart": 26,
                  "offsetEnd": 33
              },
              "parsedValue": {
                  "numeric": 1,
                  "structure": {
                      "type": "NUMBER",
                      "formatted": "1"
                  },
                  "parsed": "1"
              },
              "normalizedQuantity": 60,
              "normalizedUnit": {
                  "name": "s",
                  "type": "time",
                  "system": "SI base"
              },
              "offsetStart": 18,
              "offsetEnd": 19
          },
          "quantityMost": {
              "type": "time",
              "rawValue": "2",
              "rawUnit": {
                  "name": "minutes",
                  "type": "time",
                  "system": "non SI",
                  "offsetStart": 26,
                  "offsetEnd": 33
              },
              "parsedValue": {
                  "numeric": 2,
                  "structure": {
                      "type": "NUMBER",
                      "formatted": "2"
                  },
                  "parsed": "2"
```


(continues on next page)

(continued from previous page)

```
                },
                "normalizedQuantity": 120,
                "normalizedUnit": {
                    "name": "s",
                    "type": "time",
                    "system": "SI base"
                },
                "offsetStart": 24,
                "offsetEnd": 25
            }
        }
    ]
}
```

3.3 Process Quantities from PDF

Process PDF and generate annotations of measurements. The results are annotations which, by containing coordinate information, can be used to annotate directly a PDF. The access point can be reach by:

```
POST    /service/annotateQuantityPDF
```

and the file can be supplied using the `input` FormData parameter.

For instance with a `curl` query:

```
curl --form input=@./myFile.pdf localhost:8060/service/annotateQuantityPDF
```

The result follow the usual schema described above. For this case the resulting JSON contains the list of *pages* and their dimensions. Each measurement provides the coordinate for annotating each part of the entity on the PDF.

```
{
  "runtime": 32186,
  "pages": [
      {
          "page_height": 792,
          "page_width": 612
      },
      [...]
  ],
  "measurements": [
      {
          "type": "value",
          "quantity": {
              "type": "time",
              "rawValue": "many",
              "rawUnit": {
                  "name": "years",
                  "type": "time",
                  "system": "non SI",
                  "offsetStart": 2730,
                  "offsetEnd": 2735
              },
              "parsedValue": {
                  "numeric": 0,
                  "structure": {
                      "type": "ALPHABETIC",
                      "formatted": "many"
                  },
                  "parsed": "many"
```

(continues on next page)

(continued from previous page)


```
            },
            "normalizedQuantity": 0,
            "normalizedUnit": {
                "name": "s",
                "type": "time",
                "system": "SI base"
            },
            "offsetStart": 2725,
            "offsetEnd": 2729
        },
        "boundingBoxes": [
            {
                "p": 2,
                "x": 169.346,
                "y": 422.195,
                "w": 20.9665,
                "h": 8.341
            },
            {
                "p": 2,
                "x": 194.178,
                "y": 422.195,
                "w": 18.453750000000003,
                "h": 8.341
            }
        ]
    },
    [..]
  ]
}
```


3.4 Parse measures

This function takes in input a partially structured measurement and returns the normalised version.

It can be reached by

```
POST    /service/parseMeasure
```

with `raw body` with the following schema:

```
{
   "from" : "10",
   "to" : "20",
   "type" : "length",
   "unit": "km"
 }
```

It will returns something like:


```
{
  "runtime": 2120,
  "measurements": [
      {
          "type": "interval",
          "quantityLeast": {
              "type": "length",
              "rawValue": "10",
              "rawUnit": {
```


(continues on next page)

(continued from previous page)


```
                    "name": "km"
                },
                "normalizedQuantity": 10,
                "normalizedUnit": {
                    "name": "m",
                    "type": "length",
                    "system": "SI base"
                }
            },
            "quantityMost": {
                "type": "length",
                "rawValue": "20",
                "rawUnit": {
                    "name": "km"
                },
                "normalizedQuantity": 20,
                "normalizedUnit": {
                    "name": "m",
                    "type": "length",
                    "system": "SI base"
                }
            }
        }
    ]
}
```


3.5 Parse units from Text

This entry point is used to structure units.

It can be accessed at:

```
POST    /service/processUnitsText
```

The following text `cm^2W` with a `FormParam` parameter `text` will be structured in the following products:


```
[
  {
      "prefix": "c",
      "base": "m",
      "pow": "^",
      "rawTaggedValue": "<prefix>c</prefix><base>m</base>^<pow>2</pow>"
  },
  {
      "prefix": "",
      "base": "",
      "pow": "",
      "rawTaggedValue": "<base></base>"
  },
  {
      "prefix": "",
      "base": "W",
      "pow": "",
      "rawTaggedValue": "<base>W</base>"
  }
]
```

3.6 Service checks

You can check whether the service is up and running by opening the following URL:

- GET `http://yourhost:8060/service/health` will return you the result of the health check
- GET `http://yourhost:8060/service/isalive` will return true/false whether the service is up and running

3.7 Maximum parallel requests limit

This parameter allow to limit the number of parallel requests that can be send to the service. It can be modified in the configuration file the item *maxParallelRequests*. By default, the number is set to 0, which indicate to allow a number of parallel requests not higher than the number of available CPUs.

CHAPTER 4

Annotation guidelines

The first step of the annotation process is to generate training data from unlabeled documents based on the current models. The procedure is explained in details in *Training and evaluation*. GROBID will create the training data corresponding to these documents in the right TEI format and with pre-annotations. The annotation work then consists of manually checking the produced annotations and adding the missing ones. **It is very important not to modify the text content in these generated files, not adding spaces or other characters, but only adding or moving XML tags.**

When the training data has been manually corrected, move the file under the repository `resouces/dataset/${model}/corpus/` for retraining, or under `resouces/dataset/${model}/evaluation/` is the annotated data should be used for evalutation only. To see the different evaluation options, see GROBID documentation on training and evaluating.

NOTE: the exact directory where the data is picked up could be also a `final` under `corpus`. Please check the description under each model definition, below.

In this document we describe the annotation guidelines for the following models:

- *Quantities CRF model* - STABLE
- *Units CRF model* - STABLE
- *Values CRF model* - STABLE
- *Quantified objects CRF model* / substance CRF model - WIP

4.1 Quantities CRF model

The Quantities CRF models is the first model of the chain and segment text into units and values. This model pick up the training data from `resouces/dataset/${model}/corpus/final`.

Currently it supports three types of measurements: single value (or *atomic* value), continuous values in intervals (or range of values) or lists of discrete values. At the present time we do not distinguish between conjunctive and disjunctive lists.

4.1.1 Unit type vocabulary

The list of unit types (temperature, pressure, length, etc.) is controlled and based on SI definition. This controlled vocabulary contains currently 50 types. The unit types are given in the file `` `src/main/java/org/grobid/ ``

`core/utilities/UnitUtilities.java`. They are used to get the right transformation. The given names of the unit types has to be used when annotating measurement.

In the future, the list of units should however not be controlled and GROBID should support units never seen before.

For now it is admitted to annotate with `UNKNOWN` in case of doubt about the type. Examples:

- `m^2/kg` (specific surface area)
- `kD` (dissociation constant)
- `rad.m^-2`, in issue #51
- others in issue #53

4.1.2 Atomic values

We can distinguish two kinds of atomic values expressions:

Atomic values with units

Following TEI, the numeric value is identified with the element `<num>` and the unit with the element `<measure>` where the unit type is given by the attribute `@type`. The indicate of the unit name by the attribute `@unit` is optional: if present it might be used to augment the unit lexicon if not yet represented in the lexicon. A global `<measure>` element encodes the complete measurement (composed by the numeric value and the unit) associated with the measurement type given by the attribute `@type` `value`.

Example 1:

```
We monitored nutritional behaviour of amateur ski-mountaineering athletes during
↪<measure type="value"><num>4</num>
<measure type="TIME" unit="day">days</measure></measure> prior to a major␣
↪competition to compare it with official
recommendations and with the athletes' beliefs.
```

Example 2:

```
... <measure type="value"><num>20</num> <measure type="ENERGY" unit="MJ">MJ</
↪measure></measure> (<measure
type="value"><num>4,800</num> <measure type="ENERGY" unit="kcal">kcal</measure></
↪measure>) for the shorter race route...
```

A percentage (and similar expression per mil and per ten thousand) has a unit type **Unit_Type.FRACTION**:

```
<measure type="value"><num>5</num> <measure type="FRACTION" unit="%">%</measure></
↪measure> of fat mass...
```

Atomic value without unit

They correspond to a count (implicit **Unit_Type.COUNT**). The numeric value is encoded with element `<num>` and the global `<measure>` element indicating the measurement type is added.

For example:

```
consists of <measure type="value"><num>two</num></measure> different race routes
```

The implicit **Unit_Type.COUNT** type will be infer by this particular encoding. Not that this encoding is only relevant to countable quantities.

4.1.3 Intervals

An interval introduces a range of values. We can distinguish two kinds of interval expressions:

1. Bounded value

Interval defined by a lower bound value and an upper bound value:

```
team races that can last from <measure type="interval"><num atLeast="4">4</num> to␣
↪more than <num atMost="12">12</num>
<measure type="TIME" unit="hour">h</measure></measure>
```

If the unit is mentioned twice, both units are annotated, example for `3 AU  rh <~ 5 AU`:

```
<measure type="interval"><num atLeast="3">3</num> <measure type="LENGTH" unit="AU">
↪AU</measure>  r h <num atMost="5">5</num> <measure type="LENGTH" unit="AU">AU</
↪measure></measure>
```

Note that an interval can be introduced by only one boundary value:

```
A rotor shaft according to any one of the preceding claims having a diameter of at␣
↪least <measure type="interval"><num
atLeast="1">1</num><measure type="LENGTH" unit="m">m</measure></measure>

[..]sky positions lie within a <measure type="interval"><num atMost="7">7</num>
↪<measure type="ANGLE" unit="°">°</measure>
</measure> radius of other planets[..]
```

2. Base and differential value

Take the example

```
4 women and 15 men, 30± 10 years, 176±7 cm, 70±9 kg, 15±5 % of fat mass, VO2max:␣
↪50±8 ml·kg1·min1 and 21 of race A
```

after two "counts", four measurements express intervals following this form.

```
<measure type="value"><num>4</num></measure> women and <measure type="value"><num>
↪15</num></measure> men,
```

Similarly as in the previous interval case, an attribute in element `<num>`, here `@type`, characterizes the *base* value and the *differential/range* value.

```
<measure type="interval"><num type="base">30</num> ± <num type="range">10</num>
↪<measure type="TIME" unit="year">years</measure></measure>,
<measure type="interval"><num type="base">176</num> ± <num type="range">7</num>
↪<measure type="LENGTH" unit="cm">cm</measure></measure>,
<measure type="interval"><num type="base">70</num> ± <num type="range">9</num>
↪<measure type="MASS" unit="kg">kg</measure></measure>,
<measure type="interval"><num type="base">15</num> ± <num type="range">5</num>
↪<measure type="FRACTION" unit="%">%</measure></measure> of fat mass
```

If the quantity is expressed only in term of range (without base) it can be implicitly assumed that the base=0, see example ± 10 years

```
<measure type="interval">± <num type="range">10</num><measure type="TIME" unit=
↪"year">years</measure></measure>
```

If the interval has a base without a range, it's annotated with only the base (issue *#64 <https://github.com/kermitt2/grobid-quantities/issues/64>*):

```
a certain temperature interval1 around <measure type="interval"><num type="base">4
↪0</num> <measure type="TEMPERATURE" unit="°C">°C</measure></measure>
```

Notes about intervals

- Interval markers such as `more than`, `less than`, and so on, are left outside the annotation when it's possible (see issue #35). Example:

```
more than <measure type="interval"> <num atLeast="2">2</num> </measure>
```

- An interval can be bounded with quantities expressed in different unit multiples (see issue #45). For the sentence `radii between 10 µm and 1 cm` the result will be:

```
grains with radii between <measure type="interval"><num atLeast="10">10</num>
↪<measure type="LENGTH" unit="µm">µm</measure> and <num atMost="1">1</num>
↪<measure type="LENGTH" unit="cm">cm</measure></measure>
```

- The `From ... to` markers are **not necessarily introducing an interval**, example:

```
the rate was reduced from <measure type="value"><num>1.87</num></measure> to
↪<measure type="value"><num>0.82</num></measure>
```

- When interval boundaries are given alphabetically, the <num> attribute must be converted in numbers:

```
between <measure type="interval"><num atLeast="1">one</num> and <num atMost="10
↪">ten</num></measure>
```

- Rational numbers are writter with a slash (/) (issue *#66 <https://github.com/kermitt2/grobid-quantities/issues/66>*)

```
at least <measure type="interval"><num atLeast="2/3">two-thirds</num></measure>
↪ of wins for their favourite team
```

4.1.4 Lists

Lists introduce series of values. The unit can be expressed per value or for several values at the same time. A `<measure>` element encloses the whole list of values including their units:

```
<measure type="list"><measure type="ENERGY" unit="cm^-1">cm-1</measure>: <num>3440
↪</num>(br), <num>1662</num>,
<num>1632</num>, <num>1575</num>, <num>1536</num>, <num>1498</num>, <num>1411</num>
↪, <num>1370</num>, <num>1212</num>,
<num>1006</num>, <num>826</num>, <num>751</num></measure>

<measure type="list"><num>1.27</num> <measure type="LENGTH" unit="Å">Å</measure>
↪for 1H5Y, <num>1.52</num>
<measure type="LENGTH" unit="Å">Å</measure> for 1KA9, and <num>1.69</num> <measure
↪type="LENGTH" unit="Å">Å</measure>
</measure> for 1THF
```

List can be disjunctive, conjunctive, or a combination. We do not distinguish the different kinds of list at the present time:

```
batches of <measure type="list"><num>three</num> or <num>four</num></measure>
↪observations

for flexural samples the size is <measure type="list"><num>100</num> <measure type=
↪"LENGTH" unit="mm">mm</measure>
```

(continues on next page)

(continued from previous page)

```
 x <num>100</num> <measure type="LENGTH" unit="mm">mm</measure> x <num>400</num>
↪<measure type="LENGTH" unit="mm">mm
 </measure></measure>
```

If there are no intermediary values, it's an argument for deciding to annotate an element as a list, for example this ranked list (issue *#65 <https://github.com/kermitt2/grobid-quantities/issues/65>*):

```
for every lower position in the general classification the prize money was more or␣
↪less halved between
 the first <measure type="value"><num>seven</num></measure> ranked riders: from
↪<measure type="list">
 <measure type="CURRENCY" unit="euro">€</measure> <num>450,000</num> for the␣
↪winner over <measure type=
 "CURRENCY" unit="euro">€</measure> <num>200,000</num> for the runner-up to
↪<measure type="CURRENCY"
 unit="euro">€</measure> <num>100,000</num> for the third ranked rider, and so on␣
↪to
 <measure type="CURRENCY" unit="euro">€</measure> <num>11,500</num></measure> for␣
↪the rider ranked in seventh place.
```

4.1.5 Additional items

Dates

Dates are time measurements, they are thus also encoded in the training data as a complement to the other _TIME_ expressions involving time units. In TEI P5, the dates are marked with a specific element `<date>` which can be contained in an element `<measure>`. The encoding is then straightforward for atomic values (with attribute `@when`), intervals (with attribute `@from-iso` and `@to-iso` in case on min-max intervals) and lists:

```
Comet C/2013 A1 (Siding Spring) will have a close encounter with Mars on <measure␣
↪type="value"><date when=
"2014-10-19">October 19, 2014</date></measure>.

The arrival time of these particles spans a <measure type="interval"><num type=
↪"range">20</num>–<measure
type="TIME" unit="min">minute</measure> time interval centered at <date type="base
↪" when="2014-10-19T20:09">
October 19, 2014 at 20:09 TDB</date></measure>


Observations took place from <measure type="interval"><date from-iso="2014-10-19">
↪October 19, 2014</date> to
<date to-iso="2014-10-25">October 25, 2014</date></measure>.

the emergence of this sport in the <measure type="interval"><date from-iso="1980"␣
↪to-iso="1989">1980 s</date>
</measure>


Observations were performed on <measure type="list"><date when="2013-10-29">
↪October 29, 2013</date>, on <date
when="2014-01-21">Jan 21, 2014</date>, and on <date when="2014-03-11">March 11,␣
↪2014</date></measure>.
```

Time tag (and difference with Date tag)

- if only the part of a date is expressed (for example the time of a day), but we can infer the date, a complete date is implicit and the context can make it being fully quantified.

For example `20:10 UTC` will be annotated:

```
<measure type="value"><date when="2014-10-19T20:10Z">20:10 UTC</date></measure>
```

With UTC inside the annotation which is important to know exactly the "time" measure.

- for a time expression not linked to a date, like the expression of an "hour", it's appropriate to annotate with the tag `<time>`, to distinguish from the `<date>` case (see issue #48):

```
To sleep well, relax everyday at <measure type="value"><time when="21:00:00">21:00
↪</time></measure>
```

```
It consists of infusing the [...] drugs in the following order: leucovorin between
↪<measure type="interval"><time from-iso="10:30:00">10 h 30</time> and <time to-
↪iso="21:20:00">21 h 20</time></measure> [...]
```

Special cases

Frozen quantity expressions like *decade* **or** *Room temperature*

- **Room temperature** (Raumtemperatur, température ambiante, ...) is used very frequently in chemistry and related fields. It can be considered as 20 °C (293 Kelvin), although not defined in a standard manner (https://de.wikipedia.org/wiki/Raumtemperatur).

```
<measure type="value"><measure type="TEMPERATURE">Raumtemperatur</measure></
↪measure>
```

- **Decade** (issue #52)

```
over  <measure type="interval"><num atLeast="2">two</num> <measure type="TIME"␣
↪>decades</measure></measure>
```

4.1.6 Miscellaneous / Examples

Plus (+) and minus (-) signs

The + and - signs must be put **inside** the `<num>` tag. Examples:

```
a recent study [...] showed that cycling efficiency was lower (<measure type="value
↪"><num>11</num><measure type="FRACTION" unit="%">%</measure></measure>) and␣
↪energy cost of running was greater (<measure type="value"><num>+11</num><measure␣
↪type="FRACTION" unit="%">%</measure></measure>) in the master compared with␣
↪young triathletes
```

Units without values

Case where it's not annotated: When we refer to the units as such, to express something about the units, we are not using the units to quantify something with a value:

```
and r H are the geocentric and heliocentric distances in cm and AU, respectively,␣
↪and F comet and F
```

Like here for the units: `cm` and `AU`.

Case where it's annotated: We could have units expressed without values, when the value is implicit:

```
that can extend <measure type="interval"><measure type="LENGTH" unit="mm">
↪millimeters</measure></measure> or even <measure type="interval"><measure type=
↪"LENGTH" unit="cm">centimeters</measure></measure> from the cell body
```

(continues on next page)

(continued from previous page)

here the value of millimeters and centimeters is unspecified (e.g. equivalent to `several`), but we have a quantity and more precisely an interval. See issue #31

Imprecise quantifiers

When used with units, quantifers like `few`, `several`, `a couple`, `a large amount of` is annotated, and whatever quantifies even imprecisely :

```
the reference solution becomes distinct from the ballistic solution only a
↪<measure type="value"><num>couple</num> of <measure type="TIME" unit="week">weeks
↪</measure></measure> before the encounter.
```

Determiners are left outside (*a <measure type="value"><num>couple</num> of <measure type="TIME" unit="week">weeks</measure></measure>*). See issue #34

X-fold

Quantifiers like `two-fold`, `sevenfold` meaning "two times/part", "seven times/part" are annotated, to capture the full expression of quantity including this notion of "part":

```
allowing a <measure type="value"><num>sevenfold</num></measure> compaction of the␣
↪length

LUCA-HisF displays a clear <measure type="value"><num>two-fold</num></measure>␣
↪symmetry
```

Constants

Precise number (for example `c` , the speed of light in vacuum) and imprecise numbers (for example π which has an infinite number of decimals) are annotated. See issue #37

Example:

```
`decelerating from <num>5</num><measure type="VELOCITY" unit="% c">% c</measure>`
```

Exponents for powers of ten

Exponents notation might be lost in documents, for example 10 power -6 in pdf becomes `10 6`. The correct exponents are written in the attribute when there is one, 10 power -6 will be written `10^-6`. Example in interval:

```
<measure type="interval"><num atMost="10^-6">10 6</num></measure>
```

See issue #38

Numbers which seems to be only tags but are in fact quantifying

For example expressions like *at day 21* or *between day 56 and day 91*, which are really quantifying and for which range queries can be expressed.

OCR errors

OCR errors are annotated as if they were the correct sequences, since they are realistic noise. For example:

```
20 aC -> <measure type="value"><num>20</num> <measure type="TEMPERATURE" unit="°C">
↪°C</measure></measure>
2.5 ◦ -> <measure type="value"><num>2.5</num><measure type="ANGLE" unit="°">°</
↪measure></measure>
```

4.1.7 Out of scope

Only **expressions of quantities** are annotated, which can use numbers or alphabetical words.

Some numbers are also used for other stuff like markers, call-out, section number, identifiers, index, reference expressions, formula parameters, ill-encoded characters, etc. and all these cases are out of scope. See issue #36

Some sequences not annotated (not commented)

Markers, call-out, section number, numerical bullet points, identifiers, index, reference expressions, formula parameters

Examples:

Reference markers:

```
lower than those derived by Vaubaillon et al. (2014) and Moorhead et al. (2014)␣
↪computing the corresponding impact probabilities (Milani et al. 2005)
```

Figure/table titles, and other numbers who don't quantify anything:

```
Figure 1 shows the residuals of C/2013 A1's observations
[Figure 1 about here.]
Table 1 contains the orbital elements of the computed solution.
our new orbit solution (JPL solution 46)
```

Inline formulas, like:

```
a minimum point of v 2 = |v| 2 under the constraint that the particle reaches Mars,
↪ i.e., (ξ, ζ)(r, β, v) = (0, 0).
```

Some sequences to be commented out

Ill-encoded characters

Examples:

Sequence that would be usually annotated but contain encoding problems / characters in the free unicode range, like:

```

```

4.1.8 Case not yet supported

The following cases are not annotated at this stage. **The sentence when these cases occur should be put in comments** for the moment.

Sigma estimation

```
We selected the A 1 uncertainty so that its range would span from 0 au/d 2 to␣
↪twice the nominal value at 3&#x3C3;.
```

Intervals embedded in intervals

```
[..]only Mars is near enough that the orbital motion can extend a single viewing␣
↪window from 45 days to as much as 60 to 90 days.

For the wide scenario the uncertainty goes from 45 min down to 1–2 min.
```

Note: one possibility would be to only mark the external boundaries of the interval.

```
[..]only Mars is near enough that the orbital motion can extend a single viewing␣
↪window from <measure type="interval">
<num atLeast="45">45</num><measure type="TIME" unit="day">days</measure> to as␣
↪much as 60 to <num atMost="90">90</num>
<measure type="TIME" unit="day">days</measure></measure>.

For the wide scenario the uncertainty goes from <measure type="interval"><num␣
↪atLeast="45">45</num>
<measure type="TIME" unit="days">min</measure> down to 1–<num atMost="2">2</num>
↪<measure type="TIME" unit="min">min</measure></measure>.
```

List of intervals

```
No significant difference in running and total times was observed between the age␣
↪groups 25 to 34 and 35 to 44 years
```

Atomic value expressed with different values and units

```
The current Hawaii Ironman triathlon record is 8:54:202 for females
```

```
[a] male athlete was able to finish [an] ultra-marathon in a time of 19 h 44 min
```

Unit embedded in numerical value

For example `92°.5` wich would require to embed `<measure>` in `<num>` (issue #49). Or also `126 withdrawals out of 162 riders.`

Discontinuous cases

Interval quantities whith base and range multiplied as a whole by a power of ten (see issue #42):

```
A1 (Siding Spring) will pass Mars with a close approach distance of 1.35 ± 0.05 ×␣
↪10 5 km
```

or like:

```
The gas production rates, Q(CO 2 ) = (3.52> ± 0.03) × 10 26 molecules s 1
```

Other cases

```
This process takes place between t = 30[ 12 h] and t = 140[ 12 h]
```

4.1.9 Various examples

XML examples (ended with the name of the file between brackets)

```
Note that only <measure type="value"><num>47 out of the 76</num></measure>␣
↪patients experienced a first accident.[hal-00643787.training.tei.xml]

Branson Sonifier W-250D, <measure type="value"><num>2 x 2</num> <measure type="TIME
↪" unit="min">min</measure></measure> in <measure type="value"><num>15</num>
↪<measure type="TIME" unit="s">sec</measure></measure> intervals. [hal-00924047.
↪training.tei.xml]

ranging from <measure type="interval"><num atLeast="1.14">1.14</num> <measure type=
↪"LENGTH" unit="Å">Å</measure> to <num atMost="1.43">1.43</num> <measure type=
↪"LENGTH" unit="Å">Å</measure></measure>.[hal-00924047.training.tei.xml]

the emission maxima shifted from <measure type="value"><num>345</num> <measure␣
↪type="LENGTH" unit="nm">nm</measure></measure> to <measure type="value"><num>325
↪</num> <measure type="LENGTH" unit="nm">nm</measure></measure> [hal-00924047.
↪training.tei.xml]

[...] (<measure type="list"><num>72</num> for compressive test and <num>72</num></
↪measure> for flexural test) [hal-00962359]

Patients with SS have a <measure type="interval"><num atLeast="20">20</num>-<num␣
↪atMost="40">40 fold</num></measure> increased risk of developing lymphoma [hal-
↪00987664]
```

4.2 Units CRF model

The Units model is used to parse chunks of text recognised as units from the Quantity CRF model. The data handled as unit is modelled based on the SI representation of unit, which models any unit as a triple: prefix, base and pow. For example `m^2` can be represented as `[-, m, 2]`. Complex units are naturally supported: `m^3/kg*s` can be rewritten as `[(-, m, 3)(k, g, -1)(-, s, -1)]`.

There are 3 labels for this model:

- `<prefix>` represent the unit prefix, for example `k` for `kilo`, `G` for `giga` and so on and so forth. The definition of these prefix is in the prefix dictionary under `resources/lexicon/${lang}/prefix.txt`.
- `<base>` carry out the unit base information. For NON-SI units the entire unit should be contained as `<base>`.
- `<pow>` contains information about the exponents and divisions. It's used to correctly assign exponent values and to revert the sign to blocks in the denominator.

The training data follows a simple structure, see an example on how `cm` is represented:

```
<units>
  <unit><prefix>c</prefix><base>m</base></unit>
</units>
```

General principles:

- in complex units the division mark should be labelled as `<pow>`, for example:

```
<unit><base>m</base><pow>/</pow><base>s</base></unit>
```

- the indication of a power, like `^` should be taken out of the `<pow>` label, for example `m/s^4` should be annotated as:

```
<unit><base>m</base><pow>/</pow><base>s</base>^<pow>4</pow></unit>
```

- non-SI units are added in the `<base>` as anticipated above:

```
<unit><base>miles</base></unit>
```

- denominator written with subsequent `division marks`, like `cal/kg/m/days`, which means `cal/(kg*m*days)`, all the `/` should be annotated:

```
<unit><base>cal</base><pow>/</pow><base>kg</base><pow>/</pow><base>m</base>
↪<pow>/</pow><base>days</base></unit>
```

- in general `numerator / den1 den2 den3 is equivalent` to `numerator/(den1*den2*den3)`. Groups by parenthesis are not supported.

4.3 Values CRF model

The Values model is used to parse chunks of text recognised as values from the Quantity CRF model.

The Values model uses the following labels:

- `<number>` identify numeric values
- `<exp>` identify the exponent in the base-E expressions (written as e and corresponding to the constant `Euler's number` 2.71828)
- `<base>` identify the base of the base-N representation
- `<pow>` identify the exponent for base-N or base-E expressions
- `<time>` represent date and time expressions
- `<alpha>` identify values expressed in alphabetic

This labels are combined to recognise this type of values:

- numeric value:

```
<value><number>2</number></value>
```

- alphabetic value:

```
<value><alpha>two</number></value>
```

- power with base-10 expression, discussed in #7. Example:

```
<value><base>10</base>^<pow>-5</pow></value>
```

  The `<base>` and `<pow>` should contains only values. Additional markers (like `x`, `^`, `x`) should be left out. The `<base>` label should contains expression of the "base", usually `10` but could be set to different values.

  For example:

```
<value><number>1</number> × <base>10</base> <pow>7</pow></value>
```

- Euler's number based expressions, discussed in: #8. Example:

```
<value><number>1.2</number>e^<exp>-5</exp></value>
```

  `e` or `E` should be left out, because is a constant and does not have variability.

- time and date expressions, discussed in #12

```
<value><time>2001 August</time></value>
```

4.4 Quantified objects CRF model

Currently work in progress The quantified object (or substance) is the object for which the measurement is expressed. For example *A mixture of 10kg of silicon nitride powder*. The object is the *silicon nitride powder* which is attached to the measurement of 10 with unit Kg.

Cf. issue #19

The quantified object model currently rely on a simplistic approach that involve the dependency parsing of the sentence and the identification of the head in the phrase with some heuristic.

In this section we discuss the guidelines for annotating training data used for a CRF model to replace the current implementation.

In the training data are identified the measurement (and their type) and the quantified object.

For example:

```
<p>A mixture of 10kg of silicon nitride powder.</p>
```

can be annotated as:

```
<p>A mixtured of <measure type="value" ptr="#1235324324321">10kg</measure> of
↪<quantifiedObject id="1235324324321">silicon nitride powder</quantified Object>.
↪</p>
```

The quantified object is identified by its ID and linked to the measure via the attribute *ptr="#ID"*.

NOTE This implementation allows the linking of objects directly attached on the left or right of the measurement, for the time being far entities are not supported.

How to annotate?

Annotating the quantifiedObject is a complicated task, because it requires a clear definition to avoid misunderstanding. Firstly the question each annotator should ask is "What is being measured?".

CHAPTER 5

Training and evaluation

As the rest of GROBID, the training data is encoded following the TEI P5. See *Annotation guidelines* for detailed explanations and examples.

5.1 Generation of training data

Training data generation works the same as in GROBID, with executable name `createTrainingQuantities`, for example:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar trainingGeneration -
↪dIn ~/grobid/grobid-quantities/src/test/resources/ -dOut ~/test/ resources/
↪config/config.yml
```

by default the generation of training data is performed on the specified directory. The argument `-r` will process all files within the subdirectories, recursively.

Help can be invoked with

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar trainingGeneration --
↪help
```

The input directory can contain PDF (.pdf, scientific articles only), XML/TEI (.xml or .tei, for patents and scientific articles) and text files (.txt).

5.2 Training

This section describes quickly how to run the training with grobid-quantities. The models will be saved under `grobid-home/models/quantities` and `grobid-home/models/units` respectively, make sure those directories exist.

To run the training, assuming the current directory is the `grobid-quantities` directory:

```
cd PATH-TO-GROBID/grobid/grobid-quantities
```

The training can be invoked using gradle or via the the *java* command using the *dropwizard* command line, which offers more options.

5.2.1 Quantities CRF model

The trainer uses all the available training data from `resouces/dataset/quantities/corpus/final`.

Via Gradle :

```
./gradlew train_quantities
```

Via command line:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar training -m quantities␣
↪-a train resources/config/config.yml
```

5.2.2 Units CRF model

The trainer uses all the available training data from `resouces/dataset/units/corpus`. The training instance is the unit itself (a `<unit></unit>` entry in the XML training file)

Via Gradle :

```
./gradlew train_units
```

Via command line:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar training -m units -a␣
↪train resources/config/config.yml
```

5.2.3 Values CRF model

The trainer uses all the available training data from `resouces/dataset/values/corpus`. The training instance is the unit itself (a `<value></value>` entry in the XML training file)

Via Gradle :

```
./gradlew train_values
```

Via command line:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar training -m values -a␣
↪train resources/config/config.yml
```

5.2.4 Quantified objects CRF model

This model is not yet enabled at the moment because it's still WIP

The trainer uses all the available training data from `resouces/dataset/quantifiedObject/corpus`. The training instance is the paragraph itself (a `<p></p>` entry in the XML training file)

Via Gradle :

```
./gradlew train_quantifiedObject
```

Via command line:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar training -m␣
↪quantifiedObject -a train resources/config/config.yml
```

the training and save the model produced in the latest iteration. 1000 iterations are largely enough.

5.3 Evaluation

Grobid-quantities can be evaluated using a random *80/20* ratio, using an *holdout* set or using *n-fold cross-validation*.

The 80/20 evaluation uses random 80% training data in `resouces/dataset/MODEL_NAME/corpus` and the remaining 20% for evaluation.

The command to run the 80/20 evaluation is:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar training -m model_name␣
↪-a train_eval resources/config/config.yml
```

The holdout evaluation train the model and run the evaluation against a fixed set of training data. The training data is taken from `resouces/dataset/MODEL_NAME/corpus` and the evaluation data is taken from `resouces/dataset/MODEL_NAME/evaluation`.

The command to run the holdout evauation is:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar training -m model_name␣
↪-a holdout resources/config/config.yml
```

The N-fold cross-validation perform the training and evaluation N times, partition the training data in N sets and using each set for evaluation while training with the rest. More detailed explanation here. The evaluation will then give the average scores over these n models (against test set) and for the best model which will be saved.

The command to run the n-fold cross-validation with N folds is the following:

```
java -jar build/libs/grobid-quantities-{version}-onejar.jar training -m model_name␣
↪-a nfold --fold-count N resources/config/config.yml
```

CHAPTER 6

References

6.1 How to cite

If you want to cite this work, please simply refer to the github project, with optionally the Software Heritage project-level permanent identifier:

```
grobid-quantities (2015-2021) <https://github.com/kermitt2/grobid-quantities>,␣
↪swh:1:dir:dbf9ee55889563779a09b16f9c451165ba62b6d7
```

Here's a BibTeX entry using the Software Heritage project-level permanent identifier:

```
@misc{grobid-quantities,
    title = {grobid-quantities},
    howpublished = {\url{https://github.com/kermitt2/grobid-quantities}},
    publisher = {GitHub},
    year = {2015--2021},
    archivePrefix = {swh},
    eprint = {1:dir:dbf9ee55889563779a09b16f9c451165ba62b6d7}
}
```

6.2 Main works using grobid-quantities

Kyle Hundman and Chris A. Mattmann.
Measurement Context Extraction from Text: Discovering Opportunities and Gaps in Earth Science.
2017, KDD 2017, Halifax, Nova Scotia, Canada.
https://arxiv.org/pdf/1710.04312.pdf

Luca Foppiano, Laurent Romary, Masashi Ishii, and Mikiko Tanifuji.
Automatic identification and normalisation of physical measurements in scientific literature.
September 2019, ACM, DocEng '19, Berlin, Germany.

6.3 Other

UNISCOR (Units Segmentation Corpus) is available at (`resources/dataset/units/evaluation/unit-evaluation-corpus.tei.xml`). It was created with the support of NIMS (National Institute for Material Science), in Japan. For more information, see:

```
Leveraging Segmentation of Physical Units through a Newly Open Source Corpus.
September 2019, JSAP Fall 2019, Sapporo, Japan
```