whisker_serial_order

Serial order task for Whisker (http://www.whiskercontrol.com/). Referred to as **SerialOrder** in this document. By Rudolf Cardinal (rudolf@pobox.com).

Windows installation from PyPI source

Don't use Windows XP; it's too old for MySQL. The following has been tested on Windows 10.

- Install Visual C++ Redistributable Packages for Visual Studio 2013, which you'll want in order to get MySQL Workbench installed.
  - Get this from https://www.microsoft.com/en-GB/download/details.aspx?id=40784 .
- Install Python 3.4, which by default will be installed to C:\Python34\.
  - Explore from https://www.python.org/, or go direct to https://www.python.org/downloads/release/python-344/ .
- Install MySQL. (The alternative is PostgreSQL; see later.)
  - Browse to http://dev.mysql.com/downloads/installer/ and follow the instructions.
  - The web installer works fine here. Choosing the defaults works well, and you can add additional users during setup. The default port is 3306, and the default superuser account is root.
- Download a binary version of PySide 1.2.2, since source code versions have all sorts of tricky compiler requirements.
  - Download `PySide-1.2.2-cp34-none-win_amd64.whl` or `PySide-1.2.2-cp34-none-win32.whl` from http://www.lfd.uci.edu/~gohlke/pythonlibs/#pyside . Remember where you stored it.
- Create and activate a virtual environment. Upgrade the installation tools (may be unnecessary, but confusing errors appear if it was, in fact, necessary). Install PySide and MySQL Connector/Python, then whisker_serial_order.
  - Start a command prompt (Start ⟶ Command Prompt) and type the following.
  - `C:\Python34\python.exe -m pip install --upgrade pip`
  - `C:\Python34\python.exe -m pip install --upgrade virtualenv`
  - `C:\Python34\python.exe -m virtualenv C:\venv_whisker_serial_order`
  - **`C:\venv_whisker_serial_order\Scripts\activate.bat`**
  - `pip install PySide-1.2.2-cp34-none-win_amd64.whl` (change this filename if you are using the 32-bit version; add a path if you stored it somewhere other than the current directory)
  - `pip install https://cdn.mysql.com/Downloads/Connector-Python/mysql-connector-python-2.1.3.tar.gz`
  - `pip install whisker_serial_order`

Ubuntu installation from PyPI source

- Install MySQL or PostgreSQL. See below.
- You should have Python 3 already. Create a virtual environment, activate it, and install the Python things.
  - Start a shell. Enter the following commands:
  - `python3 -m virtualenv ~/venv_whisker_serial_order`
  - **`source ~/venv_whisker_serial_order/bin/activate`**
  - `pip install whisker_serial_order`

After installation

The SerialOrder program itself will now be accessible as the command **`whisker_serial_order`**

without any PATH modifications as long as you have activated the virtual environment (see activation command in bold above).

Create a database (see below).

Compose your database URL. If you create a database named serialorder, with a user named researcher and a password of blueberry, then to use the MySQL Connector/Python interface, the URL would be:

mysql+mysqlconnector://researcher:blueberry@localhost/serialorder

Tell SerialOrder about this URL, either via an environment variable or as a command-line parameter (see below). The simplest is as a command-line parameter, as below.

Launch SerialOrder:

```
whisker_serial_order --dburl=your_URL_as_above
```

For details of all possible command-line options, use:

```
whisker_serial_order --help
```

The **first time** you run it, you should get an error like "Database revision should be 0003 but is None". Update the database with:

```
whisker_serial_order --dburl=your_URL_as_above --upgrade-database
```

Then re-run.

Task overview

The task tests memory for serial position/order.

The task operates with a five-hole box (5 holes of the traditional 9-hole box[1]), with a rear food magazine at the back. All 'holes' have a light inside and an infrared nosepoke detector; the food magazine can also dispense pellets.

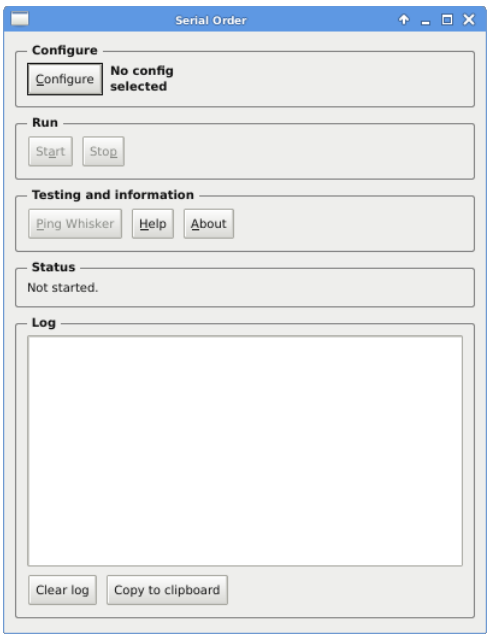
In brief, trials are as follows:

- Trials begin with illumination of the food magazine/tray, and the subject must respond.
- A sequence of lights is presented. For example, if a 4-light sequence is used, the task may aim to present 4-1-3-5. After each individual light presentation, the subject must acknowledge by responding to the location of the light, and after that to the magazine (again signalled by magazine light illumination).
- After the sequence has been presented and acknowledged, a choice of two of the presented lights is offered (e.g. 4, 3). The subject must choose the correct hole to answer the question: **"which one came first?"** Success leads to food.

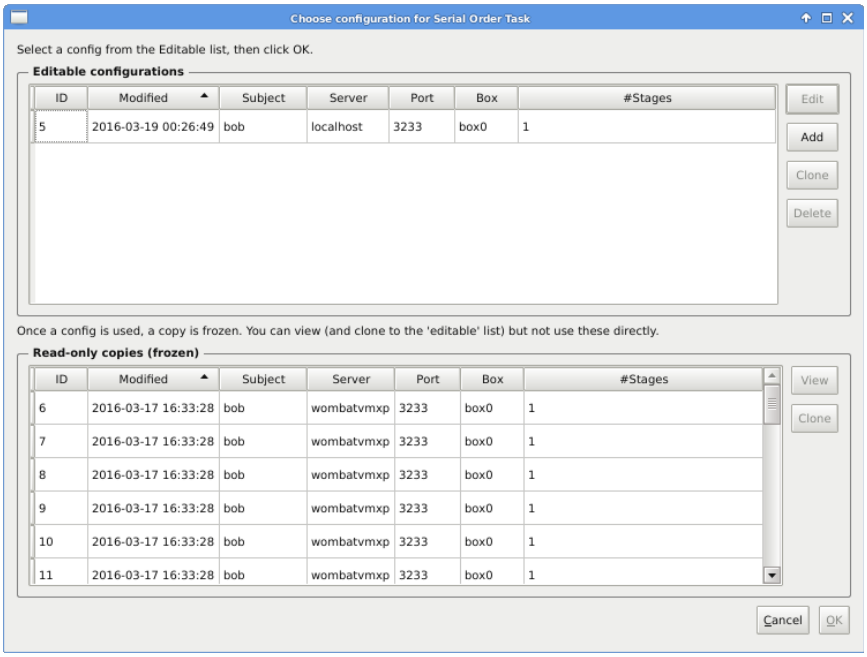
Task operation

When you start successfully, it'll look like this:

1 Rodent five-choice task: Carli et al. (1983), PubMed ID 6 639 741.

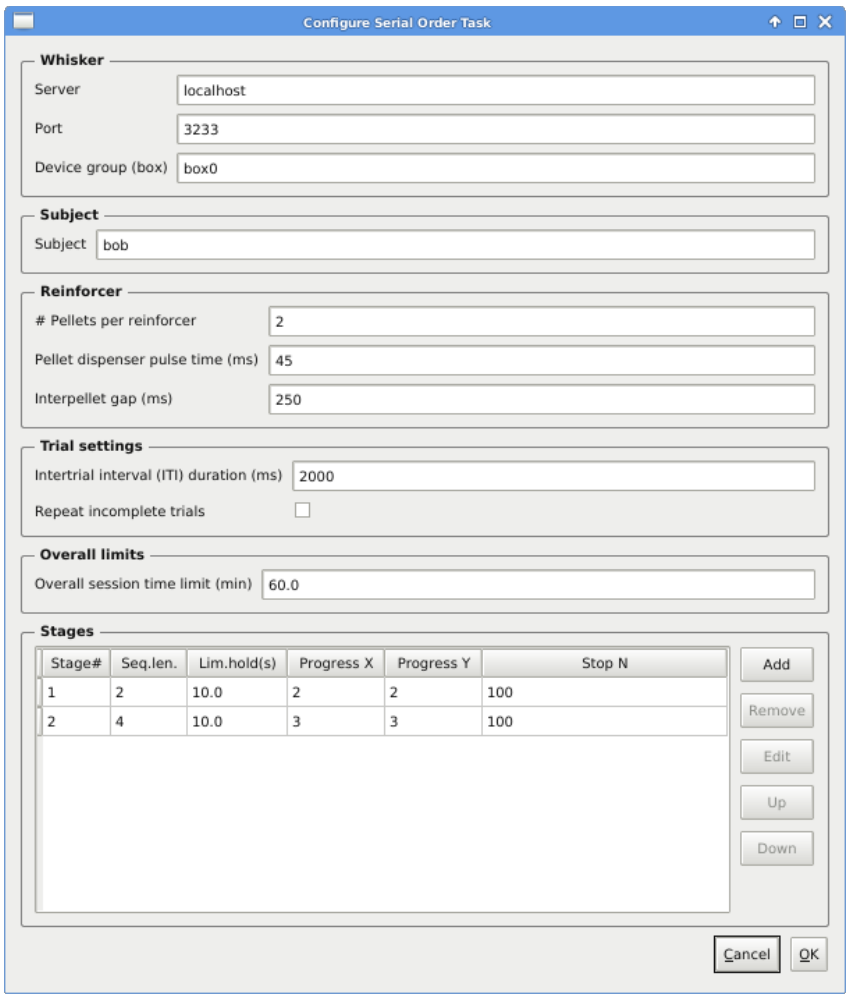


First, configure:



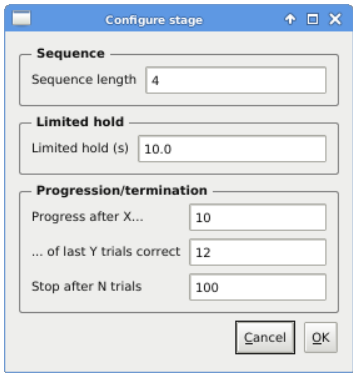
At the top are *editable* configurations. Edit them, and then select one (and click OK) when you're ready to run. When you run a task, the program *copies the editable config into a 'frozen' state* so it's permanently accessible. These frozen copies are shown at the bottom. (You can 'clone' a frozen config back into the editable list, as well.)

When you edit a config, you see this:



Note that the default Whisker server is **localhost** (meaning "this computer") and the default TCP/IP port[2] is **3233**. The *device group (box)* setting should match a device group specified in your Whisker device definition file.[3]

When you edit a stage definition, you see this:



When you're happy with everything, select the config and click **Start** at the main screen. The program will attempt to connect to Whisker and run the task.

Data is logged 'live' to the database, with a COMMIT at the end of every server event processed.

Make sure you use a database with appropriate concurrency (multi-user/multi-client) support; see below.

Device definitions

Within the task and its results, holes are numbered from 1–5. Existing Whisker tasks (e.g. FiveChoice) use HOLE_0 to HOLE_4 and STIMLIGHT_0 to STIMLIGHT_4, so for backwards compatibility, we could use those; however, that's likely to be terribly confusing for anyone trying to debug this task. We therefore

2 http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=whisker

3 See http://www.whiskercontrol.com/

provide new device names starting 'SO_' (which can co-exist with old names within one Whisker device definition file if necessary).

Inputs

SO_HOLE_1
SO_HOLE_2
SO_HOLE_3
SO_HOLE_4
SO_HOLE_5
REARPANEL

Outputs

SO_STIMLIGHT_1
SO_STIMLIGHT_2
SO_STIMLIGHT_3
SO_STIMLIGHT_4
SO_STIMLIGHT_5
HOUSELIGHT
PELLET
MAGLIGHT

Choice of database: MySQL 5.6.4+ or PostgreSQL

SerialOrder uses SQLAlchemy (http://www.sqlalchemy.org/) to talk to database. This permits a wide variety of back-end databases. **However,** some additional constraints apply (detailed below). These are:

- We want lots of task instances to be able to use the same database, because that helps analysis enormously.
- We want to use a freely available database engine.
- Use of a lightweight database like SQLite poses some problems if many tasks are writing to the same database at once[4], as SQLite locks the whole database when writing. This might cause problems if many tasks are writing events at high speed to the database – but it also prohibits one task editing a config (with an SQLite transaction active) while other tasks write. That's a very common situation, which argues strongly for a formal client/server database.
- In addition, we want to store timestamps to millisecond accuracy. There are a variety of ways of doing this, with and without timezone storage. The only entirely consistent way across databases is to use a textual format (e.g. ISO-8601, such as 2016-03-02T22:43:03.710817+01:00 or an equivalent with less punctuation). However, this reduces the ability to perform simple arithmetic. For the purposes of behavioural tasks, time differences (latencies) are important, and timezones aren't, so we can use a high-precision UTC date/time. This gives us the DATETIME(6) type in MySQL 5.6.4+, or the TIMESTAMP type in PostgreSQL. More detail below.

Installing PostgreSQL

To choose between MySQL and PostgreSQL, let's compare quick installation and startup, using an old Windows version (Windows XP, 32-bit). First PostgreSQL 9.5:

- *Installation under Windows XP.* Simple. The default port is 5432, and the default superuser account is postgres. You can specify your data directory as you install, so it's probably worth putting this somewhere outside "C:\Program Files", such as c:\postgresql_data.
- *Running something.* Using *Start → Programs → PostgreSQL 9.5 → SQL Shell (psql)* fails to connect with supplied defaults.[5] If you use a short[6] or full[7] command line version, it does work, so this

4 http://www.sqlite.org/whentouse.html
5 This calls `"C:\Program Files\PostgreSQL\9.5\scripts\runpsql.bat"`.
6 `"C:\Program Files\PostgreSQL\9.5\bin\psql.exe" --username=postgres`
7 `"C:\Program Files\PostgreSQL\9.5\bin\psql.exe" -h localhost -U postgres -d postgres -p 5432`

indicates a bug in `runpsql.bat`.[8] Once at the SQL command line, you can use the `help` command, providing you are at the `postgres=#` (command start) prompt not the `postgres-#` (command continuation) prompt. The GUI administrator works better out of the box: *Start → Programs → PostgreSQL 9.5 → pgAdmin III*.

- *Creating a user.* Within the GUI administrator, double-click on the local server to connect. Right-click "Login Roles" to choose "New Login Role…". Create a user, giving it a name (e.g. 'researcher') in the Properties tab and a password in the Definition tab. Don't forget the password, or you won't be able to connect with this user from the psql tool.[9]
- *Creating a database.* Within the GUI administrator as before, right-click "Databases" to choose "New Database…". Give it a name (e.g. serialorder) and assign one of your users as its owner.

Installation of PostgreSQL under Ubuntu is also easy.[10]

Installing MySQL

Then MySQL 5.7.1:

- *Installation under Windows XP.* In short, don't use this OS. Failure details are below.
  - Installation of Microsoft .NET 4.0 is a prerequisite, and this is easy. Installation of MySQL itself is easy[11]. However, the web community installer (1.6 Mb download) failed miserably. The installer failed to start the server. The MySQL57 service reports that it isn't a Win32 program when you try to run it. The full community installer (377.9 Mb download) failed. MySQL Workbench failed to install; in the depths of the very long log, it said "The operating system is not adequate for running MySQL Workbench 6.3 CE. You need Windows 7 or newer and .Net 4.0 Client Profile installed." Then the installer crashed, saying "This class is designed only for Windows Vista and higher." (At least the error messages are better with this installer!)
- *Installation under Windows 10.* Much better.
  - *Prerequisite:* Visual C++ Redistributable Packages for Visual Studio 2013, which you'll want in order to get MySQL Workbench installed.[12]
  - *Prerequisite:* Python 3.4, for MySQL Connector/Python.[13]
  - The web installer works fine here. Choosing the defaults works well, and you can add additional users during setup. The default port is 3306, and the default superuser account is root.
- *Creating a user.* You might have already done this during installation, as above. If not, run MySQL Workbench, click *USERS AND PRIVILEGES*, then "Add Account". Specify the details and click "Apply".
- *Creating a database.* Run MySQL Workbench. Under *SCHEMAS*, right-click one and click "Create schema…" Give the new schema (database) a name and click "Apply", then "Apply" again to confirm.

So either is perfectly reasonable.

Installing MySQL under Ubuntu is also easy.[14]

I suggest **MySQL,** but only because I've used it more.

---

8 It gets stuck, or takes an *extremely* long time, on this line: `for /f "delims=" %%a in ('chcp ^|find /c "932"') do @ SET CLIENTENCODING_JP=%%a`, presumably relating to Windows version incompatibility.

9 http://www.postgresql.org/docs/current/static/auth-methods.html

10 Install with `sudo apt-get install postgresql postgresql-contrib pgadmin3 libpq-dev`. Connect with `sudo -u postgres psql postgres`. Set a password for the 'postgres' user using `\password postgres`. Quit with `\q`. Use `pgadmin3` for the rest.

11 From http://dev.mysql.com/downloads/installer/

12 https://www.microsoft.com/en-GB/download/details.aspx?id=40784

13 https://www.python.org/downloads/release/python-344/

14 For the version that comes with the OS: `sudo apt-get install mysql-server mysql-client mysql-workbench`. For a more up-to-date version, download a .deb file from https://dev.mysql.com/downloads/repo/apt/, install it (e.g. `sudo dpkg -i mysql-apt-config_0.6.0-1_all.deb`), and follow the on-screen prompts. This reconfigures APT, so you then need to run `sudo apt-get update` and finally `sudo apt-get install mysql-server mysql-workbench`. If you are upgrading, note also the command `mysql_upgrade -u root -p` (which you run when the server has started); this repairs relevant tables, after which you must restart MySQL (with `sudo service mysql restart`).

Telling SerialOrder which database to use: the database URL

If your database is called ‘serialorder’, your database user is ‘researcher’ and their password is ‘blueberry’, then you can use an environment variable or a command-line switch to tell the task how to connect with the database, like this:

Linux, environment variable, PostgreSQL	`export WHISKER_SERIAL_ORDER_DB_URL=postgresql://researcher:blueberry@localhost/serialorder` `whisker_serial_order`
Any OS, command-line switch, PostgreSQL	`whisker_serial_order --dburl postgresql://researcher:blueberry@localhost/serialorder`
Linux, environment variable, MySQL	`export WHISKER_SERIAL_ORDER_DB_URL=mysql://researcher:blueberry@localhost/serialorder` `whisker_serial_order`
Windows, environment variable	Use the “SET *var=value*” syntax, or (better?) set the environment variable from the Control Panel, then run `whisker_serial_order`.
Any OS, command-line switch, MySQL	`whisker_serial_order --dburl mysql://researcher:blueberry@localhost/serialorder`

For MySQL under Windows,

Advice on analysis using the database, focusing on MySQL

MySQL Workbench is pretty good, and free.

Start it, then connect to your MySQL instance.

You can then open a query window. To get going, presuming your database is called serialorder, you can use the commands:

```
USE serialorder;
SELECT * FROM session;
```

… and click the lightning symbol to run the query.

To copy/paste results, click in the output and then use Ctrl-A to select all rows and Ctrl-C to copy. It should paste right into spreadsheets (using commas to delimit cells and apostrophes to delimit text). If you select rows using Ctrl-A or shift-click, then the right mouse button offers more copy/paste options.

To make your query a permanent part of the database, you can create it as a **view.**

A number of views are pre-created for you. Their names contain ‘view’. You’ll find them by exploring the **SCHEMAS** list (e.g. SCHEMAS → serialorder → Views → …). Similarly, you can explore the tables directly like this (SCHEMAS → serialorder → Tables → …); right-click a table and choose “Select Rows” to see the raw data.

Avoid editing data in the database. It carries a significant risk of problems. There are few situations it would be wise. Use is only for experts. The only realistic use-case is if you entered a wrong subject name and failed to notice as you were starting the session; you are then probably looking for the SQL syntax `UPDATE config SET subject='newname' WHERE id=`*XXX*`;`, where *XXX* is the ID number of the record you have determined to be faulty. You can also edit cells directly from the Table view in MySQL Workbench (right-click the cell and choose “Open Value in Editor”).

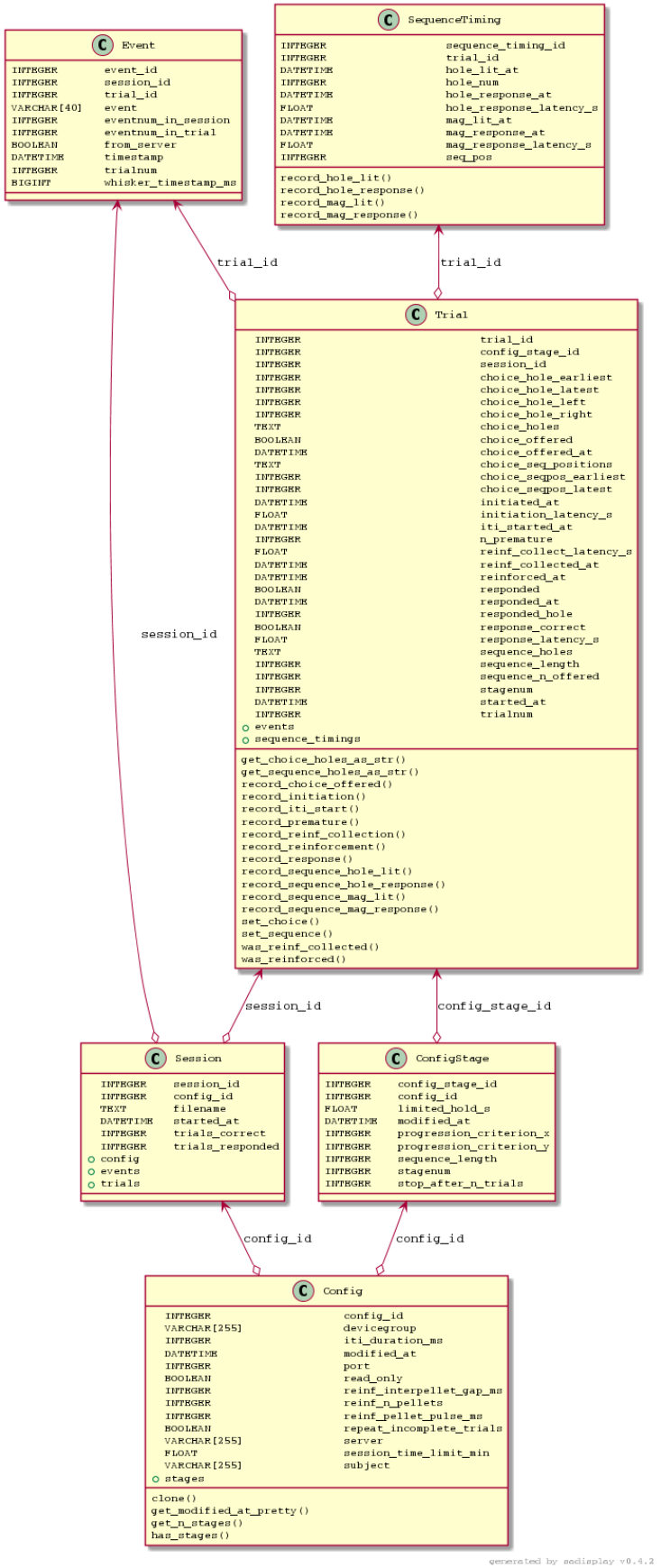
Instead, add new tables. Suppose, for example, that you have information about session numbers, or group membership (sham or lesion?), or drug manipulation prior to the session. You should **create a new table,** store the linking information, and link dynamically to produce your queries. Here’s a made-up example:

```
-- This is an SQL comment. We will create a table:
CREATE TABLE lesion (
    -- This table tells us which subjects have what lesion.
    -- Suppose a subject can only ever be in one lesion group;
    -- that means only one row per subject in this table.
    -- So subject can be our primary key.
    subject VARCHAR(255) NOT NULL,
```

```
    expgroup VARCHAR(50),  -- Avoid using the name 'group' as it is an SQL
        -- keyword. We will store words like 'sham' and 'lesion'.
    exclude BOOLEAN, -- In case we want to remove all data for a subject.
    PRIMARY KEY (subject)
);
CREATE TABLE drug_session (
    -- This table tells us which subjects had what drug/dose on which day.
    drug_session_id INTEGER NOT NULL AUTO_INCREMENT,
    session_id INTEGER NOT NULL,
    drug VARCHAR(50),  -- Maybe 'amphetamine_0_3', 'amphetamine_1_0'.
    PRIMARY KEY (drug_session_id),
    FOREIGN KEY (session_id) REFERENCES session(session_id)
       -- The session_id field will refer to session.session_id; this prevents
        -- us from creating drug information for non-existent sessions, etc.
);
-- Then, after inserting some data, we could fetch all trial
-- information and bring in matching lesion/drug details with:
SELECT
    L.subject,
    L.expgroup,
    D.drugsession,
    T.*
FROM
    lesion L
    INNER JOIN config C ON L.subject = C.subject
    INNER JOIN session S ON S.config_id = C.config_id
    INNER JOIN drug_session D ON D.session_id = S.session_id
    INNER JOIN trial T ON T.session_id = S.session_id
WHERE
    NOT L.exclude
;
```

Schema generation

Try `whisker_serial_order --help` to see a few other things it can do. If you have Java and the PlantUML .jar file (http://plantuml.com/), you can use the `--schema` option to generate a schema picture of the database, shown below.



Additional safety data output

As always, for safety, good Whisker tasks write their data to two places: the database and a text file. In the brave new world of real databases and Python, this task writes its data to the proper database ‘live’, so the database is always up to date. When the task finishes, it reads that database and writes both *structure* and

data to a text file. It does so in SQL format, so that a fully structured representation of all data relevant to a given session can easily be regenerated simply by replaying the SQL output into a fresh database.

The directory used for these files is one of the following, in descending order of priority: (1) the `--outdir` command-line parameter; (2) the WHISKER_SERIAL_ORDER_OUTDIR environment variable; (3) the current working directory from which the task was started.

Some development notes

Dates and times

As above, the end user is probably best off with a native format that supports microsecond-accuracy precision, such as the DATETIME(fsp), e.g. DATETIME(6), format available in MySQL 5.6.4 and higher.[15] There are others[16], notably PostgreSQL, which uses a TIMESTAMP format that has microsecond precision. You'd think it can store time zones as well (TIMESTAMP WITH TIME ZONE), but close inspection shows that "For TIMESTAMP WITH TIME ZONE, the internally stored value is always in UTC… [a]n input value that has an explicit time zone is converted to UTC…".[17] In other words, there's not much to choose between PostgreSQL and MySQL on the basis of date/time handling.

Interestingly, on the commercial side, SQL Server 2008+ provides DATETIMEOFFSET, which preserves timezone information,[18] and Oracle has a version of TIMESTAMP WITH TIME ZONE that can preserve timezone information.[19] Neither PostgreSQL or MySQL appear to have an equivalent; the best they offer is UTC high-precision storage (and you could store the timezone of origin separately, e.g. with the TimezoneType from SQLAlchemy-Utils). I haven't managed to get TimezoneType working cleanly, in that (a) I'm unsure of the best general way to get the current timezone using either pytz or dateutil.tz.tzlocal(); (b) Alembic adds a "length=50" argument to the constructor, which is wrong and requires manual removal. Anyway, it's not too important here.

For general advice, see also [20].

The best Python module is Arrow[21], which offers arrow.now() to get a timezone-aware, microsecond-precision object in the local timezone.

There is an ArrowType for SQLAlchemy in SQLAlchemy-Utils[22]; however, this converts to UTC as it sends to the database (and UTC Arrow objects back out again), so the source timezone is lost. But this is actually normal PostgreSQL behaviour, as above, which always uses UTC internally. An alternative (as per CamCOPS) is to use ISO-8601 strings, but they're much less convenient for end user comparison. So, the best bet is to use Arrow, ArrowType, and accept that everything in the database is in UTC. This works fine for PostgreSQL, where the default TIMESTAMP has microsecond precision. **However,** by default the ArrowType uses a plain DATETIME in MySQL, which has only second precision; we need DATETIME(6). We therefore require not only MySQL 5.6.4+ but a custom ArrowMicrosecondType. Duly added. For SQL Server, this class uses DATETIME2, available from SQL Server 2008+.[23]

To summarize, **in this task, all timestamps are in UTC.**

Where to store BLOBs

Completely irrelevant here, but see [24].

Primary key naming convention

Primary keys as 'id' or 'trial_id'? There's no right answer.[25] However, here we are aiming for simplicity of

15 http://dev.mysql.com/doc/refman/5.7/en/datetime.html
16 http://docs.sqlalchemy.org/en/latest/core/engines.html#supported-databases
17 http://www.postgresql.org/docs/9.1/static/datatype-datetime.html
18 https://msdn.microsoft.com/en-us/library/bb630 289.aspx; https://blogs.msdn.microsoft.com/bartd/2009/03/31/the-death-of-datetime/
19 https://docs.oracle.com/cd/B19 306_01/server.102/b14 225/ch4datetime.htm#i1 006 081; but see https://tonyhasler.wordpress.com/2010/09/04/tonys-tirade-against-timestamp-with-time-zone/
20 http://stackoverflow.com/questions/1 646 171/mysql-datetime-fields-and-daylight-savings-time-how-do-i-reference-the-extra; and especially **http://stackoverflow.com/questions/2 532 729/daylight-saving-time-and-time-zone-best-practices**
21 http://crsmithdev.com/arrow/
22 http://sqlalchemy-utils.readthedocs.org/en/latest/data_types.html
23 https://blogs.msdn.microsoft.com/cdnsoldevs/2011/06/22/why-you-should-never-use-datetime-again/; http://stackoverflow.com/questions/1 334 143/sql-server-datetime2-vs-datetime
24 https://wiki.postgresql.org/wiki/BinaryFilesInDB
25 http://programmers.stackexchange.com/questions/114 728; http://stackoverflow.com/questions/1 369 593

use for database novices. Using `SELECT a.something, b.*` statements may be common, at which point when a column labelled 'id' pops up, people may be unclear as to what it is. So for this particular scenario, we will use the 'table.table_id' convention.

For queries, duplicate column names don't matter (and if their values don't *match*, that's an important clue to query failure!). For views, duplicates do matter, but views should be more carefully constructed anyway.

Strings

The SQLAlchemy String() type can be of variable length in PostgreSQL and SQLite, but needs a length in MySQL. The SQLAlchemy Text() is always of variable length.

Avoid TEXT columns for things that have a realistic maximum length, as you can't use TEXT columns for primary keys (e.g. for 'subject' tables that cross-refer).

Trial maths

Definitions. The number of k-permutations from n objects $P(n, k) = n! / (n - k)!$. The number of k-combinations from n objects $C(n, k) = n! / [(n - k)!k!]$.

In our situation, we always offer two choices and have five holes available; this gives $C(5, 2) = 10$ possible *spatial* choices. For a stimulus sequence of length l, the number of sequences is $P(5, l)$. The spatial choice is not independent of the sequence (e.g. if you present lights 314 you can't then offer a choice of 25). The *serial order* choice is independent of the sequence, and there are $C(l, 2)$ of these, as follows:

Sequence length l	Number of possible sequences, $P(5, l)$	Number of serial order choices, $C(l, 2)$	Number of spatial choices, $C(5, 2)$; not all available on any given trial	Maximum number of independent trial types, $P(5, l)C(l, 2)$
2	20	1	10	20
3	60	3	10	180
4	120	6	10	720
5	120	10	10	1200

In general, since this task is about serial order detection, *serial order choices are the most important* and should vary most rapidly (e.g. for $l = 4$, every 6 trials should cover all of the 6 possible serial order choices, in random order). Next most important is spatial choice (it's of some importance that choices are equally distributed spatially); **note** that the number of spatial choices is not always a multiple of the number of serial order choices. Last comes sequence (it's least important that all possible sequences are presented). But there is no obviously consistent way of randomizing in groups across all three (since some of them are interdependent and they are not necessarily multiples of each other), so I think the best approach is to randomize across sequences, which should give a nice spatial spread (through randomness alone), in addition to the guarantees about serial order sampling.

Therefore our algorithm will be:

- For each stage, we establish the possible serial order choices, and the possible sequences. We combine them into all possible combinations.
- We then shuffle them in blocks, such that *serial order is randomized in blocks with the highest rate of change,* so that in every $C(l, 2)$ trials there are all possible serial orders.
- Since that gives a rather dull and static stimulus sequence, we then *randomize the sequences.* So sequences vary randomly but the serial order sampling is in blocks.
- We then sample in order from our 'hat'. If we run out, we repopulate the hat, as above.

Progression maths

If we progress when x of the last y trials are performed correctly, then we should have some sense that this isn't going to happen by chance. In R, use `binom.test(x, y)` to get the p-value based on the assumption of $P = 0.5$ for chance (and it is, after all, a two-choice test). The default values are 10 out of 12, for $p = 0.03857$.

Trials can also be failed by not responding, affecting the "ignorance $\Rightarrow P = 0.5$" assumption, but in a conservative way.

Change history

v0.1.0 – Feb–Mar 2016

Started. Released 21 Mar 2016.