Remote Procedure Call Server for Unit Management

Allison Brand

Overview of Application

This server manages units in multiplication or addition operations. It is a great tool for dimensional analysis!

It comes with a set of predefined units, the user can define new ones as combinations of these. All calculated results are returned as combinations of basis SI units: kg, m, s, C, K, for mass, distance, time, electric charge, and temperature. Cost $ is also a defined basis unit, so it's possible to have calculations like cost per unit mass.

Commands are human-readable, but the server requires precise placement of space characters. When specifying units of cost, the $ sign must be placed after the number just as the other units, i.e. "3 $", not "$3". This unit has no special treatment, so the grammar is different than expected.

Predefined Units:

Name Known to Server	Full Name	Physical Quantity
kg	Kilograms	Mass
m	Meters	Distance
s	Seconds	Time
C	Coulombs	Electrical Charge
K	Kelvins	Temperature
$	USD	Monetary Cost
Hz	Hertz	Frequency
J	Joules	Energy
N	Newtons	Force
cm	Centimeters	Distance
km	Kilometers	Distance

g	Grams	Mass

The basis units are highlighted in blue.

Implementation Details:

Every unit is represented as a 6-dimensional vector in terms of the basis units, with a numeric coefficient. The number in each dimension represents the power to which that basis unit is raised in the definition of a given unit. For instance, newtons are kilograms times meters per second squared. That is: *1 N = 1 [1,1,-2,0,0,0]*, in the basis *['kg', 'm', 's', 'C', 'K', '$']*. A foot is 0.3048 meters, so *1 ft = 0.3048 [0,1,0,0,0,0]*. This backend representation allows for quick and easy comparison of units: if you want to check that two units are commensurable (they represent the same physical quantity), simply check if the unit vectors are equal. Add the coefficients to add them. If you want to multiply two units, simply add the unit vectors and multiply the coefficients. It is also easy to define new units, you just need to determine the unit vector and coefficient. This representation is simple and elegant. If applications like Microsoft Excel and Google Sheets could handle units like this, it would remove a lot of headache.

Client->Server Message Format

If you attempt to send an empty string or whitespace, my client will catch that and prompt the user for a response again.

The server can process 4 kinds operations:

1. Define

Lets a user add new units to the server's internal dictionary. These updates are stored for the course of the session. Units are case sensitive.

Commands are in the form: `new_unit_name = number known_unit`
For instance, "`mm = 0.001 m`", or "`kJ = 1e+3 kg.m.m/s/s`".
It's okay for it to be unitless, i.e. "`π = 3.14159`". (Special characters are ok.)

Spaces are necessary. The name of the new unit must be followed by the string " = ".

The number can be in any format that python's `float()` method recognizes as numeric (ex: "`-1.23e4`"). Commas in numbers are ignored, so "`5,000`" works.

The new unit can either be a number or a number followed by a space, followed by a unit specification in terms of known units. In that unit specification, `“.”` stands for multiplication and `“/”` stands for division. There cannot be spaces in the unit specification.

This command takes 2 or 3 arguments: (1) the name of the new unit, (2) the numerical coefficient, and (3) an optional specification of units in terms of those in the server’s internal dictionary.

2. Total

Adds or subtracts a sequence of quantities with commensurable units.
Valid Format: quantities separated by `" + "` or `" - "`. Whitespace is required. Parentheses will not be understood, and the command will fail if it has them. A quantity is a unit, or a number with optional units.

Ex: `"1 cm + 2 m - 10 m"` or `“10 - 3”`

This command takes at least 2 arguments. All are quantities that may be unitless.

3. Product

Multiplies or divides a sequence of quantities. \n
Valid Format: quantities separated by `" * "` or `" / "`. Whitespace is required. Parentheses are not accepted. A quantity is a unit, or a number with optional units.

Ex: `"3.00e+8 m/s / 4.30e+14 Hz * 2”`

This command takes at least 2 arguments. All are quantities that may be unitless.

4. Repeat Back

If you give a single quantity, the server will attempt to interpret it in terms of the basis units and repeat it back to you. This is a quick and easy way to test if a unit is already known to the server.

Ex: `“300”` or `“J”` or `“10 m”`

Server->Client Message Format

Human-readable strings that may be presented to the user as-is.

The response are in the form:

```
“requested operation: <Name of operation recognized by server>
<repeat of what it received from client>\n <operation-dependent
response>
```

The colored parts are client-request dependent.
For example, if the client sends “1 N”:

```
'requested operation: Repeat back  1 N
 1.0 m.kg/s/s
```

Or

```
“The received string does not conform to any known operations.”
```

If the request was malformed.

This information is valuable for a user trying to understand when something goes wrong with their supplied input.

Example output for each operation:

I bolded things for readability. They are not actually bolded.

1. **Define:**
   ```
   requested operation: Define  π = 3.14159265359
    π = 3.14159265359 added to units dictionary.
   ```

2. **Total:**
   ```
   requested operation: Total  2 N + 4 m
    Incompatible units.
   ```

3. **Product:**
   ```
   requested operation: Product  3.00e+8 m/s / 4.30e+14 Hz * 2
    1.3953488372093023e-06 m
   ```

   [My comments: The speed of light divided by the frequency of red light times two is 1400 nm. The wavelength of red light is 700 nm.]

4. **Repeat Back:**
   ```
   requested operation: Repeat back  1 N
    1.0 m.kg/s/s
   ```

Example Output

I bolded things to make the below more readable, but they were not actually bolded in the terminal trace.

Client Trace:

```
py interactive_client.py
client starting - connecting to server at IP 127.0.0.1 and port 65432
connection established using client-side port 62264 at IP 127.0.0.1

What message would you like to send to the server? ("quit" to close session.)
 > 3 m + 20 cm
sending message '3 m + 20 cm' to server
message sent, waiting for reply

Received reply 'requested operation: Total  3 m + 20 cm
 3.2 m' from server

What message would you like to send to the server? ("quit" to close session.)
 > 4 km * 5 km
sending message '4 km * 5 km' to server
message sent, waiting for reply

Received reply 'requested operation: Product  4 km * 5 km
 20000000.0 m.m' from server


What message would you like to send to the server? ("quit" to close session.)
 > 5 ft * 2 cm
sending message '5 ft * 2 cm' to server
message sent, waiting for reply

Received reply 'requested operation: Product  5 ft * 2 cm
 'ft' is not a unit known to this program. Please define it using the define function. This error
was encountered while parsing the quantity 5 ft This parsing failure occured while parsing the
multiplication: 5 ft * 2 cm' from server


What message would you like to send to the server? ("quit" to close session.)
 > ft = 0.3048 m
sending message 'ft = 0.3048 m' to server
message sent, waiting for reply
```

```
Received reply 'requested operation: Define  ft = 0.3048 m
 ft = 0.3048 m added to units dictionary.' from server

What message would you like to send to the server? ("quit" to close session.)
 > 5 ft * 1 m
sending message '5 ft * 1 m' to server
message sent, waiting for reply

Received reply 'requested operation: Product  5 ft * 1 m
 1.524 m.m' from server

What message would you like to send to the server? ("quit" to close session.)
 >
Please input a request.

What message would you like to send to the server? ("quit" to close session.)
 > π = 3.14159265359
sending message 'π = 3.14159265359' to server
message sent, waiting for reply

Received reply 'requested operation: Define  π = 3.14159265359
 π = 3.14159265359 added to units dictionary.' from server


What message would you like to send to the server? ("quit" to close session.)
 > π
sending message 'π' to server
message sent, waiting for reply

Received reply 'requested operation: Repeat back  π
 3.14159265359 ' from server


What message would you like to send to the server? ("quit" to close session.)
 > quit
Client quitting at operator request
Sent close notification to server.
client exiting.
```

Server Trace:

```
py units-server.py
```

```
server starting.
now listening for connections at  IP 127.0.0.1 and port 65432
Connected established with client IP 127.0.0.1 and port 62264, using a new socket with server
address: IP 127.0.0.1 and port 65432

Received client message: '3 m + 20 cm'.
 Attempting to parse.
Sending result 'requested operation: Total  3 m + 20 cm \n 3.2 m' back to client!

Received client message: '4 km * 5 km'.
 Attempting to parse.
Sending result 'requested operation: Product  4 km * 5 km \n 20000000.0 m.m' back to client!

Received client message: '5 ft * 2 cm'.
 Attempting to parse.
Sending result "requested operation: Product  5 ft * 2 cm \n 'ft' is not a unit known to this
program. Please define it using the define function. This error was encountered while parsing
the quantity 5 ft This parsing failure occurred while parsing the multiplication: 5 ft * 2 cm" back to
client!

Received client message: 'ft = 0.3048 m'.
 Attempting to parse.
Sending result 'requested operation: Define  ft = 0.3048 m \n ft = 0.3048 m added to units
dictionary.' back to client!

Received client message: '5 ft * 1 m'.
 Attempting to parse.
Sending result 'requested operation: Product  5 ft * 1 m \n 1.524 m.m' back to client!

Received client message: 'π = 3.14159265359'.
 Attempting to parse.
Sending result 'requested operation: Define  π = 3.14159265359 \n π = 3.14159265359 added
to units dictionary.' back to client!

Received client message: 'π'.
 Attempting to parse.
Sending result 'requested operation: Repeat back  π \n 3.14159265359 ' back to client!
Client sent close notification ''
server exiting.
```

Acknowledgments

The basic ideas for how the unit management is implemented came from:
https://github.com/gridlab-d/gridlab-d/blob/86cadc08c264c137f38fa205a432fe73102b3d90/gldcore/unit.cpp#L4
I want to thank Dave Chassin for telling me about it and showing me that GitHub file. This form of unit management is implemented in the Arras Energy simulation software, a sophisticated open-source tool to help electric utilities improve grid operation and understand the effects of proposed renewable energy plants and energy storage resources.

I did not copy any code from Arras Energy, and my implementation differs from that used in Arras Energy. I used simple basis units like ‘kg’ or ‘s’ to make for easier conversion into readable strings. The Arras Energy implementation uses fundamental physical constants like the speed of light in meters per second and Boltzman's constant in kilogram meters squared per second squared Kelvin as the basis.

I borrowed code from https://github.com/bcheikes/csc-249-p1-simple-rpc-app for my client and server.