SimpleSchedules

A Whisker client

---

by Rudolf Cardinal

www.whiskercontrol.com

Distributed by Campden Instruments Ltd (www.campden-inst.com)

SimpleSchedules

Printed: May 2017 in Cambridge, UK

Creator (Whisker)

Rudolf N. Cardinal

Design and Programming (Whisker)

Rudolf N. Cardinal

Michael R. F. Aitken

Legal Advisor (CUTS)

Adjoa D. Tamakloe

Sales (Campden)

Julie Gill

Contacting the authors:

For information about Whisker, visit http://www.whiskercontrol.com/.

If you have sales enquiries about Whisker, contact Campden Instruments Ltd at http://www.campden-inst.com/.

If you have comments or technical enquiries that cannot be answered by the sales team, contact the authors:

Rudolf Cardinal (rudolf@pobox.com)
Mike Aitken (m.aitken@psychol.cam.ac.uk)

Table of Contents

Foreword

WARNING

Whisker is a system designed for research purposes only, and should never be used to control medical apparatus or other devices that could endanger human life.

DISCLAIMER

The authors, copyright holders, and distributors disclaim all responsibility for any adverse effects that may occur as a result of a user disregarding the above warning.

1 SimpleSchedules

1.1 About SimpleSchedules

Purpose

Simple (concurrent) schedules of reinforcement.

Software requirements

Requires Whisker v2.0 or greater.

Data storage

- Text-based output to disk.
- ODBC data storage to a database (supplied).

Author

Rudolf Cardinal (rudolf@pobox.com).

Copyright

Revision history

- Version 2.1 (14 October 2002): added progressive-ratio schedules.
- Version 2.2 (4 November 2002): added XML parameter storage, FR1 with delayed reinforcement, a timeout option for all schedules, and nosepoke recording.
- Version 2.3: fixed minor bug (loss of unsaved info in parameters dialogue when selecting database)
- Version 2.4 (6 December 2002): addition of changeover delays for concurrent schedules
- Version 2.5 (4 June 2003): PR bug fixed
- Version 2.6 (14 July 2003): status message display scrolls properly
- Version 2.7 (22 Nov 2003): writes version info to summary file
- Version 2.8 (11 Mar 2004): PROGRATIO_DOUBLEINCREMENT schedule. **Traylight** support added (therefore requires a traylight). PR schedules can end based on time since last response (or time since last reward, as before).
- Version 2.9 (21 Mar 2005). Bug fix: random-number generators for concurrent random schedules not totally independent (probably). Option for timeout on one schedule to affect the other in a two-schedule situation.
- Version 3.0 (8 March 2007). Easier compilation for users.
- Version 3.1 (June 2007). Option not to reinforce the first response in an interval schedule.
- Version 4.0 (12 Jan 2009). Server default changed from "loopback" to "localhost" (Windows Vista compatibility and more general standardization).
- Version 4.1 (24 Mar 2009). Support for antique levers.
- Version 4.2 (5 Sep 2013). Option to remove 10-second hard-limit safety timer for non-IV pump use.
- Version 4.3 (23 Mar 2014). Reinforcement-associated CS lights.
- Version 4.4 (3 Sep 2014). Hammond contingency schedule.
- Version 4.5 (14 Apr 2015). RPI schedule. Rebuild to use WhiskerClientLib 4.62 with new socket code.

1.2 Required devices

The program requires to claim devices in groups named **box0, box1, box2...** with device names as listed below in bold:

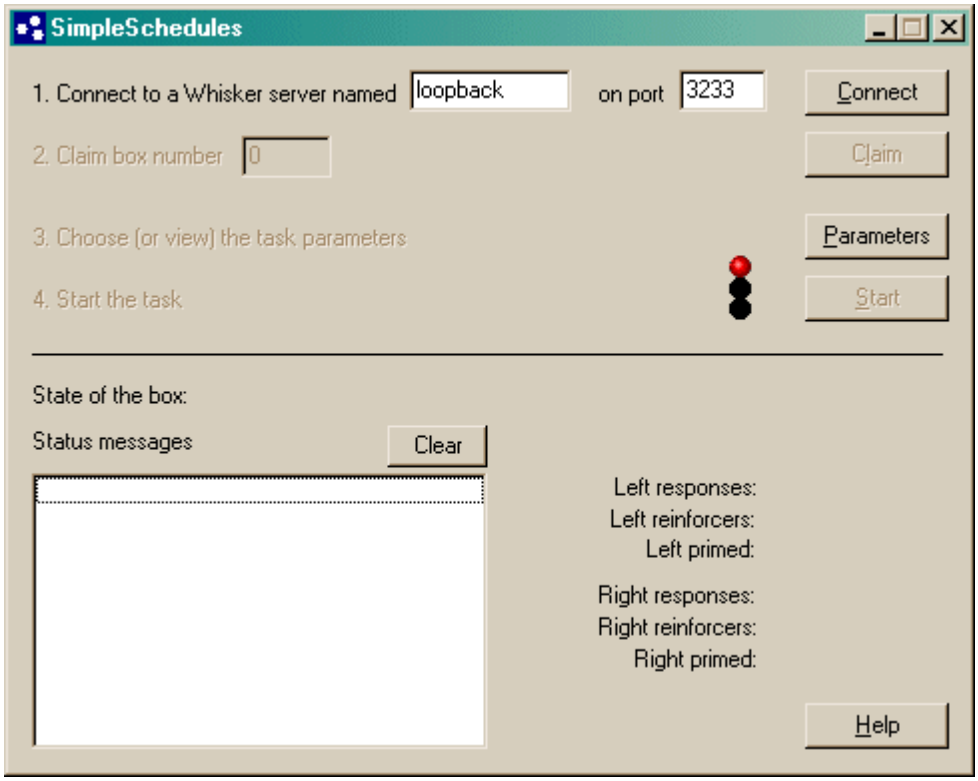
```
// Names of lines the program expects to be able to claim
NOSEPOKE                // input
LEFTLEVER               // input
RIGHTLEVER              // input
HOUSELIGHT              // output
PUMP                    // output
DIPPER                  // output
LEFTLEVERCONTROL        // output
RIGHTLEVERCONTROL       // output
PELLET                  // output
TRAYLIGHT               // output
```

Please ensure that these devices are available and listed in the device definition file in use by the server.

Note that if you are using the Antique Levers version of the task, you **do not neet** `LEFTLEVERCONTROL` and `RIGHTLEVERCONTROL`, **but you do need `LEFTLEVERMOTOR`** (output), **`RIGHTLEVERMOTOR`** (output), **`LEFTLEVERPOSITION`** (input), and **`RIGHTLEVERPOSITION`** (input).

1.3 Using the task

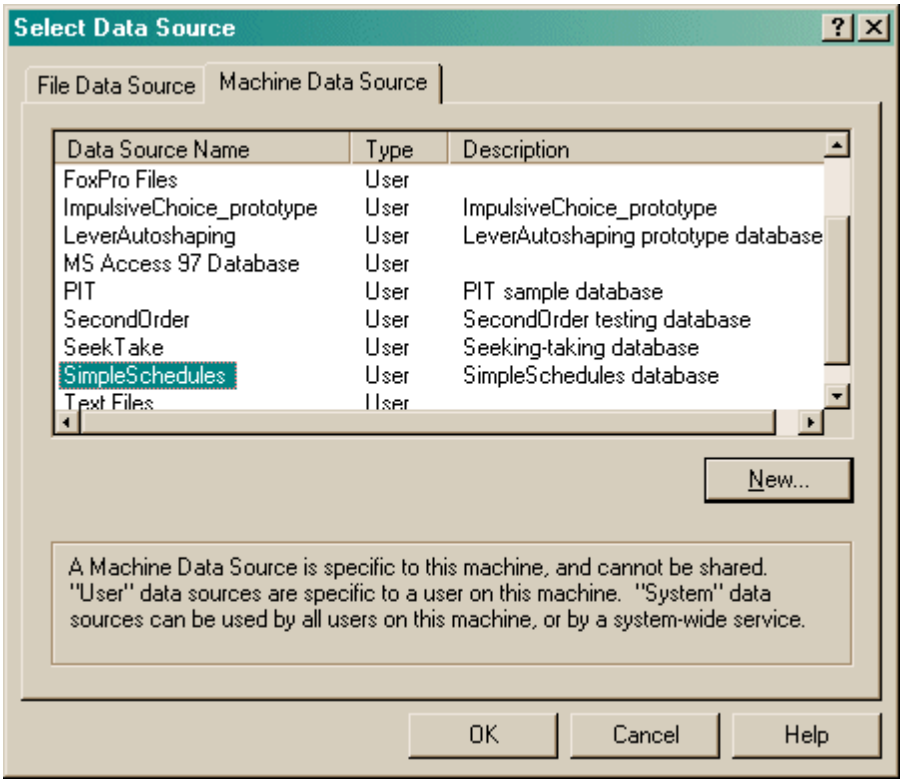
When you run the task, the main screen looks as follows:



You must connect to a Whisker server, claim an operant chamber (box), and set up the parameters

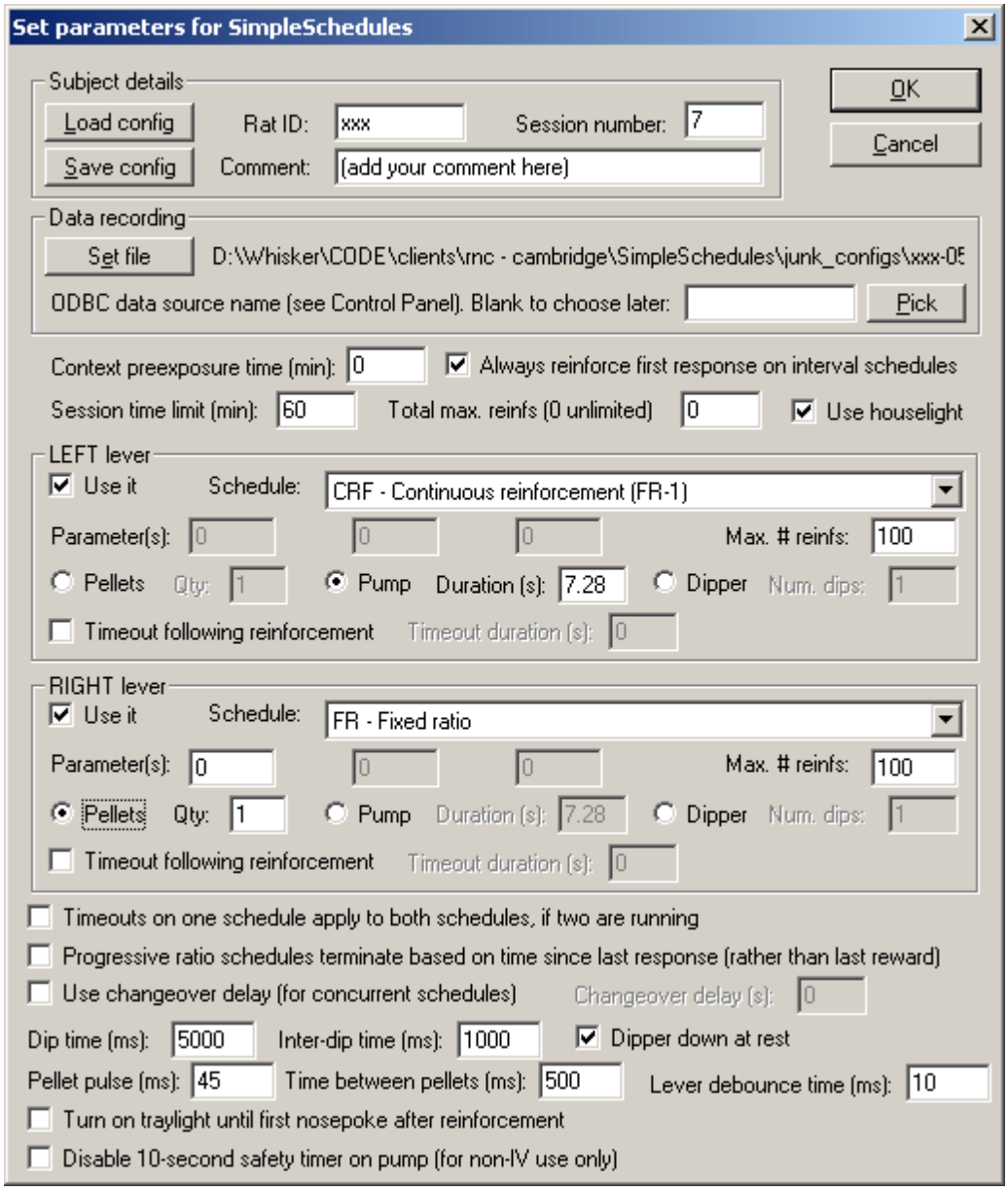
for your task. Once that's done, the traffic lights will turn amber. When you are ready, press *Start* to begin the task.

When the task finishes, it saves data to disk and pops up a new dialogue box for you to select a database to store the data to. (The data sources are configured under *Control Panel* → *ODBC*.) If you previously specified an ODBC data source in the parameters, that data source is used automatically and you will only see a dialogue box if something goes wrong and the program needs your input.



1.4 Parameters

The parameters dialogue box looks like this:



Define the overall session parameters - context preexposure time (how long the subject shuld be sitting around in the chamber before the schedules start, in minutes); maximum session length (after which the whole session stops); maximum *total* number of reinforcers available (after which the whole session stops); whether or not to use the houselight.

You can run schedules on the left and/or right levers, simultaneously if you wish. The schedules are:

- **CRF - continuous reinforcement (FR-1).** One reinforcer per response.
- **EXT - extinction.** No reinforcers.
- **FR *x* - fixed ratio.** One reinforcer per *x* responses.
- **VR *x* to *y* - variable ratio (specifying min, max).** After a variable number of responses (randomly chosen from *min* to *max* inclusive), one reinforcer is delivered.
- **RR *x* - random ratio.** P(reinforcer | response) = 1/*x*.

- **PROB *p* - probabilistic.** P(reinforcer | response) = *p*.
- **FI *x* - fixed interval.** The first response after *x* seconds is reinforced. The *first response* of the schedule is also reinforced.
- **RI *x* - random interval.** Reinforcement is set up on a random-time schedule (see below); after reinforcement has been set up, the next response is reinforced.
- **VI *x* to *y* - variable interval (specifying min, max).** After a variable time (from *min* to *max* seconds), the next response is reinforced.
- **FT *x* - fixed time (NONCONTINGENT).** No lever is present. Reinforcement is delivered every *x* seconds.
- **VT *x* to *y* - variable time (specifying min, max) (NONCONTINGENT).** No lever is present. The schedule waits for between *min* and *max* seconds, then delivers a reinforcer, then repeats.
    - **RT *x* - random time (NONCONTINGENT).** Every second, p(reinforcer delivered this second) = 1/*x*. Thus, on average, reinforcement is delivered once every *x* seconds, but the subject cannot predict the likelihood of reinforcement based on how long it has waited (unlike a typical VT schedule).
- **PR - progressive ratio - add one (1,2,3,4...)** - progressive ratio schedule, adding one to the ratio requirement at each step. The schedule termination is determined by the parameter; if *parameter* is >0, then when *parameter* minutes have elapsed since the last reinforcer (or response - see below), the schedule stops. We suggest 60 as a sensible value.
- **PR - progressive ratio - double (1,2,4,8...) -** progressive ratio schedule, doubling the ratio requirement at each step. The schedule termination is determined by the parameter; if *parameter* is >0, then when *parameter* minutes have elapsed since the last reinforcer (or response - see below), the schedule stops. We suggest 60 as a sensible value.
    - **PR - progressive ratio - Fibonacci (1,1,2,3,5...) -** progressive ratio schedule with a Fibonacci progression. The schedule termination is determined by the parameter; if *parameter* is >0, then when *parameter* minutes have elapsed since the last reinforcer (or response - see below), the schedule stops. We suggest 60 as a sensible value.
    - **PR - progressive ratio - Roberts exponential (A * exp(reinfnum * B) - A) -** progressive ratio schedule with an exponential progression, based on Roberts DCS & Richardson NR (1992), Self-administration of psychomotor stimulants using progressive ratio schedules of reinforcement, *Neuromethods* 24: 233-269 (eds Boulton A, Baker G, Wu PH; Humana Press). The ratio requirement is (A * exp(reinforcer number * B)) - A, rounded to the nearest integer. Typically, A is 5. A typical schedule might have B=0.2; these values yield ratio requirements {1, 2, 4, 6, 9, 12, 15, 20, 25, 32, 40, 50, 62, 77, 95, 118, 145, 178, 219, 268, 328, 402, 492, 603, 737, 901, 1102, 1347, ...}. A steeper PR schedule is obtained with B=0.25, giving {1, 3, 6, 9, 12, 17, 24, 32, 42, 56, 73, 95, 124, 161, 208, 268, 346, 445, 573, 737, 948, 1218, 1566, 2012, 2585, 3321, 4265, 5478, ...} The schedule termination is determined by the other parameter (on the left, labelled (min)); if this parameter is >0, then when this many minutes have elapsed since the last reinforcer (or response - see below), the schedule stops. We suggest 60 as a sensible value.
    - **DELAYED_FR1 - FR1 with delayed reinforcement.** This is an FR1 schedule, but there is a delay between responding and reinforcement. This delay is the sole parameter (specified in seconds).
    - **PR - progressive ratio - double increment every A reinforcers.** The increment starts at 1, and doubles every A reinforcers. If A is 8, then the ratio requirements are 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 18, 20, 22, 24, 28, 32, 36... The schedule termination is determined by the parameter; if *parameter* is >0, then when *parameter* minutes have elapsed since the last reinforcer (or response - see below), the schedule stops. We suggest 60 as a sensible value.
    - **HAMMOND_CONTINGENCY - Hammond (1980) instrumental contingency.** After Hammond (1980, J Exp Analysis Behav 34: 297). Time is windowed into 1-second bins. A decision about reinforcement is made at the end of each time bin (not when responding occurs). At the end of each time bin, if there has been at least one response during that time bin, reinforcement is delivered with probability P(reinf | response). If, instead, no responses were made in that time bin, reinforcement is delivered with probability P(reinf | ~response). You specify the two probabilities. The contingency is P(reinf | response) - P

(reinf | ~response).

Special case: the first response on contingent interval schedules (FI, RI, VI) is always reinforced.

Define the reinforcer for each schedule in use. For example, one reinforcer might be two pellets, or three dips, or a five-second infusion from a pump.

Choose whether the lever **light** (on the same side as the lever) should illuminate when reinforcement is triggered, and for how long. *Note: contention scheduling is not done if multiple reinforcers overlap.*

Choose whether or not to implement a **timeout** following reinforcement. (Added primarily intended for IV self-administration experiments.) During the timeout, the schedule is inactive and the lever is retracted.

See also Notes on Reinforcement Timing.

Set the **maximum number of reinforcers** for each schedule, if you wish. (Once this limit is reached, that session will stop, and if there is a lever out for that schedule it will retract. The session will then run until the other schedule also stops, if you are using two schedules.)

If you are using two schedules, you can also choose that the **timeouts on one schedule apply to both.** If you don't tick this (as is the default), timeouts on two schedules are independent. If you do, then the right schedule will be placed in a timeout state whenever the left schedule triggers a timeout, and vice versa.

For **interval schedules,** you may choose whether the very first response of the session is reinforced or not.

For **progressive ratio schedules,** you may choose whether the timeout that eventually terminates the schedule, if selected, is calculated from the last response or the last reinforcer.

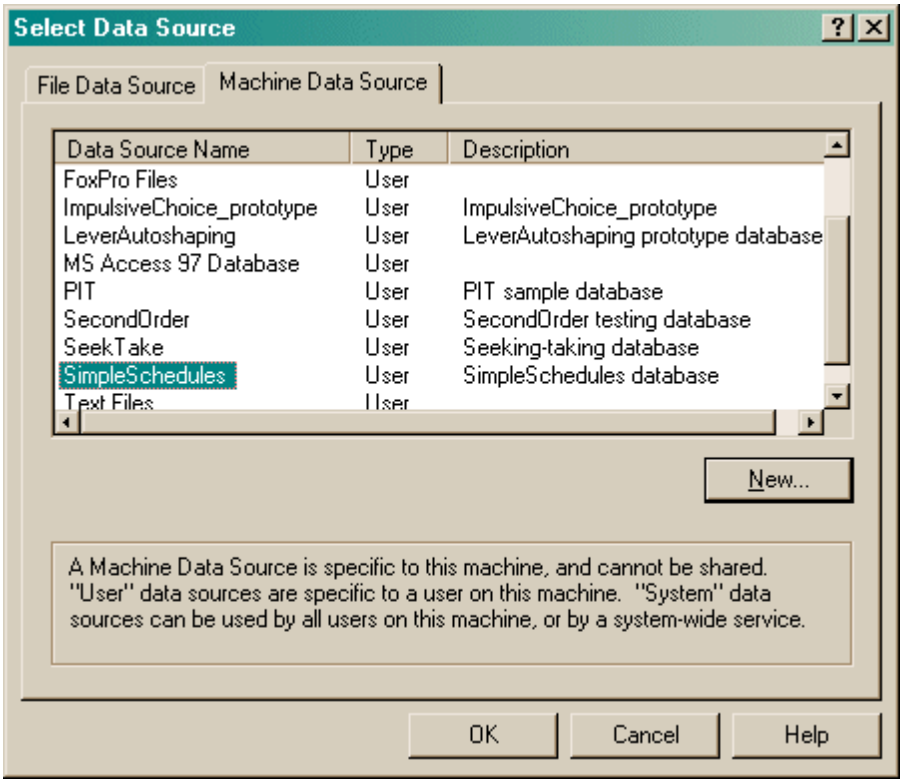
For concurrent schedules, you may implement a **changeover delay (COD).** If the COD is 2s and the subject presses the left lever, then responses on the right lever will not count towards the right-lever schedule for the next 2s. (Exactly the same applies in reverse - left-lever presses are ignored for 2s following any press on the right lever.) Note:

1. Responses may create a COD without counting towards the schedule (e.g. left - right - left: if the right lever-press occurs soon enough after the first left press, then that right press won't count towards the schedule, but it will set up a COD that may prevent the next left press from counting towards the schedule).
2. Time-based schedules still run during a COD (so, for example, RI schedules can still set up reinforcement, though that reinforcement cannot be collected until the COD has elapsed).

Choose whether or not to use the **traylight.**

Choose whether or not to **disable the 10-second safety timer on pumps.** This safety timer should be enabled for all animal intravenous infusion work. Disabling it allows infusion times longer than 10 seconds. Use caution if you disable it.

To pick an ODBC database **in advance** of finishing, click *Pick* and you will be offered the ODBC Data Source picker (below). Your choice will be recorded and will apply to this subject from now on (or until you specify a different source).



If you don't specify an ODBC data source now, or you delete the value in the "ODBC data source name" box, you'll be asked to choose when the task ends (and that choice will only apply to the session in progress).

1.5 Notes on reinforcement timing

It is possible to cause reinforcement conflicts in this task. For example,

- You can respond on two schedules which share a reinforcement device, such that both schedules want to use the reinforcement device simultaneously.
- You can respond more rapidly than your reinforcement device allows (e.g. if you have FR1 for a 7-s pump infusion with no timeout, you can respond again while the pump is still pumping from your first response).

Exactly the same is true if your reinforcement is delayed from the response that caused it (though it's much harder to perceive what's going on).

SimpleSchedules *ignores* requests for reinforcement for devices that are busy. It records in the event log (text-based and ODBC) whether a scheduled reinforcement was *actually* given.

What happens when you request a timeout on a delayed-reinforcement schedule? Should the timeout occur at the time of the response, or at the time of the reinforcement? Well, a timeout is to stop you responding, so it should occur at the time you respond. This is what SimpleSchedules does.

What happens when your schedule wants to reinforce, and to give you a timeout, but your reinforcement device is busy? SimpleSchedules will not reinforce (because the device is busy reinforcing you anyway) but it *will* implement your timeout.

1.6 Version for antique levers

Nearly all retractable/extendable operant chamber levers on the market are controlled by a single (output, from the computer's point of view) line. When the line is on (1), the lever extends and stays extended for as long as this control line is on. When the lever is off (0), it retracts. In addition, there is a response (input) line: 1 = lever depressed, 0 = lever not depressed.

However, some old (1980s?) levers from Campden Instruments, which are easily recognized because they require mains voltage (in the UK, 240 V AC) - and therefore require considerable respect when installing and handling them! - operate differently. They have the following control system (Julie Gill and David Maul, Campden Instruments, personal communication, June 2008):

- each lever has a *response* line (input): 1 = lever depressed, 0 = lever not depressed
- there is also a *lever position* line (input): 1 = lever retracted, 0 = lever extended
- and there is a *lever motor* line (output): this is normally held at 0, but a 40-100ms pulse to 1 (and then back to 0) latches the lever motor on. If the lever was extended, this pulse causes it to retract; if it was retracted, the pulse causes it to extend.

The **SimpleSchedules_AntiqueLevers.exe** program is a separate executable from the usual **SimpleSchedules.exe** program, and it supports these old levers. In all other respects it is identical to the main task. The required devices (q.v.) are slightly different; this is deliberate, so you can't accidentally run the wrong version of the task and not notice. There is also a message on the main window to announce the fact that you are using the "antique levers" version.

The program does not support the levers in a very sophisticated way, but as follows:

- Whenever the program wishes the levers to change state, it checks the current state by asking the lever. If the current state is what the computer thought it was, then it pulses the motor to change the state. If it wasn't, then the lever isn't working properly; it flags this on screen and in the text log file (and doesn't pulse the motor, since the lever is already in the target state).

Index

- S -