SameOpposite

A Whisker client

by Rudolf Cardinal

www.whiskercontrol.com

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

SameOpposite

Printed: February 2018 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 SameOpposite

1.1 About SameOpposite

Purpose

Same/opposite side reaction time task.

Citing SameOpposite

Please cite as, for example:

(in text)
The task was implemented in the SameOpposite program (Cardinal, 2007) using the Whisker control system (Cardinal & Aitken, 2001).

(in bibliography)
Cardinal RN (2007). SameOpposite (version 0.1), computer software [www.whiskercontrol.com]. Cambridge University Technical Services Ltd, Cambridge, UK.
Cardinal RN, Aitken MRF (2001). Whisker (version 2), computer software [www.whiskercontrol.com]. Cambridge University Technical Services Ltd, Cambridge, UK.

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

Version history

- v0.1 (3-7 July 2007). First version written.
- v1.0 (5 Dec 2008). Bug fix: spurious error message ('Syntax error message received unknown command "awaiting"') removed (a status message had a semicolon in; the actual problem was in the clientlib). // Traylight goes off when rear panel entered, if it was on.
- v2.0 (12 Jan 2009). Server default changed from "loopback" to "localhost" (Windows Vista compatibility and more general standardization).
- v2.1 (25 Mar 2009). Support for using different holes in a 9-hole box environment; **change to claimed device names** (from HOLE_0-HOLE_4 to HOLE_0-HOLE_8, and similarly for STIMLIGHT).
- v2.2 (1 May 2009). That was a silly choice of names, as the FiveChoice task uses HOLE_0 (etc.). So we will now use NHB_HOLE_0 (to _8) and NHB_STIMLIGHT_0 (to _8). ("NHB" for "nine-hole box".)

1.2 Required devices

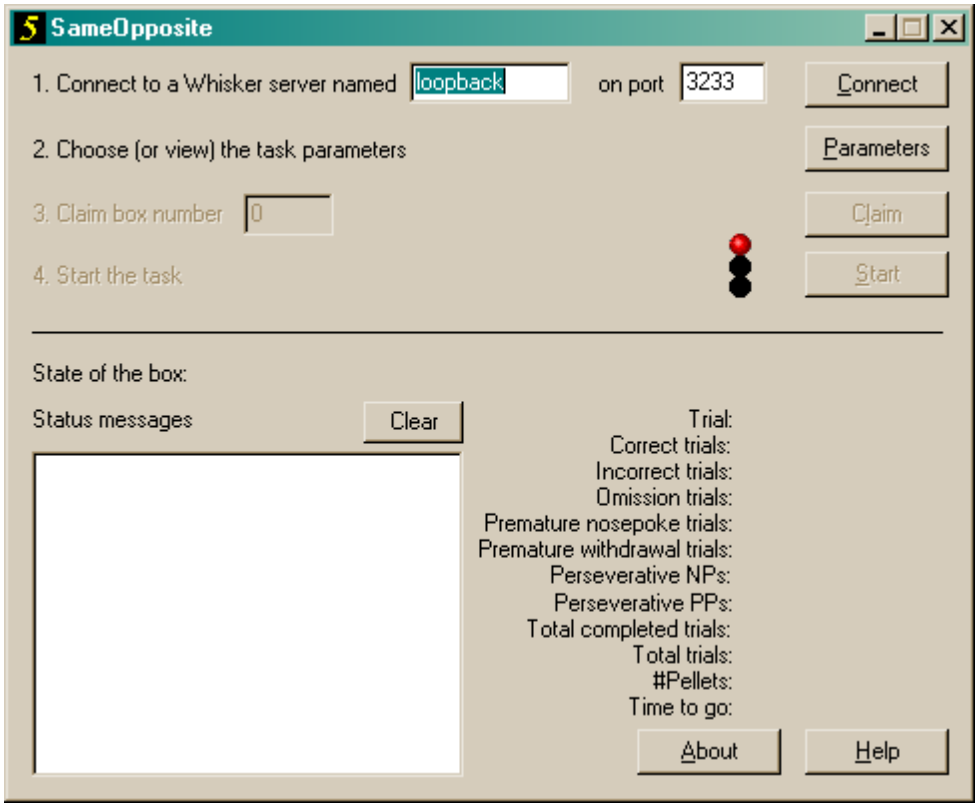
The program requires to claim devices in groups named **box0, box1, box2...** with device names as listed below in bold:

```
# ---------------- Box 0 definition
# INPUTS
line    0       box0    REARPANEL
line    1       box0    NHB_HOLE_0
line    2       box0    NHB_HOLE_1
line    3       box0    NHB_HOLE_2
line    4       box0    NHB_HOLE_3
line    5       box0    NHB_HOLE_4
line    6       box0    NHB_HOLE_5
line    7       box0    NHB_HOLE_6
line    8       box0    NHB_HOLE_7
line    9       box0    NHB_HOLE_8

# OUTPUTS
line    24      box0    HOUSELIGHT
line    25      box0    PELLET
line    26      box0    TRAYLIGHT
line    27      box0    TONE
line    28      box0    NHB_STIMLIGHT_0
line    29      box0    NHB_STIMLIGHT_1
line    30      box0    NHB_STIMLIGHT_2
line    31      box0    NHB_STIMLIGHT_3
line    32      box0    NHB_STIMLIGHT_4
line    32      box0    NHB_STIMLIGHT_5
line    32      box0    NHB_STIMLIGHT_6
line    32      box0    NHB_STIMLIGHT_7
line    32      box0    NHB_STIMLIGHT_8

# ... and so on for all your boxes
```

Please ensure that these devices are available and listed in the device definition file in use by the server. (The snippet above shows an extract from a typical definition file.)
The **TONE** device is the one used as a tone or a white-noise distractor. (TONE/NOISE DISTRACTION IS NOT CURRENTLY A FEATURE OF THIS TASK.)

1.3 Using the task

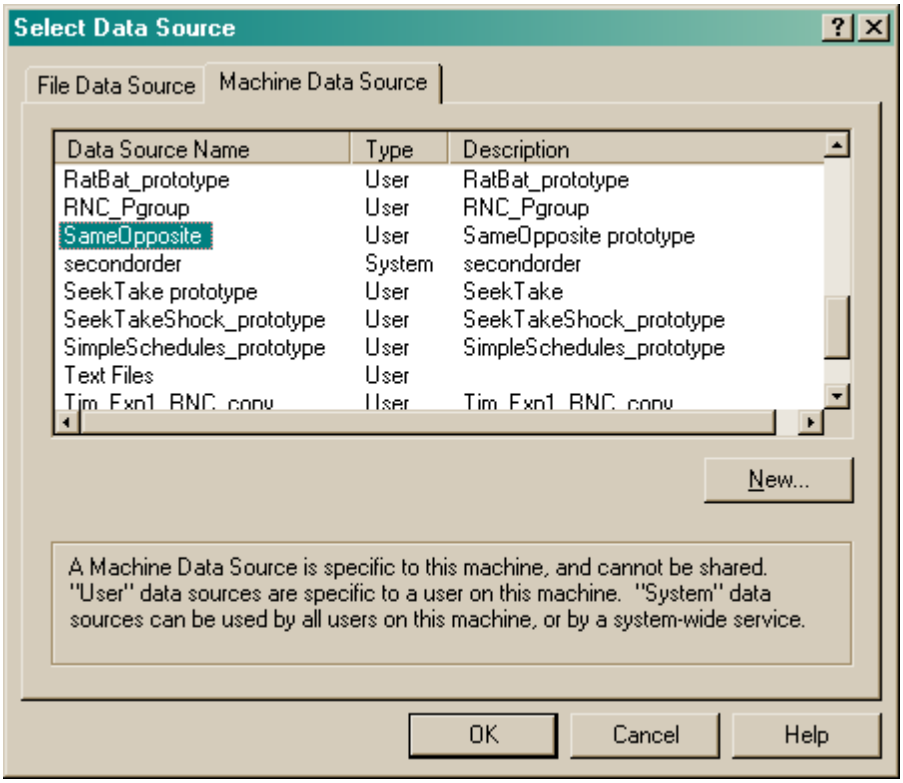
When you run the task, the main screen looks as follows:



"NPs" means nosepokes (responses at the front holes); "PPs" means panel pushes (responses at the food alcove); see the Task Design.

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 Running multiple boxes

The SameOpposite program controls **one** box in which the subject performs the same/opposite task.

To run the task with **multiple boxes**, you simply need to run **multiple copies of the SameOpposite program**.

For example, suppose you have 6 five-choice chambers (boxes), and you have started the Whisker server. You could run copy 1 of the SameOpposite program, and have it claim box 1, load subject 1's configuration, and start it. You could then run a second copy of the SameOpposite program at the same time, claiming box 2, loading subject 2's configuration, and starting it... and so on. You can also run other tasks (such as the FiveChoice task) at the same time.

1.5 Task design and trial overview

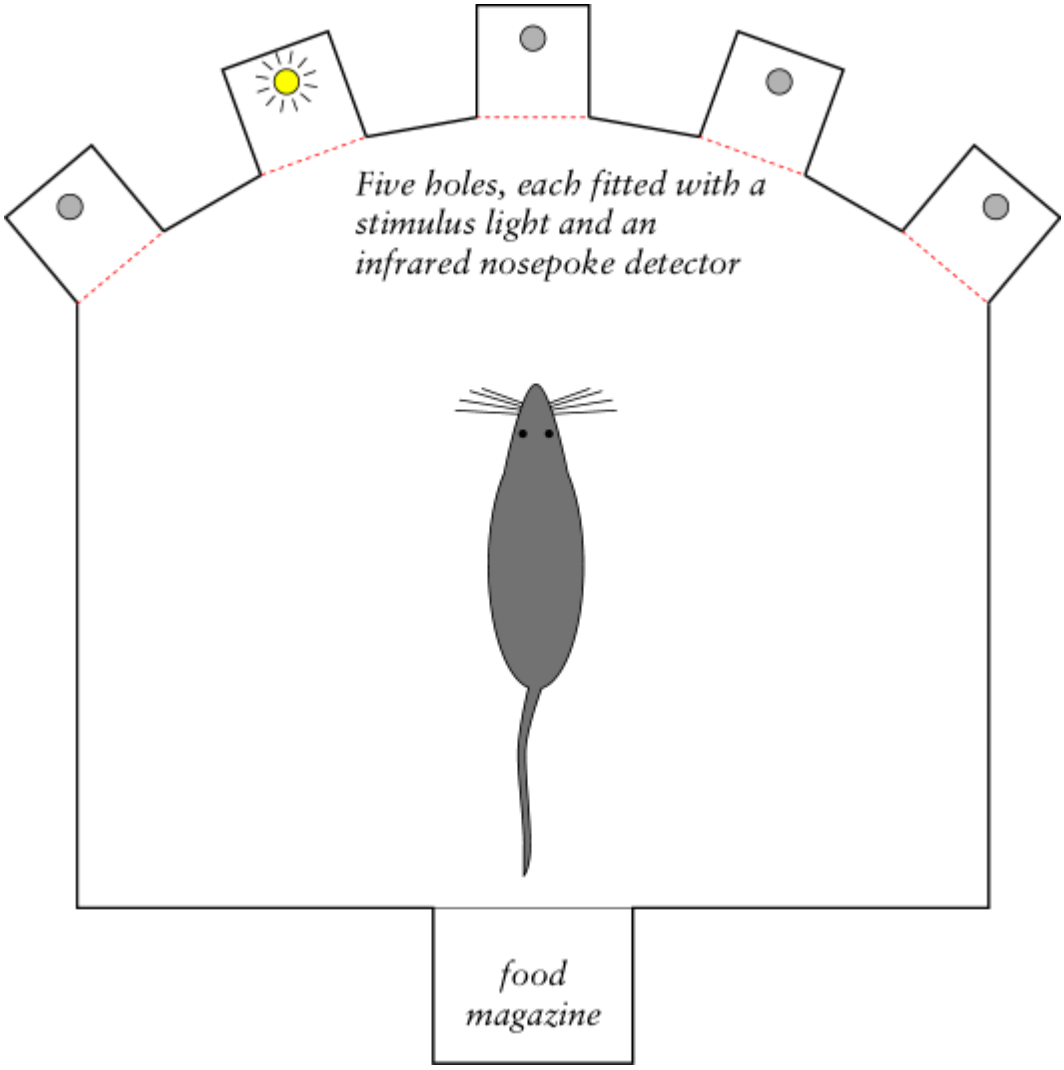
History

Carli M, Evenden JL, Robbins TW (1985). Depletion of unilateral striatal dopamine impairs initiation of contralateral actions and not sensory attention. *Nature* **313:** 679–682.

This task implements procedure (1) of Carli *et al.* (1985).

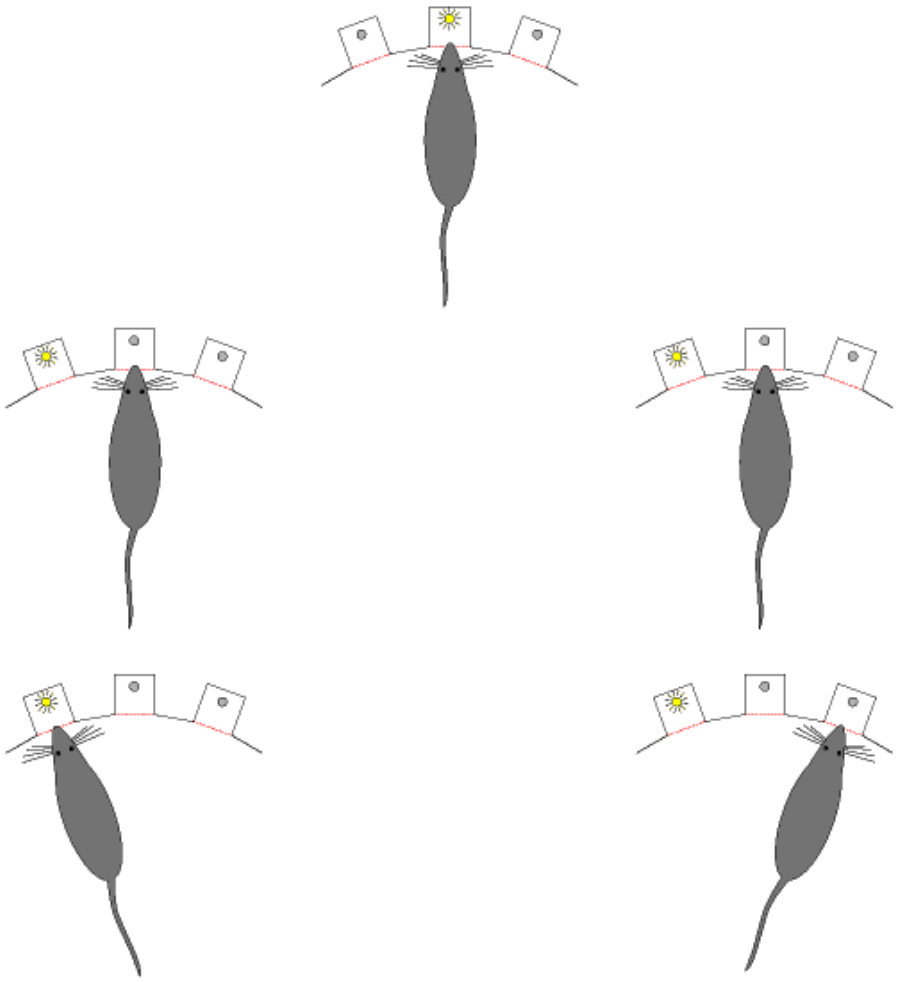
Apparatus

This task runs in a standard "nine-hole box" or "five-choice" operant chamber, shown below. However, it only uses three central holes (the centre hole and a definable pair on either side) and the food magazine.



Task outline

The trial proper begins when the subject is nosepoking in the centre hole.



In the SAME task, the rat must respond to the same side as the light. Here, the target is on the left and the rat responds to the left, so this would be a correct response.

In the OPPOSITE task, this would be an incorrect response.

In the SAME task, the rat must respond to the same side as the light. Here, the target is on the left and the rat responds to the right, so this would be an incorrect response.

In the OPPOSITE task, this would be an correct response.

See the trial details for a full description.

1.6 Trial details

Trial format

Think of the task as existing in a number of *states*, because that's how the program implements it. Incidentally, this is an exercise in mental discipline: if you can't specify a state table like this, having thought about what should happen in response to all possible events in any state, then you can't program the task properly.

Blue text shows the states that a successful subject cycles through. Green shows ways that an

unsuccessful subject can rescue itself and lead onto the blue path. Red shows errors.

State name	Description	State of the box	Consequence of nosepoking in the centre hole.	Consequence of nosepoking at the left or right hole.	Consequence of nosepoking at the rear (food) panel	Consequence of time passing
NOTSTARTED	Not started yet. Awaiting experimenter to start task.	Darkness.	-	-	-	-
PLEASE_EAT	**Task begins here.** (For the first trial, a free pellet is given and the trial number is initially set to 0.) Rat must respond to rear panel to initiate the trial. This point may also be reached by successful completion of a trial.	Houselight on. Traylight on (if used).	Score **perseverative centre response** and go to TIMEOUT.	Score **perseverative side response** and go to TIMEOUT.	Go to PLEASE_CENTRE.	-
TIMEOUT_OVER_PLEASE_REINITIATE	Punishment timeout over. Rat must respond to rear panel to initiate new trial.	Houselight on. Traylight on (if used).	Score **perseverative centre response** and go to TIMEOUT.	Score **perseverative side response** and go to TIMEOUT.	Go to PLEASE_CENTRE.	-
PLEASE_CENTRE	**Trials begin here. Trial number incremented.** Waiting for subject to respond in the centre hole.	Houselight on. Centre hole lit.	Go to CENTRED.	Score **premature side response** and go to TIMEOUT.	Recorded; no consequence.	-
CENTRED	Subject currently responding in centre hole. Brief pause before target is illuminated.	Houselight on.	If rat withdraws, score **premature centre withdrawal** and go to TIMEOUT.	Shouldn't be possible, unless equipment failure or unusually shaped rat. But score **premature side response** and go to TIMEOUT.	Shouldn't be possible, unless equipment failure or unusually shaped rat. But score **perseverative panel push** and go to TIMEOUT.	Go to TARGET_ON.
TARGET_ON	Side target illuminated. Subject may respond.	Houselight on. Target hole lit.	Score **perseverative centre response** and go to TIMEOUT.	If correct, deliver food and go to PLEASE_EAT. If incorrect, go to TIMEOUT.	Score **perseverative panel push** and go to TIMEOUT.	Go to TARGET_OFF.
TARGET_OFF	Side target has gone out, but subject may still respond.	Houselight on.	Score **perseverative centre response** and go to TIMEOUT.	If correct, deliver food and go to PLEASE_EAT. If incorrect, go to TIMEOUT.	Score **perseverative panel push** and go to TIMEOUT.	**Omission.** Go to TIMEOUT.
TIMEOUT	Rat is being punished.	Darkness.	Score **perseverative centre**	Score **perseverative side response** and restart TIMEOUT.	Score **perseverative panel push** and	Go to TIMEOUT_OVER_PLEASE

			response and restart TIMEOUT.		restart TIMEOUT.	_REINITIATE.
FINISHED	Rat finished the task	Darkness.	-	-	-	-
ABORTED	User aborted the task	Darkness.	-	-	-	-

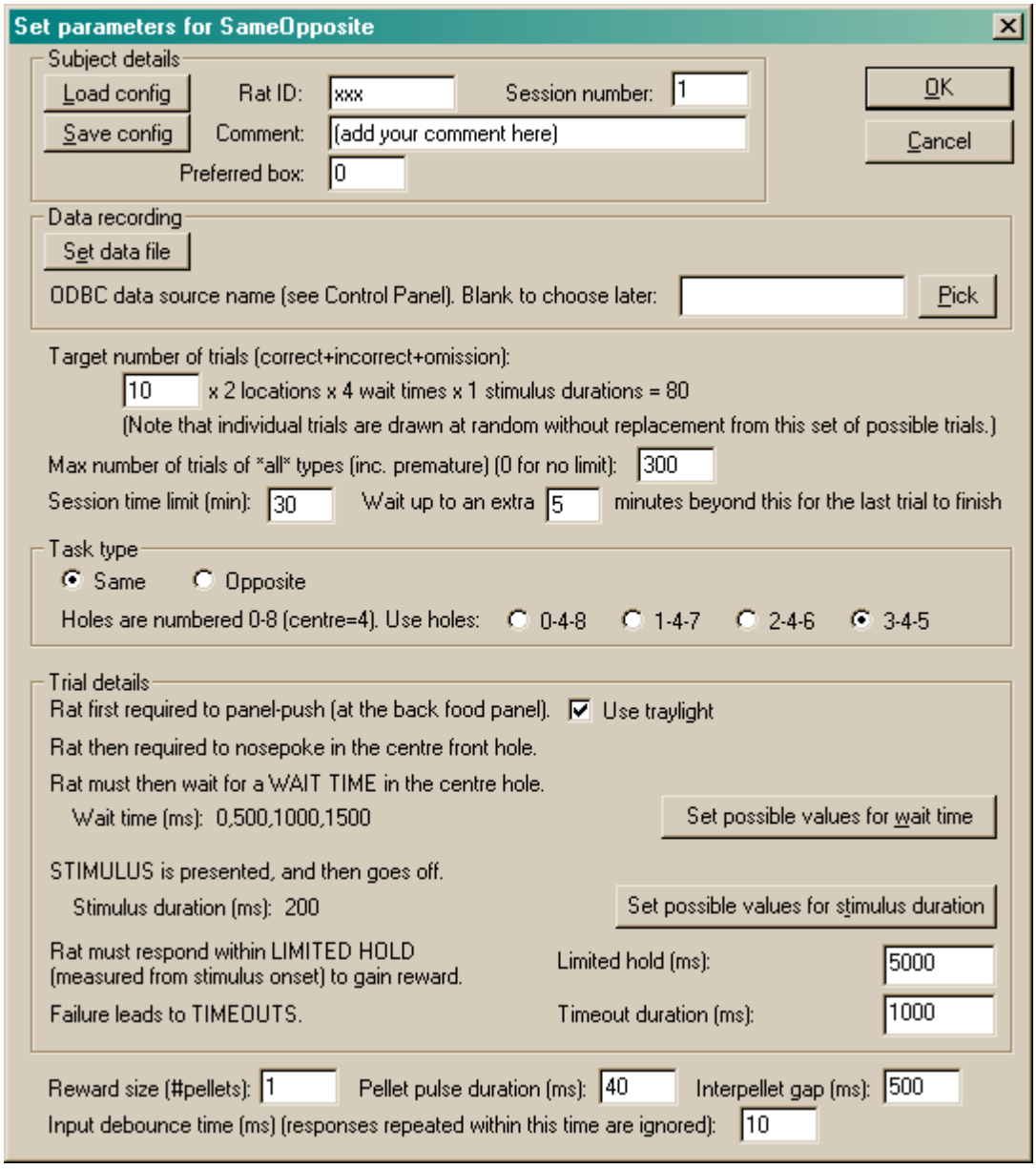
The task records every response with its location and the state (phase) the box was in when the response was made.

Additionally,

- there can be an overall time limit for the session;
- there can be a trial limit for the session.

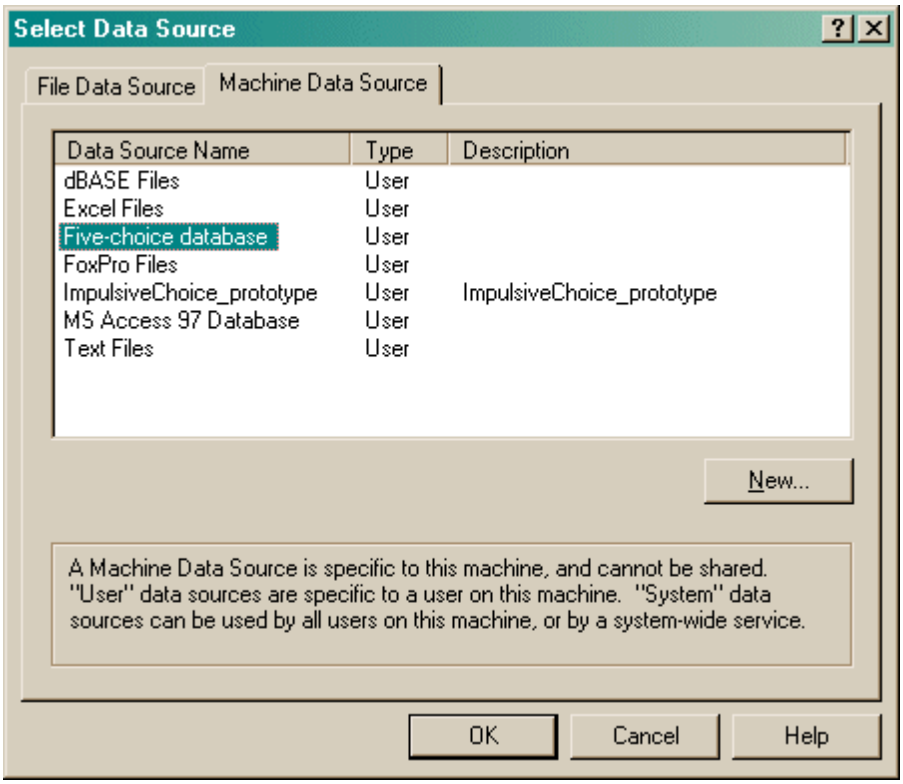
1.7 Parameters

The parameters dialogue box looks like this:



- **SUBJECT DETAILS.** Here, you can enter your subject's ID and a comment or description, choose the default box (operant chamber) number that this subject usually runs in, and set its session number. You can also **load** and **save** the whole configuration (with the subject's details and all the information you can see on this screen). We recommend that you set up **one configuration file per subject.** If you do so, then you would set up the subject's task parameters once, then save the configuration file. Every time you reload the configuration, on subsequent sessions, the subject's parameters will be correctly recalled and the program will automatically increase the session number by one. Unless you want to change the task parameters, this makes re-running a subject a very rapid procedure.
- **DATA RECORDING.** The program stores its results in two places: a text file, and a relational database. Here, you can specify the **filename** of the text file. Note also that a unique filename for the text file is generated whenever you load a subject's configuration, so you shouldn't routinely need to enter this. You can also specify the **ODBC** name of the database (see the Results section

of this help for more information). Click "Pick" to choose from a list of ODBC databases configured on your computer (see below). Your choice will be recorded and will apply to this subject from now on, or until you specify a different database. If you don't specify a database now, or you delete the value in the "ODBC data source name" box, you'll be asked to choose a database when the task ends (and that choice will only apply to the session in progress).



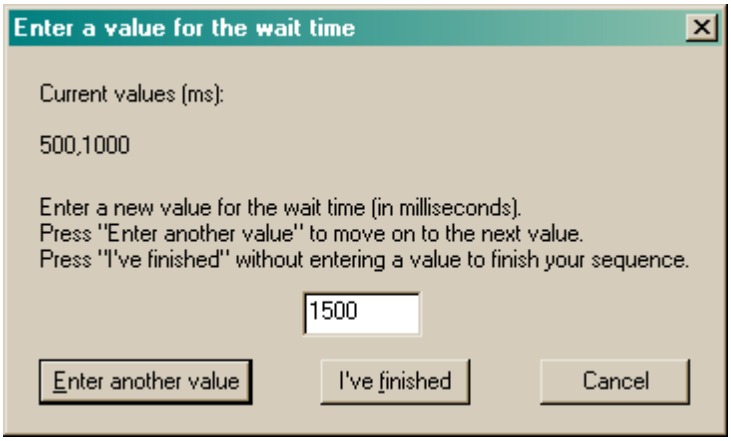
- **Target number of trials.** Set the maximum number of correct/incorrect/omission trials. Since these should be counterbalanced for stimulus duration, wait time, and target size, you specify a **multiplier;** the program calculates the target number of trials as shown. The program will terminate when this target has been reached. Trials that do not progress far enough to count as correct, incorrect, or omission trials do not count towards this limit. Once the full set of trials is created (as determined by the multiplier you set), the program draws individual trials without replacement from that set (see drawing without replacement).
- **Maximum number of trials of all types.** You can set a further optional limit here of the maximum number of trials of *all* types, including "premature" trials (see Task description and Trial details). Enter 0 for no such limit.
- **Session time limit.** Set the maximum session time. The program will terminate when this time limit is reached, after finishing any trial currently in progress.
- **Session extra time.** After the session time expires, the program will normally wait (in "extra time") for the current trial to finish. You can specify how much "extra time" to allow for this here. After the "extra time" expires, the program will terminate immediately, interrupting any trial that is still in progress.

- **TASK TYPE.** Choose either "same" or "opposite" to define the task in use. See the Task description.
- Choose also **which holes to use** for the task. This will be the very centre hole (hole 4, since the holes are numbered 0-8), plus a symmetrical pair of holes to the side (ranging from the extreme side holes and the centre, 0-4-8, to the three central holes, 3-4-5).

- **TRIAL DETAILS.** For full details of the task, see the Task description and Trial details.
- The task begins by requiring the subject to respond at the rear panel (the food alcove). Tick **use**

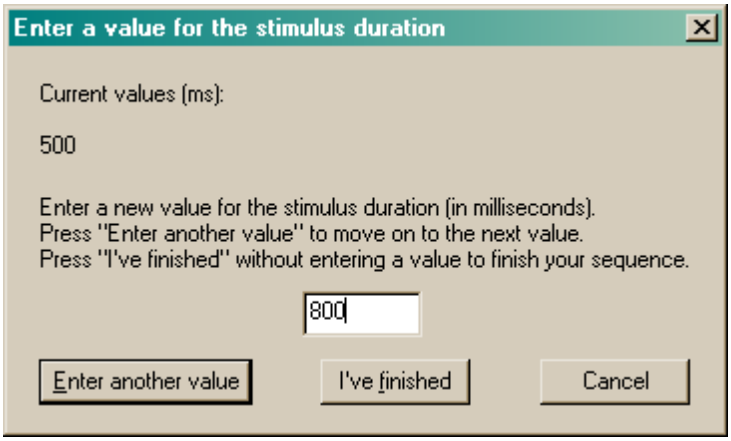
traylight if you wish the traylight (the light within the rear food alcove) to be illuminated when the subject is required to respond there.

- The trial then requires the subject to nosepoke in the centre hole, and to wait with its head in this hole for at least a specified **WAIT TIME.** You specify the possible wait times, in milliseconds (ms). You can specify one wait time (e.g. 1000 ms) or a set of possible wait times (e.g. 500, 1000, 1500, and 2000 ms). To specify values for the wait time(s), click "Set possible values for wait time":



Every time you click "Enter another value", your previous value is added to the list (shown near the top) and you can enter another. Click "I've finished" when you've finished entering the possible values you want. **Note also** that you can enter values twice, influencing their likelihood of selection. For example, if you enter "1000, 2000, 2000", then you will get wait times of 1000 or 2000 ms, but you will be twice as likely to get a 2000 ms wait time than a 1000 ms wait time.

- After the initial pause, the **STIMULUS** is presented. You can specify the stimulus duration in exactly the same way as the initial pause. A similar dialogue is shown for "Set possible values for stimulus duration":



- You can specify the **LIMITED HOLD** period. This is the time the program will wait after the stimulus *begins* before it abandons the trial, scoring it as an omission, if the subject has not responded.
- Finally, task failure in a variety of ways (see Task design and Trial details) can lead to punishments in the form of **TIMEOUTS**. Specify the timeout duration.

The last set of options concern miscellaneous aspects of the task.

- **Reward parameters.** Choose the number of pellets to be used as a reward, and the timing used for your particular brand of pellet dispenser (e.g. if you would like 2 pellets per reward, delivered

500 ms apart, and your pellet dispenser likes a 45-ms electrical pulse to dispense a single pellet, then specify 2, 45, and 500 in the boxes here).

- **Debouncing.** Electromechanical devices often "bounce": if you press a lever, a short burst of electrical pulses are sent, as physical vibration in the lever turns the device on and off very rapidly. You normally want to ignore this. Typically, you might set a "debounce time" of 10 ms; this would mean that any responses that are repeated within 10 ms of a previous response on the same device (e.g. lever, panel detector) are attributed to electromechanical bounce and ignored. Mammals generally cannot make a voluntary action twice within 10 ms! If you don't debounce inputs, strange things can happen; for example, we saw a problem as a result of a mechanical food alcove switch bouncing; this registered erroneously as perseverative rear panel pushes when it was really a single trial-starting response.

1.8 Randomness, pseudorandomness, drawing without replacement

Suppose I wish to pick a series of numbers from 1 to 6.

I could pick them **at random**. I could do this by rolling a true die. Every time I rolled the die, I would obtain a number from 1-6 with equal probability. The probability of obtaining a "1" would be 1/6 (approximately 0.17); the probability of obtainin a "2" would be 1/6, and so on.

If I rolled the die 100 times, I would expect to get roughly 17 ones, roughly 17 twos, roughly 17 threes, roughly 17 fours, roughly 17 fives, and roughly 17 sixes. It is *possible* that I would get 100 sixes—but very unlikely (the probability of this is 1.53×10^{-78}, or one in six hundred thousand quintillion quintillion quintillion quintillion). But it is certain that I would not get exactly the same number of ones, twos, threes, fours, fives, and sixes (because you can't have six equal whole numbers adding up to 100). If I rolled the die 60 times, you'd expect about 10 of each number—but it's also pretty unlikely that I'd get *exactly* 10 of each number.

If I rolled the die like this, there is absolutely no way to predict the next number that will come up on each roll.

So much for randomness.

If I want a series of 60 numbers and I want to ensure that I end up with equal numbers of ones, twos, threes, fours, fives, and sixes, I could simply take them in a **predictable order**: 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6. This gives me the distribution I want (ten of each number), but the sequence is anything but random, and is highly predictable. If you know the last number that came up, you have an extremely good idea of what the next number would be.

So much for total predictability.

To obtain a reasonable combination of unpredictability and obtaining an overall equal distribution of numbers, we could use a **pseudorandom** technique called **drawing without replacement**, or **draw-without-replacement**, sometimes shortened to **draw-w/o-replacement** or just **DWOR**. Imagine we would like a sequence of 60 numbers, each in the range 1-6, more or less randomly, but ensuring that we get 10 of each number. We could write the number "1" on ten pieces of paper, the number "2" on ten more pieces of paper, and so on. We could then put all 60 pieces of paper in a hat, shuffle them, and draw them out in sequence, *without replacing them*. We'd then be guaranteed ten of each type, but the local order would be fairly random. It would not be totally random—if we've drawn out 10 ones, 10 twos, 9 threes, 10 fours, 10 fives, and 10 sixes, then we can be absolutely certain that the last number out of the hat will be a 3. However, we'd have to remember the numbers that had come out so far.

So it's impossible to have a completely random order and to guarantee a certain distribution—but

drawing without replacement is a good way to approximate randomness while guaranteeing a certain distribution.

There are several ways to draw without replacement. I've just described a situation in which 10 copies of each number (1-6) are put into the hat, shuffled, and drawn individually for 60 trials. It's possible (but again extremely unlikely!) that the first ten trials would all be ones, the next ten all twos, and so on. If we wanted to guarantee that *in every six consecutive trials* we will see each possible digit (1-6) once, we should do something different. We should write the number "1" on one piece of paper, the number "2" on a second piece of paper, and so on, up to six pieces of paper. We should then put the six into the hat, shuffle them, and draw them out (without replacement) for the first six trials. We should then put the six pieces of paper back in the hat, shuffle, and repeat the process for the next six trials, and so on. This process gives *less randomness* (if you know just the first five trials in a set of six, then in principle, you can have perfect knowledge of the sixth number to be drawn) but *better control over the local distribution* of numbers.

We've just seen two examples in which a **list** of six numbers (1, 2, 3, 4, 5, 6) are put into a hat and drawn for 60 trials. In the first, we put ten copies of each item in the list into the hat (giving 60 pieces of paper in total) and drew them. I call this a **draw-without-replacement (DWOR) multiplier** of 10. In the second, we put one copy of each item in the list into the hat, drew them until we'd run out of numbers (after six trials), then put them all back into the hat. I call this a DWOR multiplier of 1.

We've just seen the concept of a *list* of possibilities, which is multiplied by a *DWOR multiplier* to give a set of options that are then *drawn at random without replacement* from a hat; when the hat is empty, we restart the process.

The bigger the DWOR multiplier, the closer the DWOR technique comes to total randomness (if the DWOR multiplier were infinitely large, they are exactly the same process). The smaller the DWOR multiplier, the closer the technique is to total predictability.

This program offers the DWOR technique as a way of selecting possible values for various parameters.

1.9 Results

The program always stores results in two places. One is a human-readable text file. The other is a database. (You choose the name of this file in the main parameters dialogue box, and you can choose the database here as well.)

1.9.1 Text-based results file

A sample results file is shown below. The configuration information is shown first; the results follow. (There aren't very many results, because I got bored creating the file.) The results section is shown in bold, with **trial summary** information followed by **individual response** information.

I encourage you to think of this file as a backup. The database contains all this information and can be used to retrieve both simple and highly detailed information about a subject's performance.

```
SAME/OPPOSITE SIDE REACTION TIME TASK -- SUMMARY FILE
_____________________________________________________

SameOpposite v0.1 - release build compiled on Jul  6 2007 at 19:16:24

IDENTIFICATION
```

```
Rat:                    xxx
Session:                2
Date/time code:         07-Jul-2007 (18:15)
Comment:                (add your comment here)

Box:                    0
Client computer name:   EGRET
Server computer name:   loopback
Summary file name:      xxx-07Jul2007-1828-SameOpposite-summary.txt
Default ODBC database:
Preferred box number:   0

CONFIGURATION

Target number of trials: multiplier =     : 10
  (x 2 sides x 4 wait times x 1stimulus durations = 80 trials as a target)
Max trials of all types (0=no limit)      : 300
Session time limit (min)                  : 30
... extra time to finish current trial    : 5
Task type                                 : SAME
Use traylight?                            : Y
Wait time values (ms)                     : 0,500,1000,1500
Stimulus duration values (ms)             : 200
Limited hold duration (ms)                : 5000
Timeout duration (ms)                     : 1000
Number of pellets                         : 1
Pellet pulsing time (ms)                  : 40
Interpellet gap (ms)                      : 500
Input debounce time (ms)                  : 10

SUMMARY DATA

Started at:         07-Jul-2007 (18:30)
Finished at:        07-Jul-2007 (18:31)

TOTALS (master box)

Number of trials                : 5
Number of correct responses     : 1
Number of incorrect responses   : 5
Number of omissions             : 2
Number of 'valid' trials        : 5
Number of pellets earned (total) : 2


TRIAL AND RESPONSE DATA

Rat,Box,Trial,SessionBeganAt_ms,IntendedWaitTimeMs,IntendedStimulusDurationMs,
IntendedTargetOnLeft,TrialBeganAt_ms,PleaseCentreBeganAt_ms,CentredAt_ms,
LatencyToCentre_ms,TargetOnAt_ms,TargetOffAt_ms,HeadOutOfCentreAt_ms,
HeadIntoSideHoleAt_ms,RewardedAt_ms,TimeoutBeganAt_ms,PleasePushBeganAt_ms,
RearPushAt_ms,PerseverativeCentreNosepokes,PerseverativeSideNosepokes,
PrematureCentreWithdrawals,PrematureSideNosepokes,PerseverativeRearPanelPushes,
Initiated,Centred,Waited,WithdrawnFromCentre,Responded,RespondedCorrectly,
RespondedIncorrectly,Omission,Rewarded,PunishedWithTimeout,OfferedHole,ChosenHole,
ResponseLatency_ms,ExperiencedTimeout_ms,CollectionLatency_ms
xxx,0,1,959187882,1500,1239824,0,959192758,959192758,959196124,3366,959197631,4294
967295,4294967295,959201482,4294967295,959201491,959202494,959205564,0,0,0,0,0,1,1
,1,0,1,0,1,0,0,1,3,1,3851,1003,4294967295
xxx,0,2,959187882,500,1239964,0,959205564,959205564,959208272,2708,959208779,42949
67295,4294967295,4294967295,4294967295,959213790,959214850,959224834,0,1,0,0,0,1,1
,1,0,0,0,64,1,0,1,3,-1,4294967295,1060,4294967295
```

```
xxx,0,3,959187882,1000,1239964,0,959224834,959224834,4294967295,4294967295,4294967
295,4294967295,4294967295,4294967295,4294967295,959229403,959230405,959232623,0,0,
0,1,0,1,0,0,0,0,0,64,0,0,1,-1,-1,4294967295,1002,4294967295
xxx,0,4,959187882,500,1239964,0,959232623,959232623,959235719,3096,959236275,42949
67295,4294967295,959239146,959239149,4294967295,959239149,959242537,0,0,0,0,0,1,1,
1,0,1,1,64,0,1,0,3,3,2871,4294967295,3388
xxx,0,5,959187882,0,1244276,0,959242537,959242537,959245375,2838,959245385,4294967
295,4294967295,4294967295,4294967295,959250395,959251402,4294967295,0,0,0,0,0,1,1,
1,0,0,0,64,1,0,1,3,-1,4294967295,1007,4294967295

Rat,Box,Trial,ResponseNum,Location,Phase,TimeAbsolute_ms,TimeInSession_ms,
TimeInTrial_ms,InterestingTime_ms,ResponseType
xxx,0,1,0,99,1,959192758,4876,0,0,1
xxx,0,1,1,2,3,959196124,8242,3366,3366,2
xxx,0,1,2,1,5,959201482,13600,8724,3851,5
xxx,0,1,3,99,2,959205564,17682,12806,4073,1
xxx,0,2,4,99,2,959205564,17682,0,0,1
xxx,0,2,5,2,3,959208272,20390,2708,2708,2
xxx,0,2,6,3,7,959213846,25964,8282,56,8
xxx,0,2,7,99,2,959224834,36952,19270,11044,1
xxx,0,3,8,99,2,959224834,36952,0,0,1
xxx,0,3,9,1,3,959229392,41510,4558,4558,9
xxx,0,3,10,99,2,959232623,44741,7789,3220,1
xxx,0,4,11,99,2,959232623,44741,0,0,1
xxx,0,4,12,2,3,959235719,47837,3096,3096,2
xxx,0,4,13,3,5,959239146,51264,6523,2871,4
xxx,0,4,14,99,1,959242537,54655,9914,3388,6
xxx,0,5,15,99,1,959242537,54655,0,0,1
xxx,0,5,16,2,3,959245375,57493,2838,2838,2


Successfully wrote to database: ODBC;DSN=SameOpposite;DBQ=D:
\Whisker\CODE\clients\rnc - cambridge\SameOpposite\SameOpposite database
(prototype).mdb;DriverId=281;FIL=MS Access;MaxBufferSize=2048;PageTimeout=5;
```

1.9.2 Creating a new ODBC source

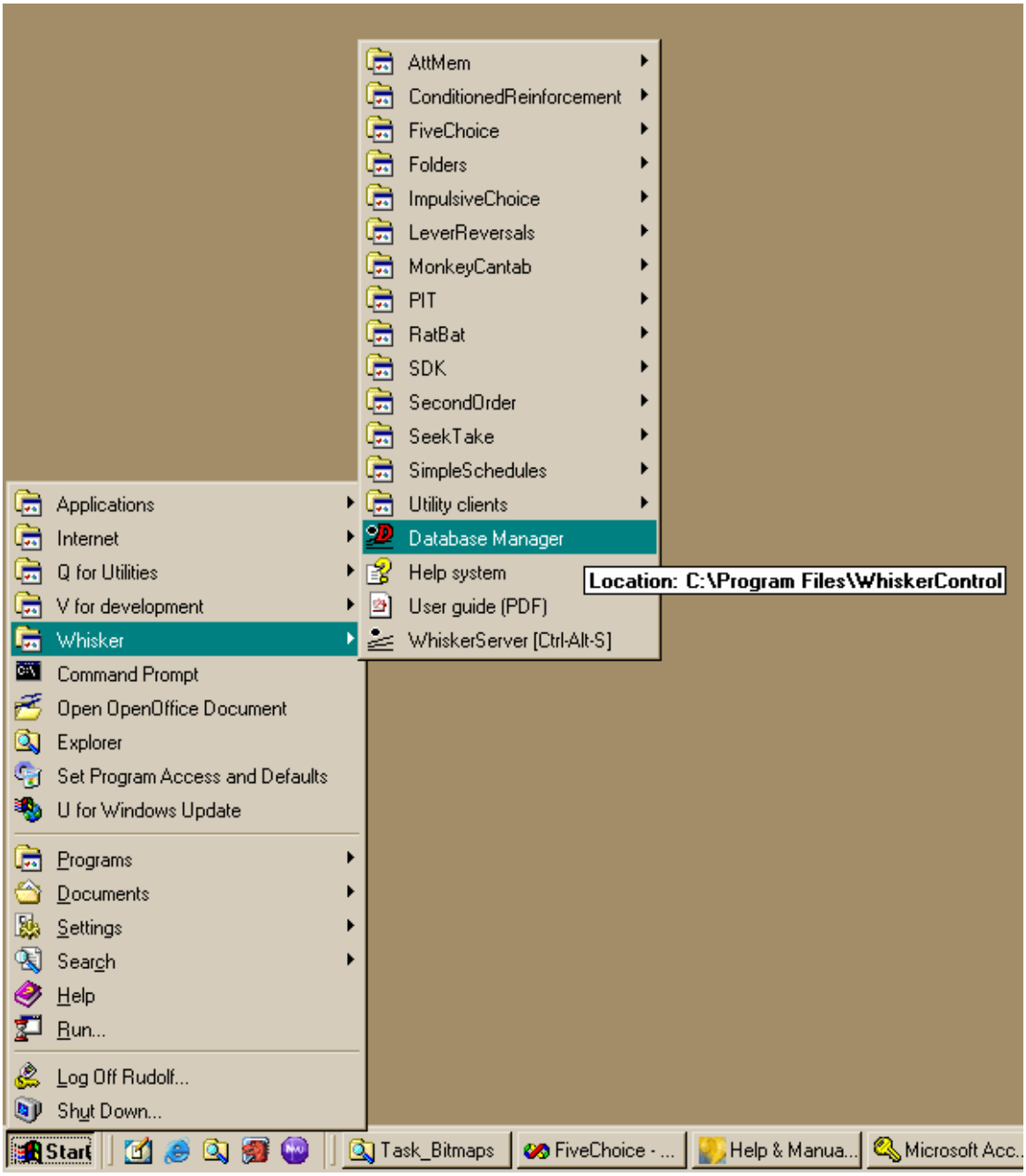
What happens if you're using SameOpposite for the first time, or you're starting a new experiment, and you need to set up a new ODBC (Open Database Connectivity) source? You should configure it via the **Whisker Database Manager** (Start → Whisker → Database Manager), or via **Control Panel → ODBC** [in Windows 2000, Start → Settings → Control Panel → Adminstrative Tools → Data Sources (ODBC)]. This section shows you various ways to achieve this.

Remember: you shouldn't use the supplied database without making a copy for yourself. (It will work, but if you ever uninstalled or reinstalled MonkeyCantab, this file might be replaced or lost. It is much safer to make your own copy and set up ODBC to use your copy.)

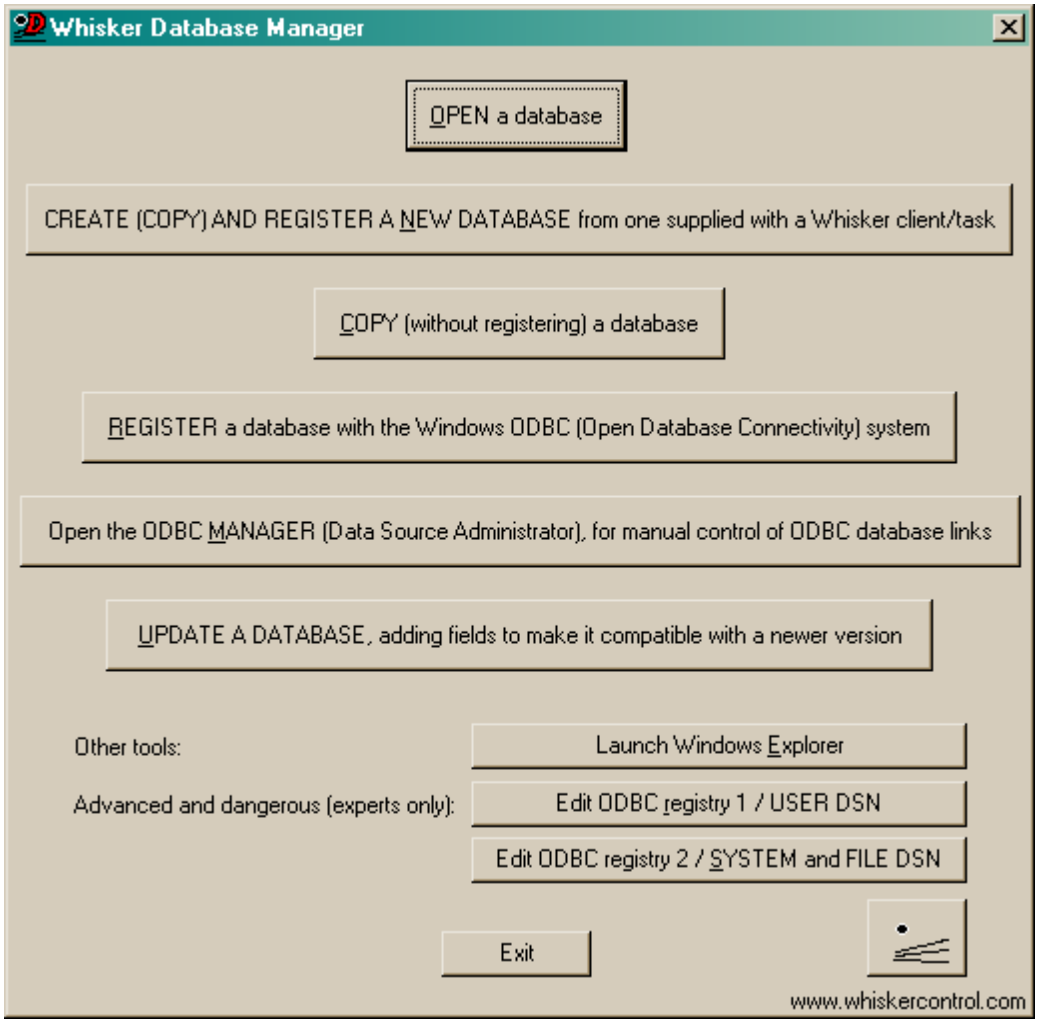
The procedure is:

1. **Make a copy of the supplied database to store *your* data in.**
2. **Register *your* copy with ODBC.**

The simplest way is to run the **Whisker Database Manager**:



You'll see the Database Manager:

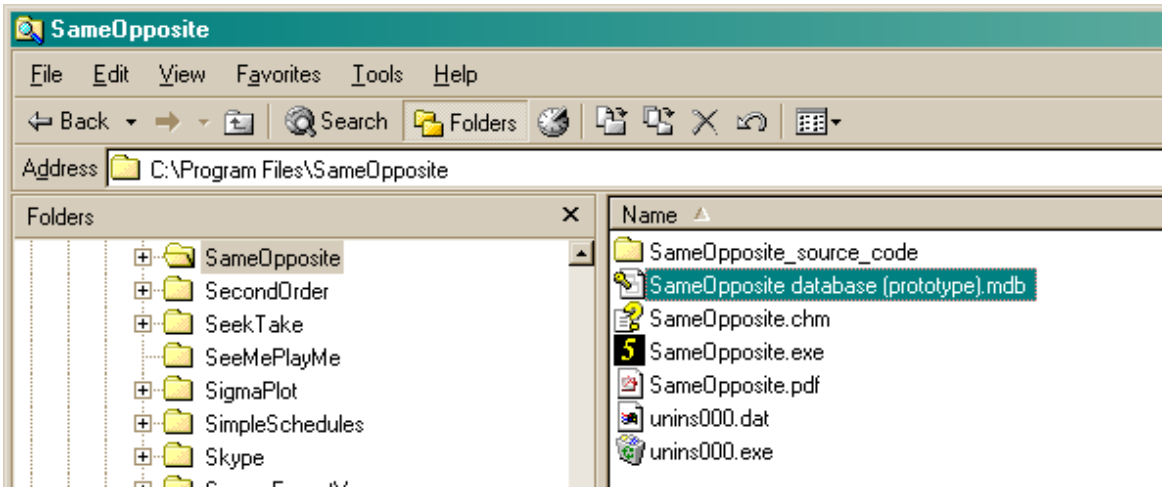


You can use this program to **CREATE (COPY) AND REGISTER** a database. Or you can **COPY** the supplied database and **REGISTER** your copy with ODBC separately. For full details of the Whisker Database Manager, see the **Whisker Help (Auxiliary Programs / Whisker Database Manager)**.

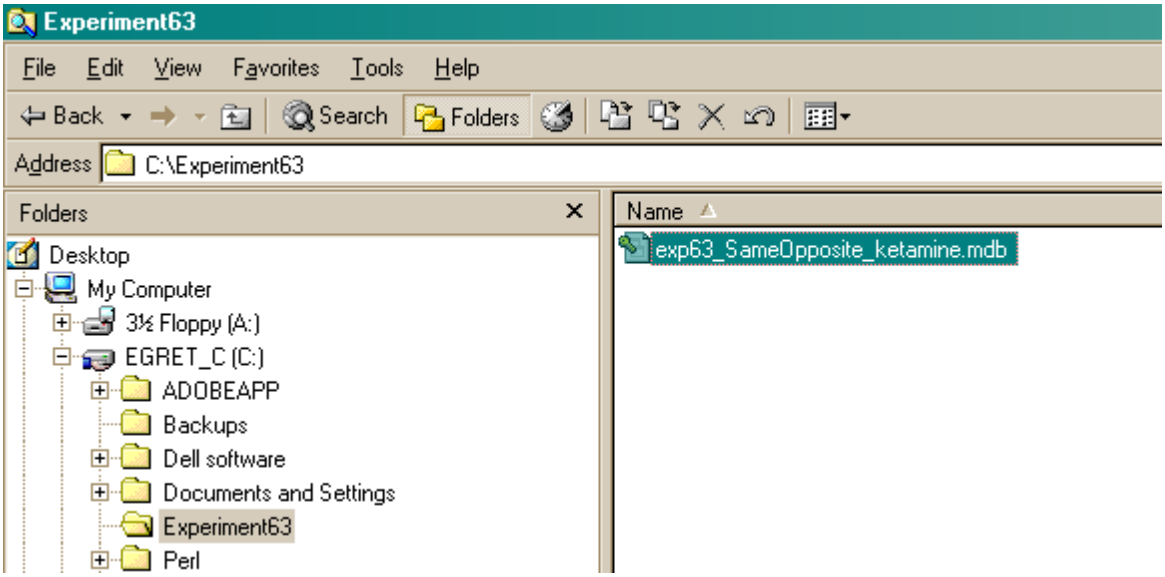
You can also do the whole thing by hand.

STEP 1.

First, make a copy of the supplied database to store *your* data in. Copy the supplied database:



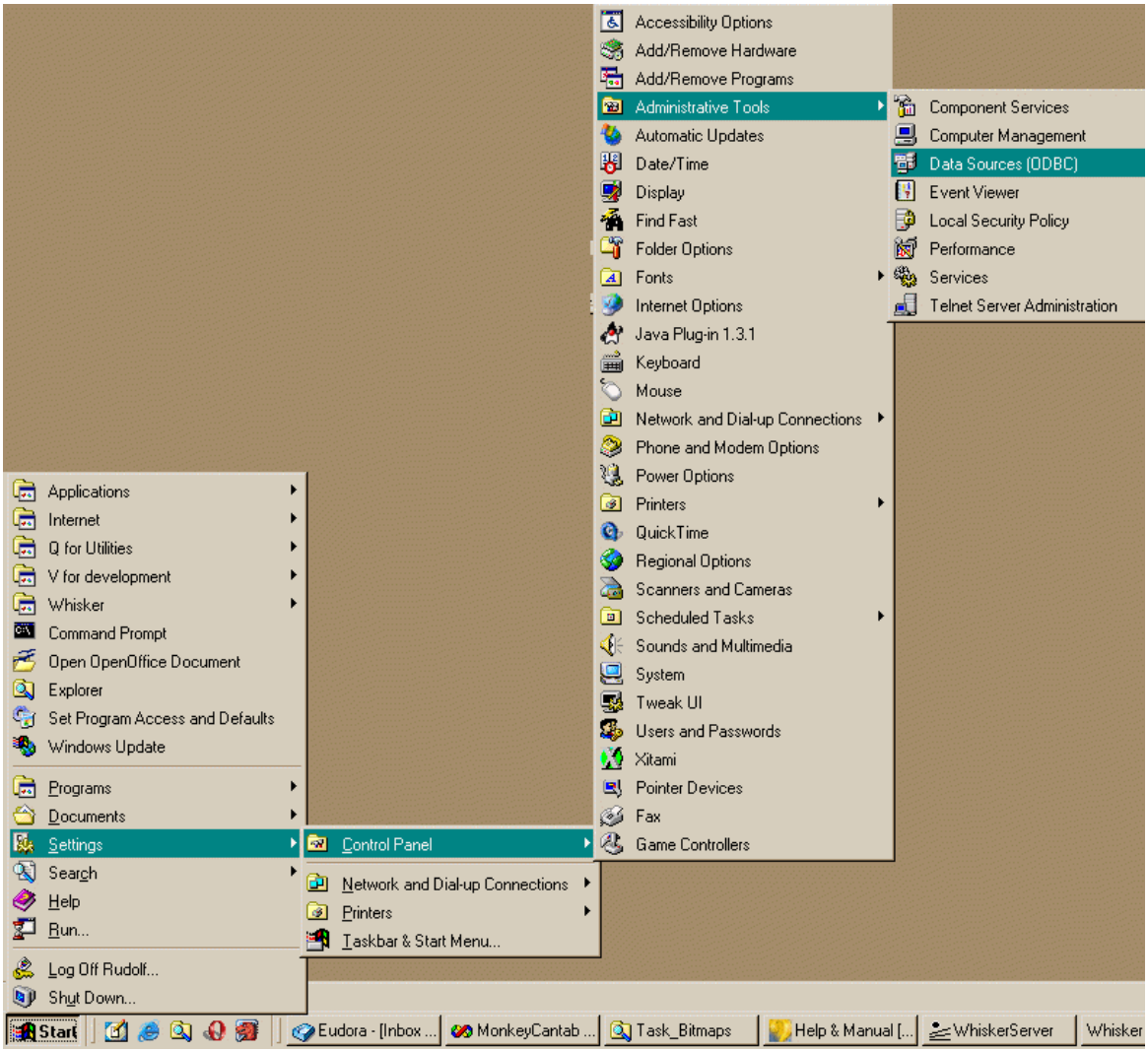
to your own:



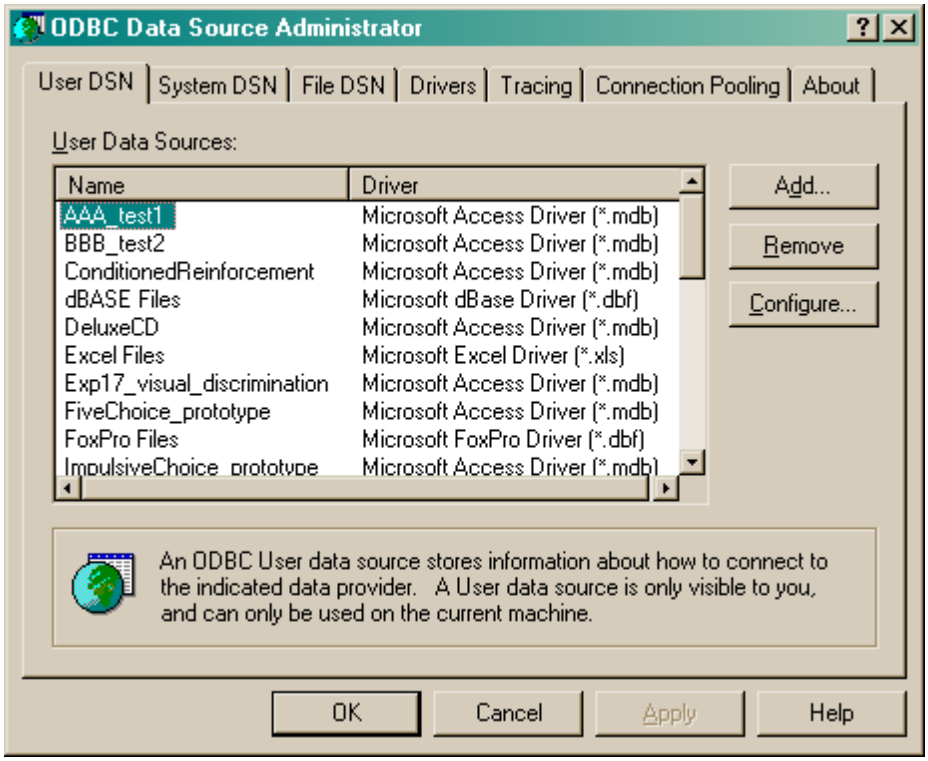
STEP 2.

Now, having already made a working copy of the prototype database supplied with the task, as described above, set *your* copy up as an **ODBC source** as follows.

Choose **Control Panel** → **ODBC** [in Windows 2000, **Control Panel** → **Adminstrative Tools** → **Data Sources (ODBC)**, or the equivalent for your version of Windows:

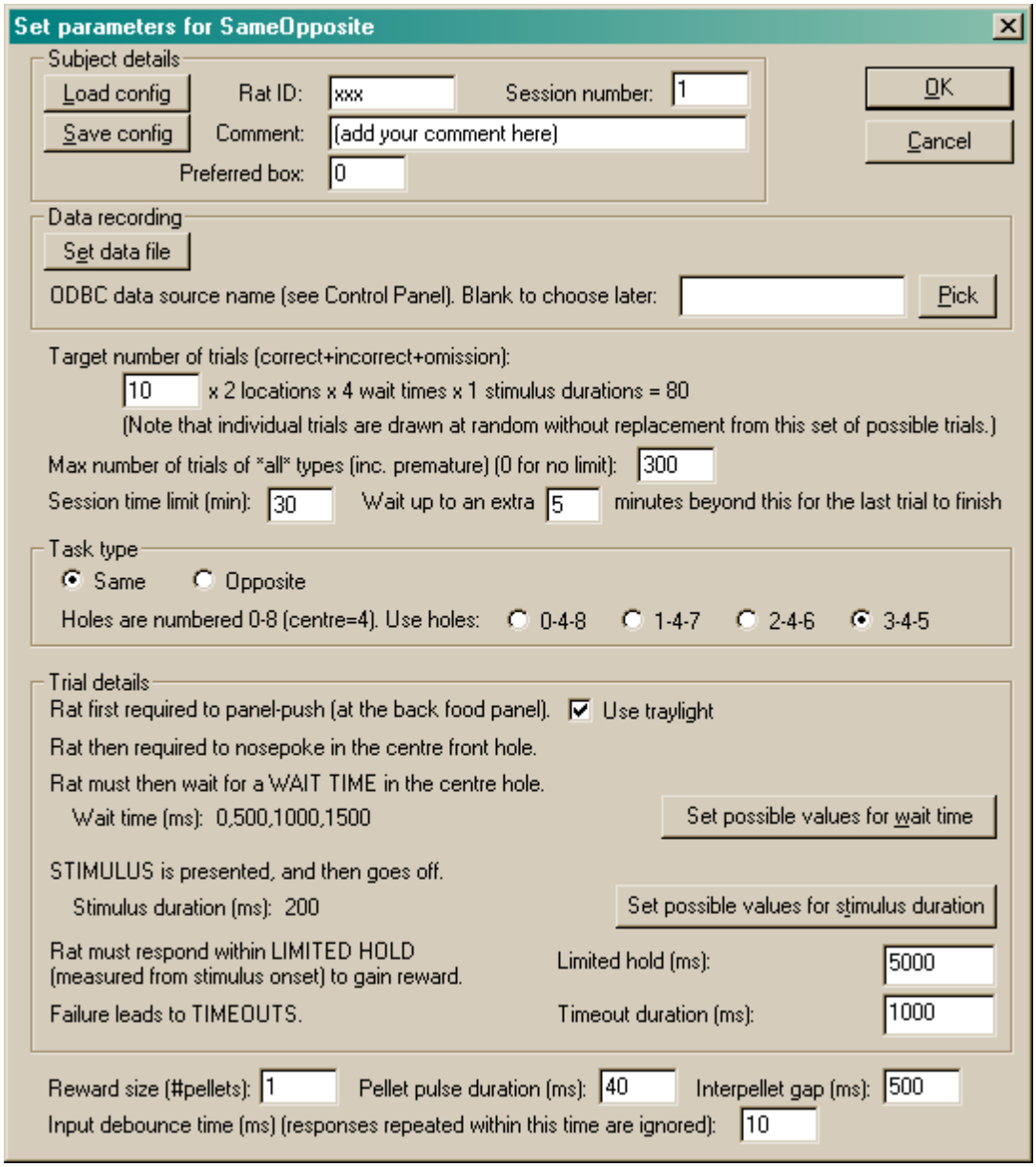


You'll see this:

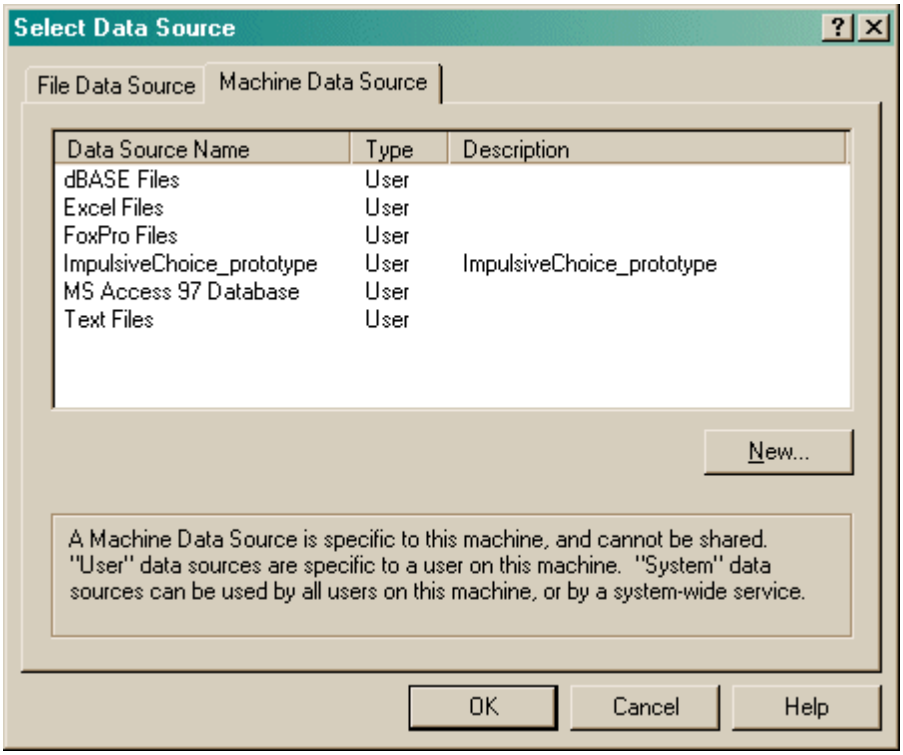


Click **Add**.

Alternatively, you can get to the same point from SameOpposite itself. Click **Pick** from the Parameters dialogue:

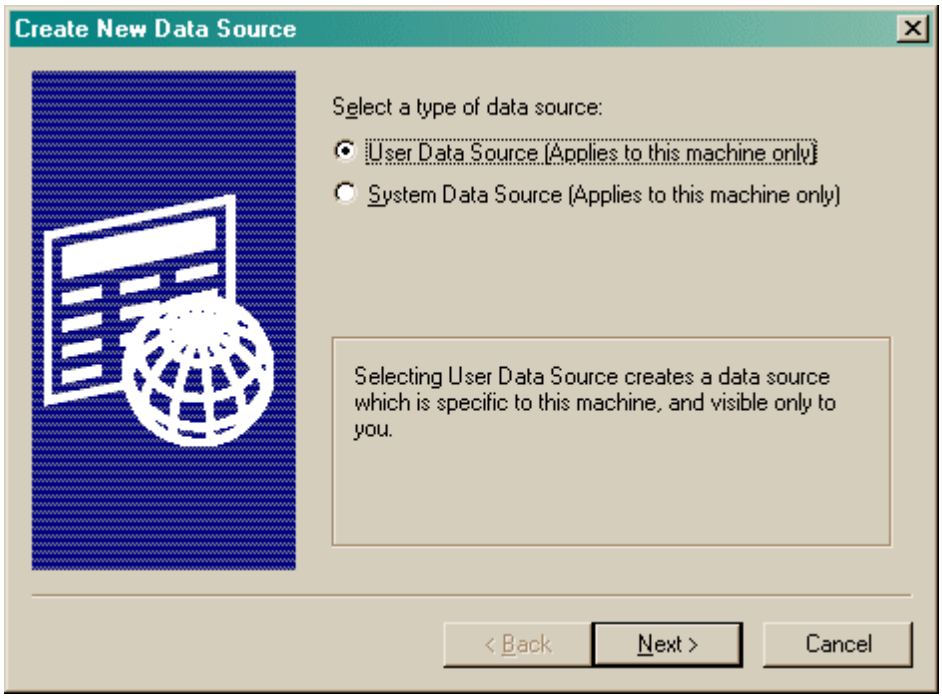


You'll see this:

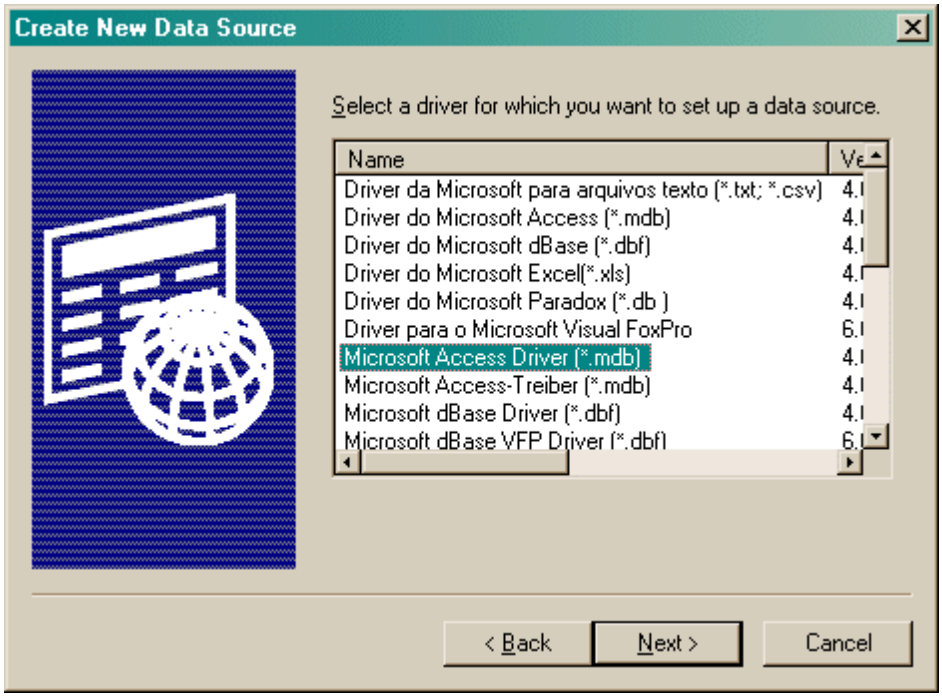


Click **New**.

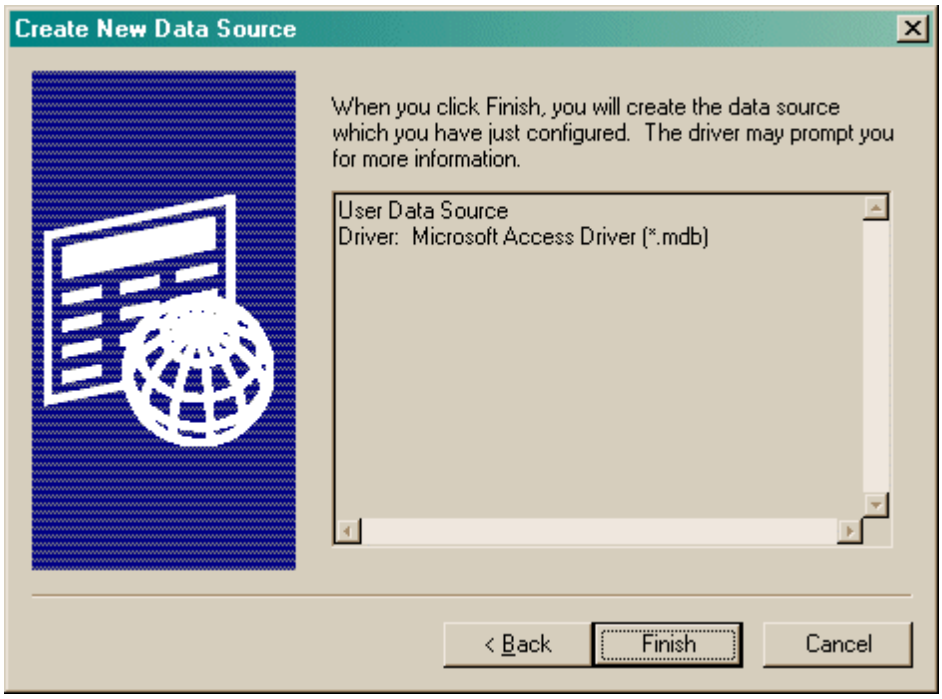
However you got here, you'll see something like this:



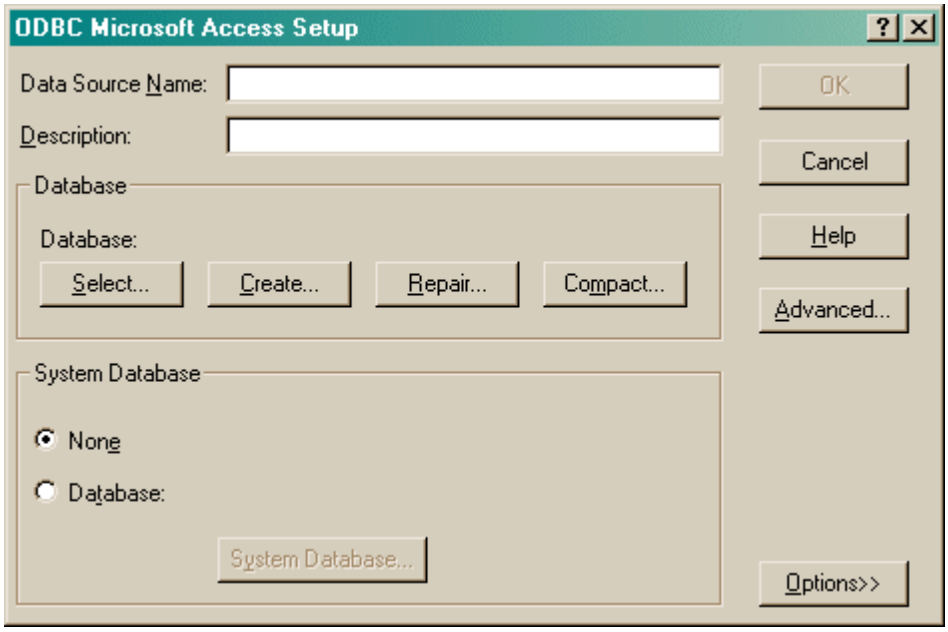
Choose a User or System data source. "User" databases are seen by people logged in as the current user. "System" databases are seen by anybody logged on to this computer. **User** is probably more sensible. Click Next.



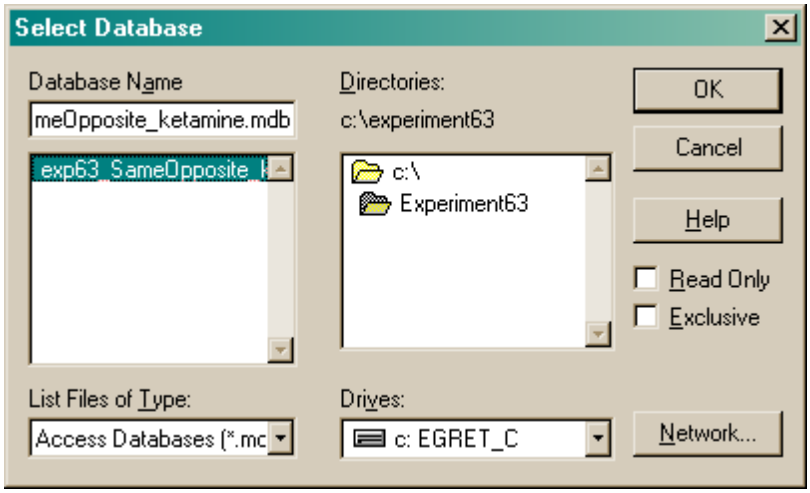
Choose your database driver. You probably want one that's in your language, and the supplied database is in Microsoft Access format (although MonkeyCantab itself will store data in any suitable ODBC-compatible database that has the right table and field names). Click Next.



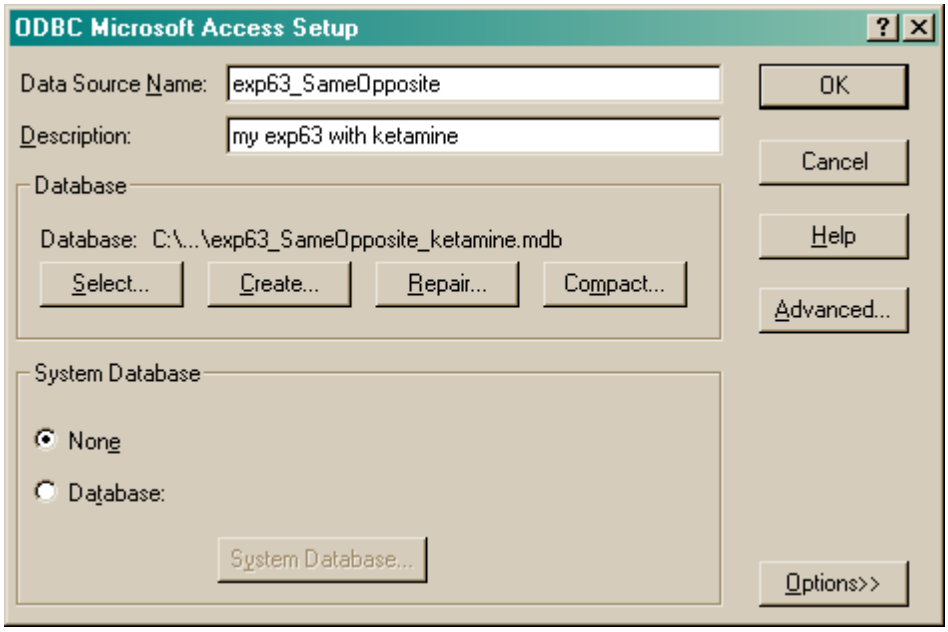
Click Finish. You'll see this:



You should fill in the **Data Source Name (no spaces)** and the **description,** and **Select** a database. When you click Select, this dialogue box appears:



Choose your database here and click OK. Your ODBC data source fields should now all be set up:



Click OK. You will be returned to the ODBC selection screen with your new data source now available.

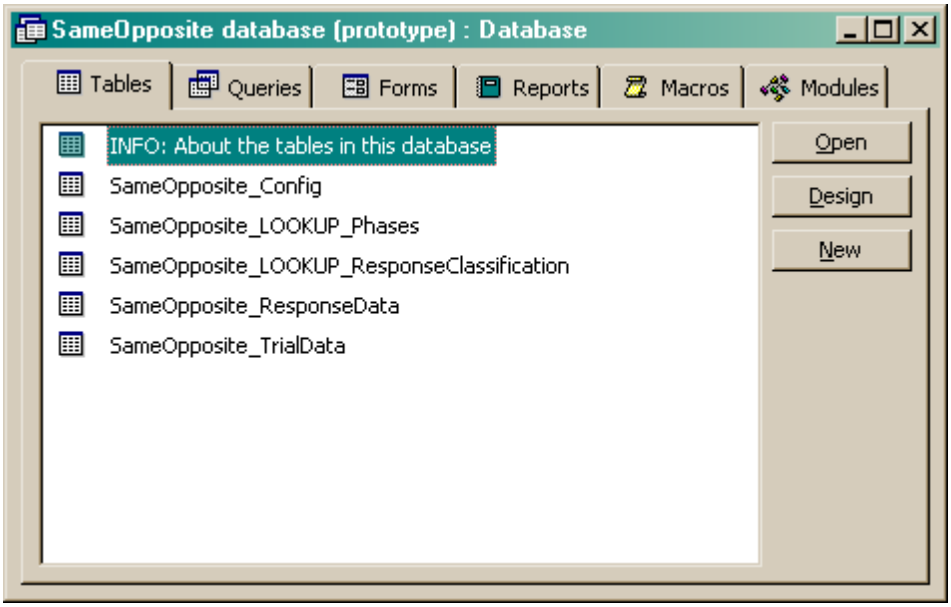
1.9.3 Using the Microsoft Access database for SameOpposite

Remember: you shouldn't use the supplied database without making a copy for yourself. (It will work, but if you ever uninstalled or reinstalled MonkeyCantab, this file might be replaced or lost. It is much safer to make your own copy and set up ODBC to use your copy. See Creating a new ODBC source.)

When supplied, the database is called **"SameOpposite database (sample).mdb"**. Make a copy before using it!

You need Microsoft Access (97 or higher) to use this database. Sorry about that.

When you open the database, it looks like this:



The programe will store its results here. The table called "**INFO: About the tables...**" contains a description of each table (double-click it, or click it and click Open, to see the descriptions).. Click a table and click **Design** to view a list of all the fields. Here, for example, is the design view for the SameOpposite_TrialData table (which stores summary results for each trial):

SameOpposite_TrialData : Table

	Field Name	Data Type	
▶	RecordNum	AutoNumber	(Automatically generated.)
🔑	DateTimeCode	Date/Time	The date/time the task was started.
🔑	Rat	Text	Subject ID
🔑	Box	Number	Box number
🔑	Trial	Number	Trial number
	SessionBeganAt_ms	Number	Time (by the system clock, in ms) the session
	IntendedWaitTime_ms	Number	Time that the program planned to use as the
	IntendedStimulusDuration_m:	Number	Time that the program planned to use as the
	IntendedTargetOnLeft	Yes/No	Did the program intend to put the target on t
	TrialBeganAt_ms	Number	Time (by the system clock, in ms) the trial sta
	PleaseCentreBeganAt_ms	Number	Time (by the system clock, in ms) the subject
	CentredAt_ms	Number	Time (by the system clock, in ms) the subject
	LatencyToCentre_ms	Number	Latency in ms to centre
	TargetOnAt_ms	Number	Time (by the system clock, in ms) the target s
	TargetOffAt_ms	Number	Time (by the system clock, in ms) the target s
	HeadOutOfCentreAt_ms	Number	Time (by the system clock, in ms) the subject
	HeadIntoSideHoleAt_ms	Number	Time (by the system clock, in ms) the subject
	RewardedAt_ms	Number	Time (by the system clock, in ms) the subject
	TimeoutBeganAt_ms	Number	Time (by the system clock, in ms) the timeout
	PleasePushBeganAt_ms	Number	Time (by the system clock, in ms) the subject
	RearPushAt_ms	Number	Time (by the system clock, in ms) the subject
	PerseverativeCentreNosepok	Number	Number of perseverative nosepokes to the c
	PerseverativeSideNosepokes	Number	Number of perseverative nosepokes to the si
	PrematureCentreWithdrawals	Number	Number of premature withdrawals from the c
	PrematureSideNosepokes	Number	Number of premature nosepokes to the side l
	PerseverativeRearPanelPush	Number	Number of perseverative rear panel pushes
	Initiated	Yes/No	Did the subject initiate the trial?
	Centred	Yes/No	Did the subject correctly centre?
	Waited	Yes/No	Did the subject wait long enough whilst centr
	WithdrawnFromCentre	Yes/No	Did the subject withdraw from the centre? (A
	Responded	Yes/No	Did the subject respond to the target stimulu
	RespondedCorrectly	Yes/No	Did the subject respond correctly?
	RespondedIncorrectly	Yes/No	Did the subject respond incorrectly?
	Omission	Yes/No	Was the trial scored as an omission?
	Rewarded	Yes/No	Was the subject rewarded?
	PunishedWithTimeout	Yes/No	Was the subject punished?
	OfferedHole	Number	Offered hole number (0-4)
	ChosenHole	Number	Chosen hole number (0-4)
	ResponseLatency_ms	Number	Response latency, in ms
	ExperiencedTimeout_ms	Number	Duration of timeout experienced, in ms
	CollectionLatency_ms	Number	Latency to collect reward, in ms

Don't modify anything in Design view unless you know what you're doing!

If you close the Design view and click **Open** instead, you see the *contents* of this table. Here is the contents of the SameOpposite_TrialData table. I entered some sample results into this table by running the program and pretending to be a rat for a few trials.

SameOpposite_TrialData : Table

	_RecordNum	DateTimeCode	Rat	Box	Trial	SessionBegan	IntendedWait	IntendedStimu	IntendedTar
▶	1	07/07/2007 18:15:04	xxx	0	1	959187882	1500	1239824	☐
	2	07/07/2007 18:15:04	xxx	0	2	959187882	500	1239964	☐
	3	07/07/2007 18:15:04	xxx	0	3	959187882	1000	1239964	☐
	4	07/07/2007 18:15:04	xxx	0	4	959187882	500	1239964	☐
	5	07/07/2007 18:15:04	xxx	0	5	959187882	0	1244276	☐
*	(AutoNumber)								☐

Feel free to explore the tables.

When you want to extract data for analysis, you may want to create **queries** to do so. (Queries are listed in the "Queries" section of the main database screen.) Queries can be created using Access's

visual query design system, or using the language SQL (Structured Query Language). A little on relational database principles and SQL follows.

1.9.4 Relational databases in general

I have found the most useful way to store data is in a **relational database,** often called a relational database management system (RDBMS). A relational database stores data in **tables,** which are made up of *fields* and *records:*

A table:

	five fields:				
	Date	Rat	NumResponses	NumStimuli	NumReinforcements
one record:	17/2/00 12:29:00	M4	56	5	1
another:	17/2/00 14:37:06	M5	437	43	8
… and so on	17/2/00 12:54:00	M4	263	26	5

The driving principle behind a relational database is this: **never duplicate data.** Let's say our rats came from two groups, Sham and Lesion. If we wanted to record this in the database, so we could analyse data by group, we could store it like this:

Table *BigData*

Date	Rat	Group	NumResponses	NumStimuli	NumReinforcements
17/2/00 12:29:00	M4	sham	56	5	1
17/2/00 14:37:06	M5	lesion	437	43	8
17/2/00 12:54:00	M4	sham	263	26	5

However, this introduces two problems. Firstly, it generates very large tables. Secondly, and more importantly, it is unclear what to do if the data is inconsistent – let's say the underlined 'sham' was changed to 'lesion' by mistake. The database would then not know whether rat M4 was in the Sham or Lesion group – there would be entries for both. The solution to both problems is to create two tables, *linked* on the smallest possible unit of information (in this example, the rat name):

Table *Responses*

Date	**Rat**	NumResponses	NumStimuli	NumReinforcements
17/2/00 12:29:00	**M4**	56	5	1
17/2/00 14:37:06	**M5**	437	43	8
17/2/00 12:54:00	**M4**	263	26	5

Table *Groups*

Rat	Group
M4	sham
M5	lesion

By using the rat name as a **key** (also known as a *foreign* key), the database can link the two tables together whenever we want to know how many responses the two groups made on average.

When we want to find out that sort of information, we **query** the database, specifying how we want to see the data. We could, for example, obtain the following (ignoring a glaring scientific error!):

Query *AverageByGroups*

Group	NumberOfSubjects	MeanNumResponses	MeanNumStimuli	MeanNumReinforcements
sham	2	159.5	15.5	3
lesion	1	437	43	8

Summary of database principles

So relational databases split up the data (which should be entered in well-designed tables without any duplication of information) from queries that look at the data in an infinite variety of ways.

A concrete example: Microsoft Access 97

Microsoft Access 97 is a commonly-used relational database for PCs. It isn't perfect, by a long shot, but I've found it good enough. It supports **structured query language (SQL)** for designing queries; this is a powerful quasi-English language. For example, the query shown above would be written in SQL like this:

```
SELECT group,
    count(*) as NumberOfRats,
    avg(NumResponses) as MeanNumResponses,
    avg(NumStimuli) as MeanNumStimuli,
    avg(NumReinforcements) as MeanNumReinforcements
FROM responses, groups
WHERE responses.rat = groups.rat
GROUP BY group
;
```

If you find all this a bit cryptic, Access also provides a graphical interface for designing queries.

Getting data out of a database

Given a well-designed database, you should be able to get the data out in any conceivable way. The size of this manual doesn't permit a detailed look at relational database design or queries, but there are abundant sources. If you use Microsoft Access, there's the help system, but I also recommend Viescas JL (1997), *Running Microsoft Access 97*, Microsoft Press. Beyond that there is a whole field of database design.

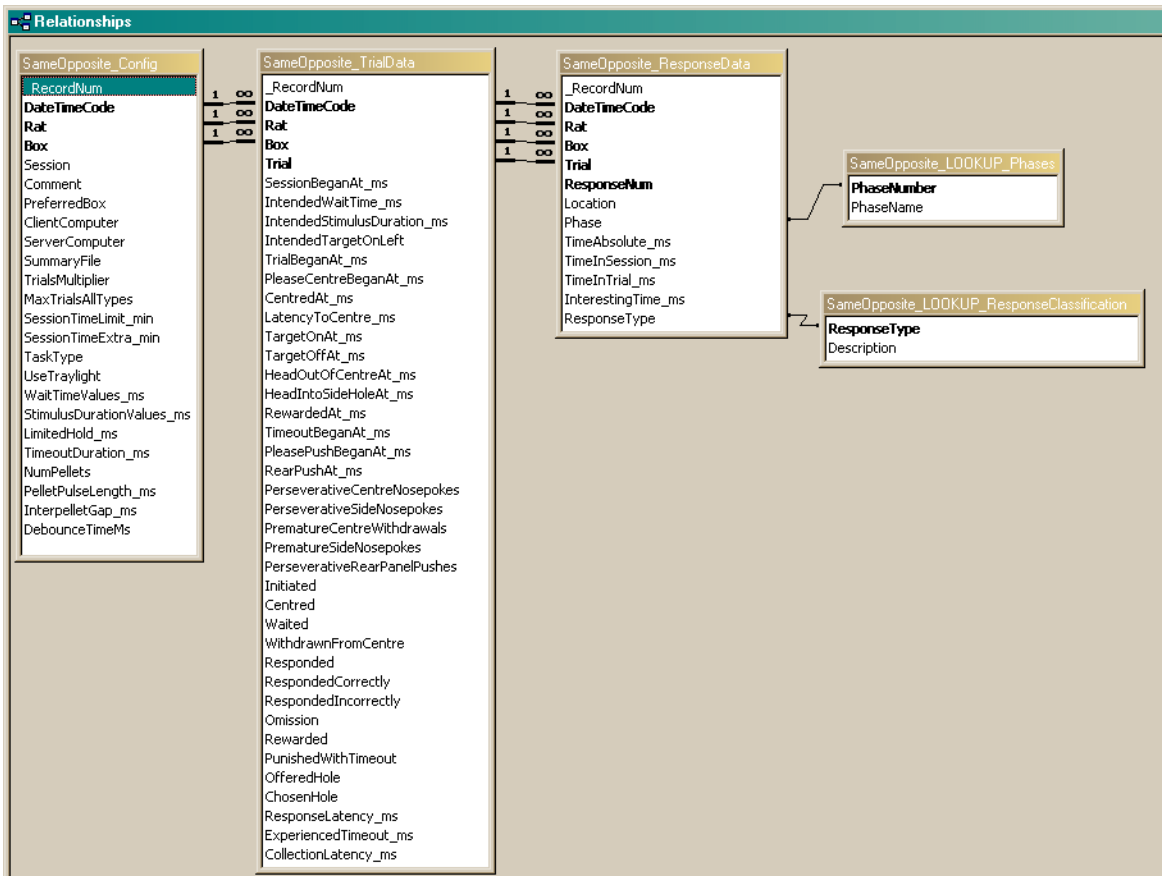
Tip

I operate on the principle that any view of the data is achievable. If the graphical query design can't do it, you can use SQL. If SQL can't do it alone, you can use Visual Basic to augment it. If all that fails (and it hasn't failed me yet) you can always re-export the data and use a general-purpose programming language to analyse it. If the data's there, you can get at it.

One thing is worth noting: modern statistical packages (e.g. SPSS, *http://www.spss.com/*) are starting to support the ODBC standard for exchanging information with databases. You can set up database queries to create views of the data that your stats packages can use, then set up sequences of ODBC capture, analysis and graphical presentation in your stats package. Then whenever you import new data, you can run the entire analysis in a matter of seconds. If you handle large volumes of data, it easily repays the initial effort.

1.9.5 Database structure

This is the structure of the SameOpposite database:



Index

- S -