Cogent 2000 v125 User Manual – 14/04/03

Terms and Conditions

This software package is provided to you free of charge under the following conditions:

1) Acknowledgement:- Acknowledgement from users helps us justify the time we are spending further developing and maintaining this free software. Therefore we request that, when you use Cogent 2000 for your experiments, you include the following statement in your publication:

"This experiment was realised using Cogent 2000 developed by the Cogent 2000 team at the FIL and the ICN and Cogent Graphics developed by John Romaya at the LON at the Wellcome Department of Imaging Neuroscience."

2) Copying:- This package may be distributed freely as long as it is distributed in its original form. It may not be sold without permission.

3) Liability:- The package is distributed as it is, without any warranty implied. No liability is accepted for any damage or loss resulting from the use of these routines.

Cogent 2000 v125 User Manual – 14/04/03

Chloe Hutton, Eric Featherstone, Oliver Josephs plus input from FIL, ICN and LON.

1 Introduction

Cogent 2000 is a MATLAB Toolbox for presenting stimuli and recording responses with precise timing. It comprises of high-level MATLAB commands that may call robust low-level functions. A stand-alone library of graphics functions is also available allowing the user to present sophisticated graphical stimuli from MATLAB. Cogent 2000 provides a flexible and user-friendly environment for the design of a wide variety of experiments. The manipulation of graphics, sound, keyboard, mouse, joystick, serial port, parallel port, subject responses and physiological monitoring hardware are facilitated, all with accurate timing and synchronisation. For fMRI, Cogent 2000 can be configured to receive synchronisation pulses from a scanner allowing experimental timing to be tightly coupled with image acquisition.

1.1 Requirements

Cogent 2000 runs in the MATLAB (http://www.mathworks.com) numerical programming environment and should be used with MATLAB 6.0 or 6.1 running under Windows 2000. Cogent 2000 also requires that a Microsoft Application Programming Interface (API) known as DirectX has been installed. This should be set up by default as part of Windows 2000 but can also be downloaded or upgraded from http://www.microsoft.com/directx. For more information about installing DirectX, see the latest version of the Cogent Graphics Manual .

1.2 Known bugs and limitations

These bugs and limitations are in addition to those in the Graphics library that are listed in the Cogent Graphics manual **G2UsrManvX.pdf.**

1. **Achieving accurate timing:**
   For most accurate timing, Cogent 2000 must be run in MATLAB 6.0 or 6.1 under Windows 2000. It is also necessary to unplug the network, to disconnect any networked drives and to disable network connections to the PC. Network connections can be disabled by going to the **Settings** option of the **Start** menu, and under **Network and Dial-up Connections**, click on the **Local Area Connection** icon and select **disable**. This means that MATLAB must be run with a local copy of the MATLAB license manager.
2. **Loading data files:** The Cogent 2000 commands for loading data files do not make the most efficient use of the MATLAB memory. (i.e. **config_display**, **countdatarows** and **getdata**). Therefore, for the current version of Cogent 2000, we recommend that instead of using these commands, standard MATLAB commands can be used for loading data files. This is especially important if the data file is large and the computer has less than 256MB RAM. This is described in detail in section **3.3.2**.
3. The first execution of the **waitsound** command takes longer to run than subsequent executions. To work around this problem, a sound should be played and the **waitsound** command used at the beginning of a script before starting timed trials.
4. The sound routines such as **playsound** and **waitsound** do not work properly with certain sound cards.

5. The Cogent graphics library command for keyboard logging **cgKeyMap()** has not been tested with high level Cogent commands. In particular this function may interact with the high level Cogent commands for keyboard logging (**readkeys, logkeys, getkeydown, waitkey** etc).
6. The Cogent graphic library command for mouse logging **cgMouse()** has not been tested with high level Cogent commands. In particular this function may interact with the high level Cogent commands for mouse logging (**readmouse, getmouse, waitmouse** etc).
7. The parallel port commands **inportb** and **outportb** will cause errors if used without any arguments.

1.3 Update history for Cogent 2000

Changes from Version 1.24 to Version 1.25:

start_cogent.m	This function now loads the low level graphics libraries into the Matlab workspace. This means that other I/O functions can be used without using the Cogent 2000 graphics and the config_display command.
inportb.dll **outportb.dll**	These functions now work with more recent Operating Systems and processors (e.g. Windows XP and Pentium 4s).

Version 1.25 includes the latest cogent graphics library v1.24 which is also available from: http://www.vislab.ucl.ac.uk/CogentGraphics.html.

Changes from Version 1.23 to Version 1.24:

CogSerial.dll	The functions for sending and receiving serial bytes which use this updated library are now robust.
paletteflicker.m	Function now accepts bits per pixel =0.
paletteflickerrest.m	Function now accepts bits per pixel =0.
preparedartboard.m	Function now accepts bits per pixel =0.
settextstyle.m	Bug corrected 'face' to 'font'.
setforecolour.m	Bug corrected – varargin requires {}.

Version 1.24 also includes the updated cogent graphics library v1.24 which is also available from: http://www.vislab.ucl.ac.uk/CogentGraphics.html.

Changes from Version 1.22 to Version 1.23:

paletteflicker.m	This function now accepts buffers other than 1 and 2 when run in direct colour mode.

Changes from Version 1.9 to Version 1.22:

paletteflicker.m	This function previously displayed an extra white frame at the start of flickering. This has been corrected.
paletteflickerrest.m	A new function that flickers a dartboard then returns to a fixation point. This only works in palette mode.

config_display.m	Now accepts a bits per pixel value of zero and this is the default value. This value will set the display to be opened in 32 bit mode. However, if this is unavailable, the display is opened in 24 bit mode and if this is also unavailable the display is opened in 16 bit mode.
UserPort.zip	This WinZip file is now included with the Cogent 2000 toolbox to enable communication with the parallel port.

Changes from Version 1.2 to Version 1.19:

config_display.m	Now correctly handles six screen sizes: 640x480, 800x600, 1024x768, 1152x864, 1280x1024, 1600x1200.

Changes from Version 1.1 to Version 1.2:

logserialbytes.m	Logs both time of command execution and of arrival of serial bytes.
logkeys.m	Logs both time of command execution and of key presses.
config_display.m	Now supports palette (8-bit) mode.
start_cogent.m	Now supports palette (8-bit) mode.
drawpict.m	Now supports palette (8-bit) mode.
preparestring.m	Now supports palette (8-bit) mode.
loadpict.m	Now gives an error warning if the user attempts to use 8-bit mode (i.e. this command does not support 8-bit mode).

New commands in Version 1.2:

preparedartboard.m	Draws a dartboard.
paletteflicker.m	Flickers the dartboard.

1.4 Availability

The complete Cogent 2000 distribution (which includes high and low-level commands plus the Cogent 2000 graphics library) is available from:

http://www.vislab.ucl.ac.uk/Cogent

This Cogent 2000 distribution includes the most recent graphics library that has been tested with the high-level Cogent commands. The Cogent 2000 graphics library is also publicly available as a stand-alone package from:

http://www.vislab.ucl.ac.uk/CogentGraphics

This manual deals with the Cogent 2000 package and has reference to Cogent Graphics where necessary. A detailed manual for Cogent Graphics, **G2UsrManvX.pdf**, is also distributed with the Cogent 2000 package.

1.5 Installation of Cogent 2000

To download the latest version of the Cogent 2000 distribution, go to:

http://www.vislab.ucl.ac.uk/Cogent2000.

From here you can download the Cogent 2000 package as a self-extracting WinZip file. The contents of the WinZip file can be extracted by double clicking on that file and selecting a folder in which to unzip the package. The self-extracting WinZip file will result in a folder (named with the current version of Cogent 2000) containing the following folders and files:

\Documents
This folder contains documents describing Cogent 2000 and Cogent Graphics functions. This includes the Cogent 2000 manual, **cogmanvX.pdf** and the Cogent Graphics manual, **G2UsrManvX.pdf** (where **X** is the Cogent version number).

\Samples
Example MATLAB scripts and data files demonstrating Cogent 2000 usage.

\Toolbox
Cogent 2000 commands and functions including high-level MATLAB commands (***.m**), low-level Cogent 2000 graphics functions (**cg*.dll**), MATLAB interface libraries (**gprim.dll**, **cogstd.dll**) and low-level functions for sound, input/output and response timing (**Cog*.dll**).

1.6 DirectX Installation

The PC should have DirectX installed. With Windows 2000 this is usually done by default. In the Cogent Graphics manual there is further information about installing and setting up DirectX.

1.7 MATLAB Installation

Cogent 2000 has been fully tested with MATLAB 6.0 and 6.1. Please refer to MATLAB documentation for installation instructions.

1.8 Using this manual

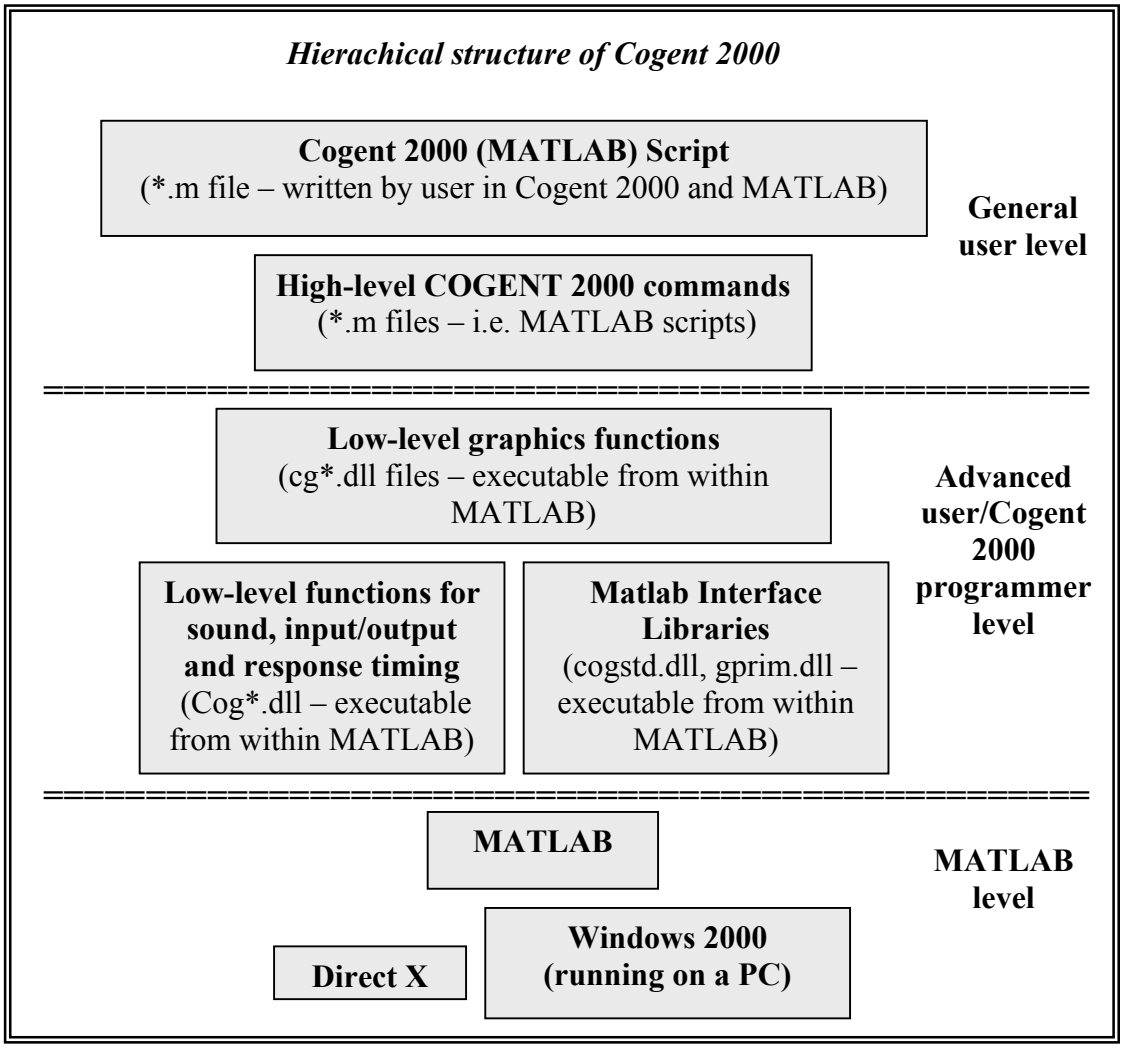
- Chapter 2 describes how to get started with MATLAB and Cogent 2000. It is mostly aimed at those with little or no experience with MATLAB but also has some information that is specific to Cogent 2000 (for example how to exit from a Cogent script).
- Chapter 3 describes in detail the high-level Cogent 2000 commands. This can be used as a high-level programming reference.

- Chapter 4 covers the example scripts that are provided with the Cogent 2000 distribution. More experienced high-level programmers may want to start with this chapter.

1.9 Structure of Cogent 2000 – general overview

In Cogent 2000, a MATLAB script is used to load data, prepare and present stimuli to the subject, record subject responses and accurately log execution and timing. Responses are transferred into the MATLAB workspace from where they can be logged, used to influence the course of the experiment and processed according to the script.

The diagram below illustrates the hierarchical structure of Cogent 2000. For general users, it is only necessary to know about the top level. Advanced users, especially those wanting to generate sophisticate graphical stimuli, and Cogent 2000 programmers also need to know about the middle level.



Cogent 2000 allows MATLAB to be run in two different modes. When a Cogent 2000 script starts, MATLAB runs in normal mode allowing the experiment to be configured. The script then switches MATLAB into Cogent mode to run the experiment and back to normal mode to process results or to run a new script.

2 Getting started in MATLAB and Cogent 2000

Concise online documentation is available for all levels and aspects of MATLAB at (http://www.mathworks.com/support/general/doc.shtml). A simple guide for using MATLAB and Cogent 2000 is presented in the following sections.

2.1 How do I execute commands in MATLAB?

After starting MATLAB, the user will be presented with the MATLAB command line prompt '>>'. MATLAB commands, functions and scripts can be executed by entering the name at this prompt. e.g.

>> **help** Lists all help topics. You can ask for help on any function, including MATLAB functions and any function in the Cogent 2000 Toolbox. e.g.

>> **help for** Explains how to use the MATLAB 'for' loop. Or,

2.2 How do I execute Cogent 2000 commands?

To run Cogent 2000 commands, MATLAB must be instructed to search the directories containing the Cogent 2000 commands. This can be done by selecting the "Set Path" option from the "File" menu of the MATLAB user interface, or by using the command:

>> **addpath** *C:\Cogent2000vX.X***Toolbox**

where *C:\Cogent2000vX.X* is the name of the path where Cogent has been installed. It should now be possible to execute Cogent 2000 commands in MATLAB, e.g.:

>> **help_cogent** Lists all Cogent 2000 commands.

Any other sub-directories containing commands to be executed by MATLAB should also be added to the MATLAB path in the same way. e.g.

>> **addpath** *C:\Cogent2000vX.X***Samples**

2.3 How do I write a script in MATLAB?

A MATLAB script (often referred to as an M-FILE) is a text file that contains MATLAB commands and functions in sequential order. It can be written using any text editor, e.g. the MATLAB text editor:

>> **edit** Opens MATLAB text editor/debugger.

The file must be saved with a filename appended by **'.m'** for MATLAB to recognise it as an executable script. (e.g. *testcript.m*). Other text editors may lack the functionality of the MATLAB **edit** program, which has colour coding as a visual aid and debugging facilities. For help with editing and debugging M-files, see http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_env/matlab_env.shtml.

2.4 How do I write a script in Cogent 2000?

A Cogent 2000 script is a MATLAB script that uses both MATLAB commands and Cogent 2000 commands in sequential order. Editing a working script to make it do what you want is usually easier than writing a script from scratch. The sample scripts in the **\Samples** sub-directory are simple examples of Cogent 2000 scripts that will provide a good starting point for many stimuli.

Lines in a script preceded by a '**%**' sign are comments to describe what the script is supposed to be doing. These lines are not executed by MATLAB. When a command is followed by a '**;**', the display of any output (from that command) to the MATLAB command window, will be suppressed.

The example script **sample1.m** is listed with comprehensive annotations in chapter **4 Tutorial and sample scripts**.

To edit this script:

>> **edit sample1.m**

NB After editing this script, save it with a new name to avoid over-writing the original.

2.5 How is a Cogent 2000 script structured?

There are four components to a Cogent script: configuration, start, run commands, stop:

Configure Cogent: Involves several commands to configure required Cogent 2000 features, e.g. **config_display** to configure display, **config_sound** to configure sound, **config_keyboard**, **config_mouse** etc.

Start Cogent: A single command, **start_cogent**, that switches MATLAB from running in normal mode to run in Cogent 2000 mode. This takes a couple of seconds to run.

Run experiment: Involves a series of Cogent commands in sequential order for stimulus preparation, presentation, logging, subject response recording and processing. Use MATLAB commands for experimental control flow, e.g. **for** and **while** loops, **if** statements etc.

Stop Cogent: A single command, **stop_cogent**, passes control from Cogent back to MATLAB.

2.6 How do I execute a script in MATLAB?

To execute a Cogent 2000 (or MATLAB) script, enter its name at the MATLAB prompt, e.g.

>> **sample1** Executes the sample Cogent 2000 script **sample1.m**.

When configuring the display in Cogent 2000, it is important that the computer monitor is correctly set up. This is addressed in section **3.2.1**.

2.7 How do I force a Cogent script to stop?

To pass control from the Cogent window back to MATLAB before a script has finished (i.e. to escape from Cogent at any time, including if the script contains an error) enter**:**

Control C, **Alt Tab**

It may be necessary to press Alt Tab more than once to get back to the MATLAB prompt. In MATLAB 6.1, it may be necessary to press **Control C** more than once. Now Cogent must be stopped to allow MATLAB to run in normal mode. To do this, at the MATLAB prompt enter:

stop_cogent

2.8 Clearing the MATLAB workspace

The MATLAB command, **clear all**, clears all variables and functions from memory (i.e. the MATLAB workspace). Using this command after **stop_cogent** will clear any variables, globals and functions that were used in the script. Using this command before running either another script or the same script again ensures that there is no interference between old and new copies of variables, globals and functions in the workspace. This command can also be included at the beginning of a Cogent 2000 script.

2.9 Using the Cogent Graphics library

The Cogent Graphics library has been developed as a stand-alone suite of graphics functions that will run independently from the high-level Cogent 2000 commands described in this manual. However, the high-level Cogent 2000 commands are dependent on the graphics library and the graphics commands can be used in conjunction with the high-level Cogent 2000 commands (except where advised not to). When used together, the Cogent script must follow the format described in the following section (i.e. the script must contain **start_cogent**, **stop_cogent** and the configuration commands **config_***).

Concise details and tutorials for using the Cogent 2000 graphics library are given in the Cogent Graphics manual **G2UsrManvX.pdf**.

The following pages of this manual describe most of the high-level Cogent 2000 commands. References to the low-level functions for sound, input/output and response timing can be found in **\Documents\Cog*Ref.doc**.

3 High-Level Cogent 2000 Commands

For a full description of any high level Cogent command, type '**help** *command_name*' at the MATLAB prompt. e.g.

>> **help start_cogent**

Cogent commands are listed in **chapter 5 Appendix I – List of Cogent 2000 functions.**

3.1 Two essential Cogent commands

Two commands that are required in all Cogent scripts are **start_cogent** and **stop_cogent**.

start_cogent	Initialises MATLAB for running Cogent 2000 commands. This command should appear AFTER any configuration commands (see below) and BEFORE ALL other Cogent commands. This command will also initialise MATLAB for running all Cogent 2000 low-level graphics and low-level input/output functions.
stop_cogent	Returns MATLAB from Cogent to normal mode. This should come at the end of the Cogent script.

3.2 Cogent 2000 configuration commands

Before running commands in Cogent it is necessary to configure various aspects of Cogent 2000 if the corresponding features will be used. The configuration commands should appear for the first time at the beginning of the script before all other Cogent commands. Configuration commands are named **config_*feature***, corresponding to the feature being configured. When used with no arguments, the default configurations for each feature are used. The argument types and default values are given in full by the help command, e.g. **help config_display**. The configuration commands are described below.

3.2.1 Configuring the display

config_display(mode, resolution, background, foreground, fontname, fontsize, nbuffers, nbits, scale);

mode	- window mode (0=window, 1=full screen)
resolution	- screen resolution (1=640x480, 2=800x600, 3=1024x768, 4=1152x864, 5=1280x1024, 6=1600x1200)
background	- background colour ([red,green,blue] or palette index)
foreground	- foreground colour ([red,green,blue] or palette index)
fontname	- name of font,
fontsize	- size of font,
nbuffers	- number of offscreen buffers
nbits	- number of bits per pixel (8=palette mode, 16, 24, 32 or 0 where 0 selects the maximum possible bits per pixel out of 16, 24 or 32)
scale	- horizontal size of screen in (visual degrees)

config_display - is equivalent to:
config_display(1, 1, [0 0 0], [1 1 1], 'Helvetica', 50, 4, 0);

When configuring the display, two important things to consider are the difference between 'palette' and 'direct colour' mode and between full screen and window mode. It may also be important to consider the screen's refresh rate. These points are discussed below.

'Palette' mode versus 'Direct Colour' mode

- In 'direct' colour mode, colours are represented by 16, 24 or 32 bits where as in 'palette' mode (a.k.a. '8-bit' mode) colours are represented by 8-bits (i.e. 256 colours).
- The use of these modes is defined by the number of bits per pixel argument in the **config_display** command. The default value of 0 will select the maximum possible number of bits per pixel for the computer's display out of 16, 24 and 32 (i.e. in 'direct' colour mode).
- Palette mode has some advantages over direct mode in terms of speed and memory requirements.

In 'direct' colour mode, the foreground and background colours are specified by n x 3 RGB arrays. In 'palette' mode the colours are specified by single indices to a colour table and 256 (0-255) palette entries can be set to any colour required. Initially all palette entries are set to black. Each palette entry can be set to a new colour using the commands **cgcoltab** and **cgnewpal**.

A full description of 'direct' colour mode and 'palette' mode can be found in the Cogent graphics library user manual, **G2UsrManvX.pdf**.

Window mode versus full screen mode

- When using window mode it is necessary to set the display (number of bits per pixel) correctly on your PC so that it corresponds to the number of bits per pixel specified by **config_display**. If the default bits per pixel=0 is used, the display must be set to 16 or more bits per pixel. If the bits per pixel=8 (i.e. 'Palette' mode) then the display must be set to 256 colours.
- To do this, go to the Windows '**Start**' menu, select '**Settings**', '**Control Panel**', double click on '**Display**', select the '**Settings**' tab and under '**Colors**', choose the setting that reflects the desired number of bits per pixel.
- When testing scripts it is better to use window mode (even if the stimulus does not require a display window). This is so that the MATLAB window is visible (i.e. not obscured by the full screen display) and any MATLAB or Cogent error messages can be monitored.
- Cogent timing is **only** accurate and properly synchronised to the screen refresh rate when the display is configured to full screen mode.
- If using window mode with palette mode, only palette entries 0-235 can be assigned colours.

Setting the screen's refresh frequency

- The setting for the screen refresh rate may influence timing for the Cogent script because this is the time taken to display a new frame. The refresh rate may also need to be adjusted to correspond with other display hardware such as a projector.
- To adjust the screen's refresh frequency, go to the Windows '**Start**' menu, select '**Settings**', '**Control Panel**', double click on '**Display**', select the '**Settings**' tab, under '**Advanced**', select the '**Monitor**' tab, then choose the required '**Refresh Frequency'**.
- To check that Cogent 2000 has detected this change, run **config_display(0)** followed by **start_cogent**. One line of the text printed to the MATLAB command window gives the display information e.g. **Display:640x480x8 60.02Hz**. This means the screen's settings are resolution=640x480, bits per pixel=8 and refresh frequency = 60Hz.
- If the selected refresh frequency has not been detected by Cogent 2000, it will be necessary to download a DirectX file called **DirectX.cpl** from http://www.microsoft.com/directx. This file should be copied into your *%WINDIR%***system32** folder. (It may be necessary to have administrator privilege to do this.
- Once installed, a DirectX icon will appear in the **Start Menu-Settings-Control Panel** folder. Double click on this icon, select the '**DirectDraw**' tab, and select the required '**Forced Refresh Rate**'.

Examples of configuring the display for 'direct' colour mode, 'palette' mode, full screen and window mode, are given in scripts **sample1.m**, **sample2.m**, **sample3.m** and **sample4.m** and described in chapter **4.**

3.2.2 Configuring input and output data

INPUT DATA

config_data(*'datafile.dat'***)** - Configure Cogent to read the data file specified by *'datafile.dat'*. NB This configuration command **always requires** an input argument.

OUTPUT LOG FILE

config_log - Sets up a log file with default name **'Cogent-YYYY-MM-DD-HH-MM-SS.log'**.

config_log(*'test.log'***)** - Sets up a log file with name *'test.log'*. NB If an existing file name is given, the file will be **appended**.

OUTPUT RESULTS FILE

config_results - Sets up a results file with default name **'Cogent-YYYY-MM-DD-HH-MM-SS.res'.**

config_results(‘*test.res***’)** - Sets up a results file with the name *'test.res'*. NB If an existing file name is given, the file contents will be **overwritten**.

3.2.3 Configuring the hardware

SOUND

config_sound(nchannels, nbits, frequency, nbuffers)

nchannels	- number of channels (1=mono, 2=stereo).
nbits	- number of bits per sample (8 or 16).
frequency	- number of samples per second (e.g. 8000, 11025, 22050 and 44100).
nbuffers	- number of sound buffers.

config_sound - is equivalent to:
config_sound(1, 16, 11025, 100);

KEYBOARD

config_keyboard(qlength, resolution, mode)

qlength	- maximum number of key events recorded between each keyboard read.
resolution	- timing resolution in ms.
mode	- device mode (‘exclusive’ or ‘nonexclusive’)

config_keyboard - is equivalent to :
config_keyboard(100, 5, ’nonexclusive’)

‘Exclusive’ versus ‘nonexclusive mode’

- In exclusive mode it is not possible to interrupt the script using the keyboard. (i.e. CTRL C, ALT TAB will not work), but the timing of key events is accurate.
- For testing scripts it is therefore better to use nonexclusive mode (timing is less accurate but scripts can be interrupted via the keyboard

MOUSE

config_mouse(interval)

interval - Configure mouse to be in polling mode with a polling interval given by interval in milliseconds.

config_mouse - Configures mouse with default settings to be in non-polling mode.

‘Polling’ versus ‘Non-polling’ mode

- In polling mode, all mouse actions are read at intervals (as specified) before the next Cogent mouse command is encountered.

- In non-polling mode, only the last mouse action is read when a Cogent mouse command is encountered.

SERIAL PORT

config_serial(port, baudrate, parity, stopbits, bytesize)

port - COM port number (1,2,3,4,etc).
baudrate - (110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 11520, 128000, 256000)
parity - (0 = none, 1=odd, 2=even, 3=mark, 2=space)
stopbits - (0 =1, 1=1.5, 2=2)
bytesize - (4 bits, 8 bits)

config_serial - Sets up a serial port with default settings:
config_serial(1, 9600, 0, 0, 8)

3.3 How do I get data into Cogent 2000?

Loading a data file can be done using Cogent 2000 commands or with standard MATLAB commands. The Cogent 2000 commands that are described in the following section are simple to use but may not make the most efficient use of the MATLAB memory. Therefore, for the current version of Cogent 2000, we recommend that instead of using the commands described in section **3.3.1**, the MATLAB commands described in section **3.3.2** can be used for loading data files. This is especially important if the data file is large and the computer has less than 256MB RAM.

3.3.1 Using Cogent 2000 commands for loading data

Loading a data file requires the configuration command **config_data(** '*filename.dat'* **)**.

The data file *filename.dat* must be an ASCII text file containing word-strings (e.g. 'ONE', 'CAR'), characters (e.g. '+', 'a', '1'), names of image or sound files (e.g. 'tree.bmp', 'camper.wav') or numerical values (e.g. 1, 20). All word-strings, characters and filenames must be written within single quotes. The data must be organised in rows and columns. Data files can be created using any text editor (e.g. the MATLAB **edit** command or Windows Notepad).
Most of the example Cogent scripts in the **\Samples** subdirectory use data files (see also chapter **4**).

The number of rows in a data file can be determined automatically using **countdatarows**, and the data extracted from the data file using **getdata**.

datum=**getdata(r, c);**	Returns data item at r=row, c=column.
numrows=**countdatarows;**	Returns the number of rows in the data file.

e.g. Consider a data file called '*testdata.dat*' containing the following lines of text:

'ONE' 1
'TWO' 2
'THREE' 3

The Cogent commands to load and read this data file are:

```
config_data( 'testdata.dat')   % Specifies data file to be read.
start_cogent                   % Start Cogent
numrows=countdatarows;         % Counts data rows. i.e. numrows=3.
string1=getdata(1,1);          % Gets data from position 1,1. i.e string1 ='ONE'.
val2=getdata(2,2);             % Gets data from position 2,2. i.e. val2 = 2.
stop_cogent                    % Stop Cogent
```

Commonly, a MATLAB 'FOR LOOP' is used to sequentially count through the data rows, getting each item of data to do something with it (see for example, **sample1.m** in chapter **4**).

3.3.2 Using standard MATLAB commands for loading data files

In order to use the MATLAB commands for loading the data file, the data file definition is slightly different from the data file definition required when using the Cogent 2000 commands. The difference is that in the data file, strings do not need to be written within single quotes. The data file to be loaded, *filename.dat* must be an ASCII text file containing word-strings (e.g. ONE, CAR), characters (e.g. +, a, 1), names of images or sound files (e.g. tree.bmp, camper.wav) or numerical values (e.g. 1, 20). The data must be organised in rows and columns with the same data types within each column. Data files can be created using any text editor (e.g. the MATLAB **edit** command or Windows Notepad).

In MATLAB, the command **textread** can be used for reading data files organised with the above format:

[*a, b, c, ...***] = textread(** *filename, format* **)**

Data from the file called filename is read into the variables a, b, c etc. The format string gives the expected type of each data item in each line. The number of return arguments must match the number of conversion specifiers in the format string.

e.g. Consider a data file called '*testdata.dat*' containing the following lines of text:

ONE	1
TWO	2
THREE	3

The Cogent commands to load and read this data file are:

```
start_cogent                                % Start Cogent
[ a, b ] = textread( 'testdata.dat', '%s%d' ); % Read the data file. The format
                                            % specifiers are %s for a string and %d
                                            % for a number.
numrows=size( a, 1);                        % Get number of data rows from the
                                            % length of the return variable.
% Get first data elements
string1=a{1};           % Gets data from position 1,1. i.e string1 ='ONE'.
Val1=b(1);              % Gets data from position 1,2.  i.e. val1 = 2.

% Or to read data elements in a loop:
for i=1:numrows
        string=a{i};
        Val=b(i);
end

% Note that for string data, curly brackets must be used and for numeric values
% normal round brackets.
stop_cogent             % Stop Cogent
```

3.4 How do I present visual stimuli (words, pictures etc.)?

All visual presentation commands require that the display has been configured using **config_display** (see **3.2.1 Configuring the display**).

- Visual stimuli are first prepared in memory and then presented onto the subject's monitor. Cogent maintains several memory buffers for display, each of which holds one screen of data. Preparation involves loading image files and/or rendering text into one of these memory buffers. The subject sees nothing at this stage.
- The stimuli are then presented by copying the memory buffer to the screen. By default, the stimuli will be displayed at the centre of the screen but an offset position can also be specified.
- Copying the memory buffer to the screen occurs within one screen refresh so that the subject sees the whole screen update at once. The contents of only one memory buffer can be displayed at any one time on the screen.

e.g. Present subject with a fixation point followed by a brief image of a tree:

```
preparestring('.', 1)          % Puts full stop (fixation point) in the middle of buffer 1.
loadpict('tree.bmp, 2,10,5)    % Puts the tree image in buffer 2 (offset from centre by
                               % x=10 and y=5).
drawpict(1)                    % Presents buffer 1 (the fixation point).
drawpict(2)                    % Presents the tree (fixation point disappears).
drawpict(1)                    % Returns to the fixation point.
```

If you wanted the fixation point to remain over the tree this could be achieved by:

```
preparestring('.', 1)          % Puts full stop (fixation point) in the middle of buffer 1.
loadpict('tree.bmp', 2)        % Puts the tree image in the middle of buffer 2.
preparestring('.', 2)          % Puts a full stop (fixation point) in the middle of buffer
                               % 2, i.e. in the middle of the tree. Since the fixation
                               % point is now present in both buffers, it will appear to
                               % remain on screen during the drawpict transitions.
```

If you want to manipulate the image contained in the graphics file, you can use **loadpict** to return a MATLAB matrix containing the image. In this case, the image is loaded into a matrix rather than a memory buffer. The MATLAB matrix can then be manipulated and **preparepict** used to display the result. e.g.

```
A=loadpict('tree.bmp');        % Puts the tree image in a MATLAB matrix specified
                               % by A.
IA=A(size(A,1):-1:1,:);        % Inverts the rows in the image (i.e. turns image upside-
                               % down).
preparepict(IA,1)              % Puts the inverted image in middle of buffer 1.
drawpict(1)                    % Presents buffer 1 (upside-down tree).
```

- The commands **preparestring** (for text) and **loadpict** (for images on disk) or **preparepict** (for images in the MATLAB workspace) are the two easy ways to place

visual stimuli in the memory buffers. In all cases, **drawpict** is used to display the picture on the screen.

- Graphics files that can be read by Cogent are BMP, TIFF, PNG, JPEG and PCX (these are all graphics files recognised by MATLAB).
- Other graphics may be rendered by making direct use of the low-level graphics library underlying Cogent 2000 (see **G2UsrManv124.pdf**).

Cogent 2000 also includes high level commands to display a black and white dartboard on a grey background, **preparedartboard**, and to flicker the dartboard at a specified frequency **paletteflicker**. This is a commonly used stimulus for primary visual stimulation. The **preparedartboard** and **paletteflicker** functions work in both palette (8-bit) mode and in direct colour mode. These commands require that **config_display** has been set up for the corresponding palette mode or direct colour mode as described in section **3.2.1**.
Another function **paletteflickerrest** will flicker the dartboard then return to a fixation point. This function only works in palettemode.
The timing of the **paletteflicker** and **paletteflickerrest** commands is dependent on the computer screen's refresh frequency which is addressed in section **3.2.3**.

preparedartboard(*b, r1, r2, dr, dt***)** Draws one dartboard (requires palette mode) in the buffer specified by b, with an inner radius specified by *r1* and an outer radius specified by *r2* (*r1* should be < *r2*, if not, a null dartboard is generated). The divisions of the dartboard are specified by the radial square size, *dr* and the angular square size *dt*.

preparedartboard(1, 20, 200, 30, 18) Draws a dartboard in buffer 1 using palette mode with the default parameters (i.e. same as **preparedartboard** with no parameters).

preparedartboard([1 2], 20, 200,30,18) Draws two dartboards in buffers 1 and 2 using direct colour mode.

paletteflicker(*b,f1,f2,n,g***)** Flickers a dartboard prepared in the buffer or buffers specified by *b*. The dartboard is displayed in one state (e.g. black/white) for *f1* frames and then the other state (e.g. white/black) for *f2* frames. The cycle of *f1* frames followed by *f2* frames is repeated *n* times. When using palette mode, the background grey level can be specified by *g*).

paletteflicker(1, 4, 4, 225, 32767) Flicker the dartboard drawn in buffer 1 using palette mode. The dartboard is displayed in each state for 4 frames (if the frame refresh rate of the display is 60 Hz, the flicker frequency is (4+4)/60 = 7.5Hz). The 8 frame cycle is repeated 225 times (i.e. for 225/7.5 = 30 seconds).

paletteflicker([1 2], 4, 4, 225)	Flicker the dartboard drawn in buffers 1 and 2 using direct colour mode. Other parameters are as above.
paletteflickerrest(1,4,4,225,32767)	Flicker the dartboard in buffer 1 for 225*8 frames then return to a fixation spot. This command only works in palette mode. See example script **flash30.m** in chapter **4** and in the **Samples**\ sub-directory.

3.5 How do I read keyboard input?

All keyboard commands require that the keyboard has been configured using **config_keyboard** (see section **3.2.3**).

- Keyboard presses can be systematically read and logged using the two commands, **readkeys**, and **logkeys**.
- The command **clearkeys** clears all previously read keyboard events.

e.g Display an image and during this time read and log all key events.

clearkeys	Clears all previous keyboard events.
drawpict(1)	Display the image in buffer 1.
readkeys	Reads all the keyboard presses since the last call to **readkeys** or **clearkeys**.
logkeys	Logs and time stamps all key presses that have been read by the last call to **readkeys.** The log is displayed in the MATLAB command window and saved to a log file if it has been defined by the **config_log** command. **logkeys** logs the time when each key was pressed and as well as the time when the information was logged.

An example of a script that demonstrates reading and logging key presses is **sample6.m** in the **Samples**\ sub-directory and chapter **4.**

- Keyboard presses and releases can be used to control the program flow. This is done using the commands **waitkeyup** and **waitkeydown,** which can wait for either any key or a specific key to be pressed or released.
- The **waitkeyup** and **waitkeydown** commands also log all keyboard activity since the last **readkeys** or **clearkeys** (in fact they internally execute **readkeys** and **logkeys**).

waitkeyup(*time*)	Waits a time *time* (in ms, e.g. 1000) for any key to be released. If no key is released within this time, execution of the script continues.
waitkeydown(1500**, 1)**	Waits 1500ms for the letter 'A' (key ID=1) to be pressed. If the 'A' key is not pressed within the specified time, execution of the script continues. All keyboard activity is logged until after either 'A' is pressed or 1500ms has elapsed (whichever comes first).
waitkeyup(inf, 71)	Waits an infinite time for the space bar (key ID=71) to be released. All keyboard activity is logged the space bar is pressed.

- Information about key presses can be extracted from the current keyboard map using **lastkeydown**, **lastkeyup**, **getkeydown**, **getkeyup**, **countkeydown** and **countkeyup**.

- All of these commands return information about keyboard activity since the last **readkeys** was executed.

[*k, t, n*] = **getkeydown**	Returns information about all keys pressed since the last **readkeys**. Return values are: *k* = key ID, *t* = time in ms and *n* = number of keys pressed. If more than 1 key was pressed, *k* and *t* will be column vectors of length *n*.
[*k, t, n*] = **getkeyup**	As above for key releases.
n = **countkeydown**	Returns the number of keys pressed since the last **readkeys.** (NB this number is also returned by **getkeydown**).
n = **countkeyup**	As above for key releases.
[*k, t*] = **lastkeydown**	Returns information about the last key pressed since the last **readkeys**. Returns the values *k* = key ID and *t* =time in ms.
[*k, t*] = **lastkeyup**	As above for key releases.

An example script using **getkeydown** is given in **sample7.m** in the **\Samples** sub-directory and in chapter **4**.

- If waiting for a specific key press, it is necessary to know the ID that identifies each key in Cogent 2000. The ID is a numerical value e.g. the ID for the 'A' key is 1, for 'B' is 2 etc. Typing **help readkeys** lists the ID of each key. It is also possible to use the command **getkeymap** to return the IDs of all keys.

keymap=**getkeymap**	Returns a structure called *keymap* containing key IDs. The variable *keymap* is a structure containing the key ID for each key. The syntax to get the ID corresponding to each key is for example for the letter P, *keymap*.X.
[*k, t*] = **lastkeyup**	Get the last key pressed.
k == *keymap.S*	Was the last key pressed an 'S'?

An example showing how to determine whether specific keys were pressed is given in example script **sound3.m** in the **\Samples** sub-directory and in chapter **4**.

3.6 How do I present sound stimuli?

All sound commands require that the sound has been configured using **config_sound** (see section **3.2.3**).

- Sound stimuli are prepared in memory and presented to the subject via earphones. In the same way that Cogent display buffers can hold image data for display, the sound buffers hold sounds to be played.
- Sounds can either be loaded from a Microsoft '.wav' sound file or can be created in Cogent 2000.
- Sound files ('.wav' format) can be generated using third party software (e.g. Cool Edit Pro - http://www.hitsquad.com/smm/programs/Cool_Edit_Pro).

e.g. Present subject with sound contained in a '.wav' file:

loadsound('*camper.wav***', 1)**	Puts sound in buffer 1.
playsound(1)	Plays sound in buffer 1.
waitsound(1)	Waits for the sound in buffer 1 to finish playing.

NB *The first execution of the **waitsound** command takes longer to run than subsequent executions. To work around this problem, a sound should be played and the **waitsound** command used at the beginning of a script before starting timed trials.*

Sounds can also be generated in Cogent 2000:

preparesound(*soundmat***,1)**	Fills buffer with sound specified by values in the matrix 'soundmat'. This matrix must be defined in the MATLAB workspace and should have 1 column for mono sound or 2 columns for stereo sound.
preparewhitenoise(1000,1)	Fills buffer 1 with 1000 milliseconds of white noise.
preparepuretone(500,1000,1)	Fills buffer 1 with a 500 Hz sine wave sound lasting 1000 milliseconds.

In all cases **playsound** must used to play the sound in the buffer.

Other Cogent 2000 sound commands are:
setsoundfreq, setsoundvol, setsoundposition, getsoundfreq, getsoundvol, soundposition, loopsound, stopsound (use matlab **help** command for usage). Also see sample scripts: **sound1.m**, **sound2.m** and **sound3.m** in the **\Samples** subdirectory and in chapter **4**.

3.7 How do I read mouse input?

Mouse commands require configuration using **config_mouse** (see section **3.2.3**).

- The mouse can be used to control the flow of the script and to return information about the mouse pointer position on the screen.

waitmouse Does nothing until any mouse button has been pressed and released. To suspend execution at any point, **config_mouse** should be set to non-polling mode or polling mode with a very short time interval, i.e. use **config_mouse** or **config_mouse(10)**.

- To read information about mouse activity, use **getmousemap** to initialise a variable that will store the current mouse map (i.e. mouse position and button presses).
- To update the current mouse map use **readmouse** .
- To extract the required information from the current map use **getmouse**.

map=**getmousemap** Defines a MATLAB variable *'map'* that will contain information about the mouse map. This only needs to be defined once. The contents of the variable *'map'* are automatically updated each time the mouse information is read by Cogent using **readmouse**.

readmouse Updates the mouse map (i.e. current information contained in the variable assigned to **getmousemap**) for the latest mouse activity (movement or button clicking). If the mouse is configured for non-polling mode, the 'map' is only updated with the last action. If in polling mode, the 'map' is updated with the last actions that were read at the interval specified in the **config_mouse** command.

x=**getmouse(***map***.X)** Returns the change in the spatial x-position of the mouse. In non-polling mode, **getmouse(***map***.X)** will only return the last value read. In polling mode **getmouse(***map***.X)** may return multiple values depending on the number of mouse actions and the polling interval.

y=**getmouse(***map***.Y)** As above for change in the spatial y-position of the mouse.

b1=**getmouse(***map***.Button1)** Returns the state of the left mouse button (key down=128, key up=0). In non-polling mode **getmouse(***map***.Button1)** can only return 0 or 128. In polling-mode **getmouse(***map***.Button1)** can return empty, single or multiple values.

b2=**getmouse(***map***.Button2)** As above for right mouse button. (Also for Button3 and Button4 if mouse has these buttons).

For an example of using these mouse commands, see sample script **\Samples\samplemouse.m.**

3.8 How do I present stimuli at specified times?

- Timing is controlled in Cogent 2000 using the commands **wait**, **waituntil** and **time**.
- There are also commands that wait for specific actions (e.g. **waitsound**, **waitkeyup**, **waitkeydown**, **waitmouse** and **waitserialbytes).**
- When the **start_cogent** command is called, time in Cogent is set to 0. The command **time** will return the current time (in ms) since **start_cogent** was called:

t=**time**;	Returns current time *t* in ms since **start_cogent** was called.

The **waituntil** command waits until a specified time *t*, where *t* is the time elapsed since the **start_cogent** command.

waituntil(*t*);	Waits until *t* ms has elapsed since **start_cogent** was called.

- Using **waituntil** together with **time** allows timing to be set relative to some other event in the script. (This is illustrated in example scripts **sample5.m** to **sample9.m** in the **\Samples** sub-directory and in chapter **4**).

t0=**time**;	Sets *t0* = current time (relative to **start_cogent**).
waituntil(*t0*+1000);	Waits until 1000ms has elapsed since **time** was executed and *t0*+1000ms since **start_cogent**.

- The **wait** command waits from the current time for a specified amount of time *t*, where *t* is the total duration of the wait and is not relative to the start of the Cogent script. This is illustrated in the example scripts **sample1.m** to **sample4.m**.

wait(*t*);	Waits for *t* ms to elapse.

It takes a small but finite amount of time to execute most commands in Cogent (i.e. in the order of a few 10s of ms). By using the **waituntil** command it is possible to avoid cumulative timing errors that may be caused by the execution of a series of commands. In contrast, the **wait** command allows the time of a delay to be specified exactly. The following examples illustrate the difference between the **waituntil** and **wait** commands.

e.g. Present an image until exactly 1000ms after the beginning of the experiment:

preparestring('+', 2)	% Puts fixation cross in the middle of buffer 2.
preparestring('TREE', 1)	% Puts the word 'TREE' in the middle of buffer 1.
drawpict(1);	% Presents buffer 1.
waituntil(1000);	% Waits for 1000ms from the **start_cogent** command**.**
drawpict(2);	% Presents buffer 2 (by default an empty buffer).

e.g. Present an image for exactly 1000ms:

preparestring('+', 2)	% Puts fixation cross in the middle of buffer 2.
preparestring('TREE', 1)	% Puts the word 'TREE' in the middle of buffer 1.
drawpict(1);	% Presents buffer 1.

wait(1000);	% Waits for 1000ms from the **start_cogent** command**.**
drawpict(2);	% Presents buffer 2 (by default an empty buffer).

In the first example, the word TREE in buffer 1 will be displayed until exactly 1000ms from time=0 in Cogent (i.e. until 1000ms since the **start_cogent** command). After this, the fixation cross will be displayed. In the second example, the word TREE will be displayed for exactly 1000ms (i.e. until a little more than 1000ms since the **start_cogent** command) and then the fixation cross will be displayed. The fixation cross will be displayed slightly later in the second example compared to the first example because the time required to prepare and display the stimulus (i.e. the **preparestring** and **drawpict** commands) is of the order of a few 10s of ms.

Other wait commands are:

waitsound(1);	Waits for the sound in sound buffer 1 to finish playing.
waitkeyup(*t, i*);	Waits a time *t* for the key with ID=*i* to be released. (See section 3.5).
waitkeydown(*t, i*);	As above for key presses.
waitmouse;	Waits for any mouse button to be pressed. (See section **3.7**).
waitserialbyte(*port_num***,** *t***,** *b***)**	Waits a time *time* (in ms) for a serial byte with value *b* to arrive at serial port *port_num*. (See section **3.11).**

There are some other functions in Cogent (in addition to **time**) that return the time at which the command was executed. e.g.

***t*=drawpict(1);**	Returns the time *t* at which buffer 1 was displayed.

Other commands are: **getmouse, getkeyup, getkeydown, lastkeyup, lastkeydown, waitkeyup, waitkeydown, waitserialbyte.**

3.9 How do I output data from Cogent 2000?

There are two ways of outputting information from Cogent 2000, using a log file or a results file. The difference between these two types of output file is that each line of a log file contains time-stamped information in a specific format whereas each line of a results file contains only information specified by the user. Results files are compatible with the Cogent 2000 data file format and therefore can also be used as input data files if required.

3.9.1 Log files

- A log file is specified using the **config_log** command (see section **3.2.2**). If no file name is specified, the log file will be given a default name **'Cogent-YYYY-MM-DD-HH-MM-SS.log'**. If an existing file name is given, the file will be appended.
- The contents of the log file are also written to the MATLAB command window for online monitoring.
- If the **config_log** command is not used, log commands will still send output to the MATLAB command window, but a log file will not be saved to disk.
- User specified information can be written to a log file using the **logstring** command. This information can be a string or a numerical value.
- Other commands that write information to a log file are **logkeys**, which writes information about keyboard activity (see section **3.5**) and **logserialbytes**, which writes information about bytes received from the serial port (see section **3.11**).
- The wait commands, **waitkeyup, waitkeydown** and **waitserialbyte** also log key presses and received serial bytes respectively.

Writing a log file

logstring(*string* **)**	Writes the string *string* to the log file and the MATLAB command window. e.g. **logstring**('Onset')
logstring(*t***);**	Writes the value *t* to the log file and the MATLAB command window. This can be useful for logging times associated with specific commands.

e.g. Log the time that an image is presented:

***t*=drawpict(1);**	Returns the time *t* at which buffer 1 was displayed.
logstring(*t***);**	Write this time to the log file**.**

NB It may also be useful to combine a value with a string. This can be done using the MATLAB command **sprintf** to define the string. e.g.

***string*=sprintf('Image at t=%8d',** ***t*);**	Assigns 'Image at t=*t*' to the variable *string* where the time *t* will be formatted as an numerical value.

Format of a log file

Each line of a log file contains several columns. The following lines describe the log file format.

- The first column is the total time (in ms) since Cogent was started (i.e. since the **start_cogent** command was executed).
- The second column (in square brackets) is the time (in ms) since information was previously logged (i.e. since a log command was previously executed).
- The third column contains either user-specified information (if using **logstring**), or information (i.e. time and data) about keyboard activity (for **logkeys**) or received serial bytes (for **logserialbytes**).

3.9.2 Results files

- A results file must be configured using the **config_results** command (see section 3.2.2). If no file name is specified, the results file will be given a default name **'Cogent-YYYY-MM-DD-HH-MM-SS.res'**.
- If an existing file name is given, the file contents will be overwritten. User specified information can be written to a results file using the **addresults** command. This information can be a string or a numerical value.
- Each line of a results file contains the information specified by a single call to the **addresults** command.

addresults(*v1***)**	Adds a line containing the value v1 to the results file.
addresults(*s1***)**	Adds a line containing the string s1 to the results file.
addresults(*v1, s1***)**	Adds a line containing both the value *v1* and the string *s1* to the results file. Multiple values and strings can be added to each line of the results file.

3.10 How do I read output files and load them into MATLAB?

Log files and results files can be read using a text-editor (e.g. the MATLAB **edit** command or **Windows NotePad**). However it is often useful to load the contents of log files and results files into MATLAB. This can be done using the Cogent command **loadlog**.

- **loadlog** can be used to load log files, results files and data files into MATLAB. This command loads the contents of the log file into a CELL ARRAY.
- The cell array is made up of one cell for each line of the log file. Each cell representing a line is made up of a number of cells containing a string representing each white-space separated string or value in that line of the log file.

The syntax for accessing the components of cell arrays is given for an example log file called *test.log*, whose contents are listed below:

```
Cog2000 log file Fri May 03 13:01:50 2002
GPrim v1.22 Compiled:Apr 30 2002 (GLib DirectDraw v7 Apr 30 2002)
Display:640x480x24 74.96Hz
Attaching C:\Cogent2000v1.22\toolbox\CogInput.dll
CogInput v1.04 Compiled: Apr  2 2002
18                          [18]     :     COGENT START
1625                        [1607]   :     Cat: 1625
```

a=**loadlog(** *'test.log'* **)**	Loads the contents of the log file *test.log* into a cell array defined by *a*.
b=*a*{7};	Sets *b* = line 7 with 5 cells containing the following strings: '1625' '[1607]' ':' 'Cat:' '1625'
c=*b*{1};	Sets the variable *c* equal to the first string in the line. i.e. *c* = '1625'. The same element can also be accessed using: *d*=*a*{7}{1}.

Other ways of loading results files

- If results file contains only numerical values and the same number of values on each line, it can be loaded into the MATLAB workspace using the MATLAB **load** command. Once loaded, it will be represented in MATLAB as an **n x m** matrix where the number of rows, **n** is equal to the number of lines in the results file and the number of columns, **m** is equal to the number of values on each line.
- Results files can also be loaded into Cogent 2000 using the **config_data** and **getdata** commands (see section **3.3**) in the same way as data files are loaded.

3.11 How do I control serial input and output?

All serial input and output commands require that the serial port has been configured using **config_serial** (see section **3.2.3**).

- Cogent can receive and send serial bytes to and from other equipment.
- Serial bytes can be systematically read and logged using **readserialbytes**, and **logserialbytes**.
- Serial bytes can be sent using **sendserialbytes.**
- Serial bytes can have values in the range 0-255.

readserialbytes(1) Reads all bytes arriving at the serial port 1 and empties the serial buffer for this port.

logserialbytes(1) Logs and time stamps all serial bytes that have been read by the last call to **readserialbytes.** The log is displayed in the MATLAB command window and saved to a log file if it has been defined by the **config_log** command. **logserialbytes** logs the time that all serial bytes are received as well as the time when the log command was executed.

sendserialbytes(1, 14) Sends a serial byte of value '14' to serial port 1.

- It is very common to use the arrival of serial bytes to control the program flow. For example the serial bytes may come from an MRI scanner and be used to synchronise the presentation of stimuli with image acquisition.
- The **waitserialbyte** command can wait for any byte or a byte of a specific value to arrive at the serial port. This command also logs all serial bytes arriving at the serial port and empties the corresponding serial buffer (in fact **waitserialbyte** internally executes **logserialbytes** and **readserialbytes**).
- The serial buffer can be cleared before testing for incoming bytes using the command **clearserialbytes**.

clearserialbytes(*p*); Empties the serial buffer for serial port number *p*.

waitserialbyte(*p*, *time*) Waits a time *time* (in ms) for any serial byte to arrive at serial port number *p*. When a byte arrives it is logged and the corresponding serial buffer is emptied. If no bytes arrive before *time* ms have elapsed, execution of the script continues.

waitserialbyte(*p, time*, *b*); Waits a time *time* (in ms) for a serial byte with value *b* to arrive at serial port number *p*. During this time all serial bytes arriving at the port are logged and the corresponding serial buffer is emptied. If no bytes arrive before *time* ms have elapsed, execution of the script continues.

e.g. Wait no more than 2 seconds for a specific value (e.g. 4) to arrive at serial port 1:

clearserialbytes(1); Empties serial buffer for serial port 1.

waitserialbyte(1, 2000, 4); Waits no more than 2000 ms for a byte with value 4. During this time all serial bytes arriving at the port are logged and the serial buffer is emptied. If a byte with value 4 doesn't arrive within this time, execution of the script continues.

Synchronising Cogent 2000 with an MRI scanner

To synchronise Cogent 2000 with hardware such as the MRI scanner, the hardware must be configured so that pulses are sent at specific times to the serial port of the PC running Cogent 2000. The **waitserialbyte** command can then be used to wait for a specific byte (as configured by the hardware) to arrive at the serial port.

For example, if the scanner causes a zero byte to be sent at a specific time to the PC running Cogent, the following commands can be used to wait for a zero byte to arrive at serial port 1:

clearserialbytes(1); Empties the serial buffer for serial port 1.

waitserialbyte(1, inf, 0); Waits forever for a 'zero' to arrive at serial port 1. When a byte arrives it is logged and the corresponding serial buffer is emptied.

Once the scanner synchronisation pulse (i.e. the zero byte) has been received, execution of the script will continue. The subject may then be presented with some stimuli and be expected to respond using hardware that sends serial bytes back to Cogent. The responses can be recorded using the **readserialbytes** and **logserialbytes** commands:

readserialbytes(1) Reads all bytes arriving at the serial port 1 and empties the serial buffer for this port.

logserialbytes(1) Logs and time stamps all key presses that have been read by the last call to **readserialbytes.**

Example scripts demonstrating the use of the serial port are **sample8.m** and **sample9.m** in the **Samples** sub-directory and chapter **4.**

3.12 How do I control parallel input and output?

Parallel input and output commands require the installation of an additional driver to access the I/O ports. A WinZip file called **UserPort.zip** contains all the necessary files and is included in the Cogent 2000 toolbox. This zip file is also available from: http://www.embeddedtronics.com/design&ideas.html.

Installing **UserPort.zip**

The contents of the WinZip file **UserPort.zip** can be extracted by double clicking on that file (this requires that WinZip has been installed (http://www.winzip.com). Extracting the contents of the WinZip file will result in a folder called **UserPort** containing all the necessary files and documentation. To access the parallel port from Cogent 2000, only the following files are important: **UserPort.sys** and **UserPort.exe**. The installation of these files is described below. More information is available in the documentation **UserPort.pdf**.

The parallel driver can be installed in the two ways:

- Copy **UserPort.sys** to %*WINDIR%***system32\drivers**.
- Double click on **Userport.exe**. A window containing I/O addresses will appear.
- Addresses can be added or removed from the window however, the default addresses should be correct for access to the parallel port.
- Click on the Start button.

Alternatively:

- Run **UserPort.exe** with the driver filename and path as an argument.
  i.e. **run UserPort.exe** *C: \UserPortDir***UserPort.sys**
- NB This can also be done by creating a shortcut.

The above steps should give access to the parallel ports that typically have decimal addresses of 888, 632 and 956. Parallel input and output can be controlled using **inportb** and **outportb**.

NB These functions will cause errors if used without any arguments.

k = **inportb**(*port_address*) Returns the decimal value of the bytes arriving at the port specified by *port_address*.
e.g. *k*=**inportb***(888)*

outportb(*port_address, num*) Send the number *num* to the specified port. *num* can be a value between 0 and 255.
e.g. **outportb**(*888, 10*)

The example script **sample10.m** in the **\Samples** sub-directory and chapter **4** demonstrates the use of the parallel port.

4 Tutorial and sample scripts

The sample scripts contained in the **\Samples** directory can be used as a tutorial to demonstrate how Cogent 2000. These sample scripts also provde a good a starting point for many stimuli.

sample1.m

sample1.m reads a list of words from a data file and presents them visually to the subject at a constant rate using the **wait** command. This script uses a full screen display in 'direct' colour mode with white letters on a black background. Comments are preceded by a '%' (these lines are not executed by MATLAB). The inter-trial interval in **sample1.m** will be slightly longer than 2000ms. This is because in addition to the specified delays it will take a small but finite time (in the order of a few 10s of ms) to execute the commands and wait for the next screen refreshes when the stimuli is displayed and cleared. This effect can be avoided by using references to absolute time (e.g. via the cogent command **waituntil**) rather than delays.

```
% SAMPLE1 – Visual presentation of words at a constant rate
config_display; % Configure display with default parameters;
config_data('sample1.dat'); %Read input data from sample1.dat

start_cogent;    % Configures MATLAB to run cogent commands. Also,
                   depending on prior configuration commands may
                   initialise keyboard and/or serial reaction time
                   recording, open full screen mode, start the real time
                   timer,etc.
for i = 1:countdatarows

% This is the standard MATLAB 'for' loop. countdatarows is a cogent
% function which returns the number of rows in the cogent data file.
% Thus the following statements, up and until the "end" statement, are
% executed once for each item in the data file.

     word = getdata( i, 1);

% getdata(r, c) returns the word at row, r, and column, c in the data
% file. In this case the data file contains only one column of words
% and so only i needs to vary. The 1 refers to the one and only column %
1 in the sample1.dat.

     clearpict( 1 );

% clearpict(1) fills buffer 1 with the current background colour,
% clearing any previous contents. This is not necessary for the
% first time round the loop but is needed subsequently to clear
% the previous word before preparing the next one. Note that this
% command does NOT affect the screen, only the memory buffer.

     preparestring( word, 1 );

% The current data item, stored in the MATLAB variable word is
% rendered in the middle of buffer 1. Again the subject does not see
% this.

     drawpict( 1 );

% The contents of buffer 1 are transferred to the screen and the
% subject finally sees the word appear. The transfer is synchronised
% with the next screen refresh so that the subject sees the whole
```

```
% screen appear at once.

     wait( 1000 );

% This command delays execution for the specified number of
% milliseconds. wait uses cogent's high precision timer for this
% purpose. It is better to use Cogent's timer, instead of the built-in
% matlab facilities, because these are synchronised with cogent's
% reaction-timing mechanisms.

     drawpict( 2 );

% To remove the word from the display another buffer, containing the
% default background colour is copied to the screen, again at the next
% screen refresh.

     wait( 1000 );

% The screen is left blank for one second before control returns to the
% beginning of the loop for the next presentation.

end

drawpict( 2);
% This is the end of the MATLAB for loop.
stop_cogent;

% MATLAB is returned to its normal mode.
```

sample2.m

sample2.m reads a list of picture files from a data file and presents them visually to the subject at a constant rate using the **wait** command. This script displays the pictures on a black background in a window. The display is set to 'direct' colour mode so the computer display must be set to 16 bits or more as described in section **3.2.1**.

```
% SAMPLE2 - Visual presentation of pictures at a constant rate
config_display( 0 )                 % Display in a window
config_data( 'sample2.dat' ); %Read input data from sample1.dat
start_cogent;
for i = 1:countdatarows
   file = getdata( i, 1 );    % Get name of picture file
   clearpict( 1 );            % Clear buffer 1
   loadpict( file, 1 );       % Load picture into buffer 1
   drawpict( 1 );             % Display buffer 1 and wait 2000ms
   wait( 2000 );
   drawpict( 2 );             % Clear screen and wait 1000ms
   wait( 1000 );
end
stop_cogent;
```

sample3.m

sample3.m is the same as sample1.m but demonstrates how to index colours when the display is set to 'palette' mode. The display must be set to 256 colours as described in **3.2.1.**

```
% SAMPLE3 - Visual presentation of words at a constant rate in palette mode
% Configure the display to use palette mode.
config_display(1, 1, 0, 1, 'Ariel', 60, 4, 8)
config_data( 'sample1.dat' ); % Specifiy data file to be 'sample1.dat;
start_cogent;
% Assign black and white to the palette indices 0 and 1
cgcoltab(0,[0 0 0; 1 1 1])
cgnewpal
for i = 1:countdatarows
   word = getdata( i, 1 );    % Get word from data file
   clearpict( 1 );         % Clear display buffer 1
   preparestring( word, 1 );  % Draw word in display buffer 1
   drawpict( 1 );          % Copy display buffer 1 to screen
   wait( 1000 );           % Wait for 1000ms
   drawpict( 2 );          % Clear screen
   wait( 1000 );              % Wait for 1000ms
end
drawpict (2 )
stop_cogent;
```

sample4.m

sample4.m is the same as sample3.m but sets the display to 'palette' mode and the screen to window mode. The display must be set to 256 colours as described in **3.2.1** for this script run.

sample5.m

sample5.m reads a list of words from a data file and presents them visually to the subject at absolute times determined by the **waituntil** command. The **waituntil** command waits until a specified time has elapsed since the **start_cogent** command. It is possible to use the time that one command has been executed to control when a following command will be executed. In **sample5.m**, all of the timing is calculated with reference to the time at which the fixation point is displayed between each word. This means that any of the small but finite times needed to execute commands and wait for screen refreshes will not alter the inter-trial interval of 2000ms. In this script, the stimuli, a code and the times at which the stimuli are displayed are logged to a file. The code may be used to identify the trial type.

```
% SAMPLE5 - Visual presentation of words using absolute time
config_display(1)
config_data( 'sample5.dat' );
config_log( 'sample5.log' ); % Configure log file
start_cogent;

preparestring( '+', 2 ); % Draw fixation point in display buffer 2
t0 = drawpict( 2 );      % Draw fixation point and get time

for i = 1:countdatarows

   code = getdata( i, 1 ); % Get code and word from data file
   word = getdata( i, 2 );

   clearpict( 1 );
   preparestring( word, 1 ); % Put word in buffer 1

   waituntil( t0 + 1000 ); % Display fixation point for 1000ms

   t1 = drawpict( 1 ); % Display word and get the time

   logstring( word );  % Log word.
   logstring( code );  % log code.
   logstring( t1 );    % Log time that word is displayed.

   % Wait until 1500ms after fixation was first presented
   waituntil( t0+1500 );

   % Clear screen and wait until 2000ms after fixation was first presented
   drawpict( 3 );
   waituntil( t0+2000 );
   t0 = drawpict( 2 );   % Display fixation point and get time
end

stop_cogent;
```

sample6.m

sample6.m reads a list of words from a data file and presents them visually to the subject at times controlled by a mixture of the **wait** and **waituntil** commands. This script reads and logs keyboard input between presentations of the fixation point using **clearkeys**, **readkeys** and **logkeys**. Keyboard input, words and times are logged to a file called **sample6.log**.

```
% SAMPLE6-Visual presentation of words, reading and logging keyboard input.
config_display(0)
config_keyboard;
config_data( 'sample6.dat' );
config_log( 'sample6.log' );

start_cogent;

preparestring( '+', 2 ); % Draw fixation point in display buffer 2

for i = 1:countdatarows

   word = getdata( i, 1 );
   clearpict( 1 );
   preparestring( word, 1 ); % Draw word in display buffer 1

   drawpict( 2 );  % Display fixation point and wait 1500ms
   wait( 1500 );

   clearkeys;    % Clear all key events
   t0 = drawpict( 1 ); % Display word and get time

   str=sprintf('%s: %d',word,t0);    % Log word and time it was displayed
   logstring(str);

   waituntil( t0+500 );    % Wait until 500ms after word was presented

   % Clear screen and wait until 1000ms after word was presented
   drawpict( 3 );
   waituntil( t0+1000 );

   % Read all key events since CLEARKEYS was called
   readkeys;

   % Write key events to log
   logkeys;
end

stop_cogent;
```

sample7.m

sample7.m presents words to the subject and calculates reaction times from keyboard responses using **getkeydown** and word presentation times using **time**. **getkeydown** returns information about all keys pressed since the last **readkeys**. The key ID, time in ms and number of keys pressed is returned. If there are multiple key presses, the response is recorded as if no keys were pressed. The reaction times and single key presses are written to a results file called **sample7.res** using **addresults**. Information is also written to a log file called **'Cogent-YYYY-MM-DD-HH-MM-SS.log'** where **YYYY-MM-DD-HH-MM-SS** is the date and time.

```
%SAMPLE7.M-Visual presentation of words, reading and logging keyboard
input. Reaction times calculated from keyboard input and written to a
results file.
config_display(0)
config_data( 'sample7.dat' );
config_log
config_results('sample7.res');
config_keyboard;
start_cogent;
preparestring( '+', 2 );   % Draw fixation point in display buffer 2
for i = 1:countdatarows
   code = getdata( i, 1 );
   word = getdata( i, 2 );
   logstring( code );
   logstring( word )
   clearpict( 1 );
   preparestring( word, 1 );  % Draw word in display buffer
   drawpict( 2 );          % Display fixation point and wait 1500ms
   wait( 1500 );
   drawpict( 1 );          % Display word
   t0 = time;              % Record time at which word is presented
   logstring( t0 );
   clearkeys;              % Clear all key events
   waituntil( t0+500 );    % Wait until 500ms after word was presented
   % Clear screen and wait until 1000ms after word was presented
   drawpict( 3 );
   waituntil( t0+1000 );
   readkeys;               % Read all key events
   logkeys;                % Write key events to log
   % Check key press and calculate the reaction time
   [ key, t, n ] = getkeydown;
   if n == 0               % no key press
      response = 0;
      rt = 0;
   elseif n == 1           % single key press
      response = key(1);
      rt = t(1) - t0;
   else
      response = 0;        % multiple key press
      rt = 0;
   end
   % Add the stimulus, reaction time and key press to the results file.
   addresults( word, response, rt );
end
stop_cogent;
```

sample8.m

sample8.m presents words, reads and logs serial input between presentations of the fixation point using **clearserialbytes**, **readserialbytes** and **logserialbytes**. Serial bytes, words and time are logged to a file called **sample8.log**.

```
% SAMPLE8.M - Visual presentation of words, reading and logging serial
bytes
config_display(0);
config_data( 'sample8.dat' );
config_log( 'sample8.log' );
config_serial( 1 )

start_cogent;

% Draw fixation point in display buffer 2
preparestring( '+', 2 );
for i = 1:countdatarows

   t0 = time;
   % Draw word in display buffer 1
   word = getdata( i, 1 );
   clearpict( 1 );
   preparestring( word, 1 );

   % Display fixation point and wait 1500ms
   drawpict( 2 );
   waituntil( t0+1500 );

   % Display word
   t1 = drawpict( 1 );

   % Log word and time
   str=sprintf('%s: %d',word,t1);
   logstring( str );

   % Clear all key events
   clearserialbytes( 1 );

   % Wait until 2000ms after start of trial
   waituntil( t0+2000 );

   % Clear screen and wait until 3000ms after start of trial
   drawpict( 3 );
   waituntil( t0+3000 );

   % Read all serial bytes since clearserialbytes was called
   readserialbytes( 1 );

   % Write serial events to log
   logserialbytes( 1 );

end

stop_cogent;
```

sample9.m

sample9.m uses **waitserialbyte** to repeatedly wait for zero bytes in order to synchronise the stimulus with incoming serial bytes (for example from an MRI scanner) and log subject responses. It is important to remember that **waitserialbyte** executes a **readserialbytes** followed by **logserialbytes**. Therefore the **waitserialbyte** in the loop will read and log all incoming bytes (from the scanner for example and the subject). When the loop ends, it is necessary to execute **readserialbytes** and **logserialbytes** to read and log any bytes arriving since the last **waitserialbytes**.

```
% SAMPLE9.M - Visual presentation synchronised with incoming serial bytes.
config_display(0)
config_serial;
```

```
config_log;
start_cogent;

preparestring( '.', 1 ); % Draw fixation point in display buffer 2
loadpict('ralf.bmp',2);
clearserialbytes(1); % Clear serial bytes buffer

% Repeat the following stimulus 10 times
% Show fixation, wait for scanner pulse, show stimulus, log response
for n=1:10
   drawpict( 1 );
   % Wait for a zero at serial port 1, log it, then empty serial buffer.
   % In the mean time, all other incoming bytes are also logged (ie any
   % subject responses).
   waitserialbyte(1,inf,0);
   t0=time; % Note the time and show stimulus
   t=drawpict( 2 );
   logstring('onset'); % log the onset
   % Show stimulus for 1.5 sec, then back to fixation.
   waituntil( t0+1500 );
   t=drawpict( 1 );
   logstring('offset'); % Log the offset
end

% Read and log any bytes that have arrived since the last waitserialbyte
readserialbytes(1);
logserialbytes(1);

% End of experiment
stop_cogent;
```

sample10.m

sample10.m presents visual stimuli and controls the parallel port. The names of the picture files and the binary codes used to control the parallel port are read from a data file. The binary codes are converted to a decimal using the MATLAB command **bin2dec**.

```
% SAMPLE10 - Visual presentation of pictures and control of parallel port.
config_display(0)
config_data( 'sample10.dat' );
start_cogent;

for i = 1:countdatarows

   % Get name of picture file
   file = getdata( i, 1 );
   % Get binary code
   code = getdata(i, 2);

   clearpict( 1 );

   % load picture into buffer 1
   loadpict( file, 1 );

   % Send signal to parallel port
   outportb(888,bin2dec(code));

   % Display buffer1 and wait 2000ms
   drawpict( 1 );
   wait( 2000 );

   % Clear screen and wait 500ms
   drawpict( 2 );
   wait( 1000 );
end

stop_cogent;
```

samplemouse.m

samplemouse.m demonstrates how Cogent 2000 reads mouse information. In this script, the current position of the mouse is updated and used to draw the word 'SQUEEK' at a new position. The mouse is in polling mode with a 10ms interval. Therefore every 10ms, the variable 'map' defined using **getmousemap**, is updated using **readmouse**. The updated contents of 'map' are extracted using **getmouse**. The size of the polling interval will determine how smoothly the word 'SQUEEK' moves around the display.

```
% SAMPLEMOUSE.M - Demonstrates how mouse position can be used to update
display.
% Configure mouse in polling mode with 10ms interval
config_mouse(10);
config_display(0);
start_cogent;

% Define variable map to contain current information about mouse
map = getmousemap;
x = 0;
y = 0;
while 1

   % Draw text at position defined by mouse (0,0 at first).
   clearpict( 1 );
   preparestring( 'SQUEEK!', 1, x, y );
   drawpict( 1 );

   % Update mouse map using readmouse.
   readmouse;

   % Update coordinates for text.
   x = x + sum( getmouse(map.X) );
   y = y - sum( getmouse(map.Y) );

   % Exit if left mouse button is pressed
   if ~isempty(getmouse(map.Button1)) & any(getmouse(map.Button1) == 128 )
      break;
   end
end

stop_cogent;
```

sound1.m

sound1.m sequentially loads the sound files listed in a data file and plays them. Each sound file is loaded only after the previous sound has finished playing. This is done using **waitsound**. A known bug means that the first execution of **waitsound** takes longer than subsequent executions. For this reason, a sound file is loaded and played before the trials begin.

```
% SOUND1.M - Plays sound stimuli loaded from a sound file.
config_display(0);
config_sound;
config_data('sound1.dat');
start_cogent;
% Load sound file and play it before trials begin to
% work around problem with waitsound.
file = getdata( 1, 1 );
loadsound( file, 1 );         %  Load sound file into buffer 1
playsound( 1);                %  Play sound in buffer 1
waitsound( 1);                %  Wait for sound to finish
for n=1:countdatarows
    file = getdata( n, 1 );       % Get name of sound file
    loadsound( file, 1 );         % Load sound file into buffer 1
    playsound( 1);                % play sound in buffer 1
    waitsound( 1);                % Wait for sound to finish
end
stop_cogent;
```

sound2.m

sound2.m creates sounds of increasing frequency using **preparepuretone.** The sounds are of 1000ms duration. The **waituntil** command is used to ensure constant trial length. As with the previous example, a sound is prepared and played to work around the bug in **waitsound**.

```
% SOUND2.M - plays a serious of tones created using preparepuretone.
config_display(0);
config_sound;
start_cogent;
% Create sound and play it before trials begin to
% work around problem with waitsound.
preparepuretone( 200 + 200, 1000, 1 );
playsound( 1 );
waitsound( 1 );
for i = 1:5
   t0=time
   preparepuretone( 200 + 200*i, 1000, 1 );
   playsound( 1 );
   waitsound( 1 );         % Wait for sound to finish playing
   time-t0
   waituntil(t0+2000 ); % Leave 2000ms between each sound
end

stop_cogent;
```

sound3.m

sound3.m presents words and sounds together, reads keyboard presses and presents feedback based on the key presses. A variable *keymap*=**getkeymap** is defined to determine whether the correct keys have been pressed. The variable *keymap* is a structure containing the key ID for each key. The syntax to get the ID corresponding to each key is

for example for the letter P, *keymap*.X. Responses and error codes are recorded in a results file.

```
% SOUND3.M - Presents visual stimuli, sound stimuli, reads key presses
% and presents feedback.
config_display( 0 ); % Configure hardware, input and output files
config_keyboard;
config_sound;
config_data( 'sound3.dat' );
config_log('sound3.log');
config_results('sound3.res');

start_cogent;

keymap = getkeymap; % Get map of keyboard to check for key presses
preparestring( '+', 2 ); % Draw fixation point in display buffer 2

% Prepare fixation sound (500ms, 200Hz sine wave) in sound buffer 2
preparepuretone( 500, 200, 2 );

for i = 1:countdatarows

   % Get data from file
   code      = getdata( i, 1 );
   word      = getdata( i, 2 );
   soundfile = getdata( i, 3 );

   % Load sound file into sound buffer 1
   loadsound( soundfile, 1 );

   % Draw word in display buffer 1
   clearpict( 1 );
   preparestring( word, 1 );

   % Display fixation point, play fixation sound and wait 1500ms
   drawpict( 2 );
   playsound( 2 );
   wait( 1500 );

   drawpict( 1 ); % Display word
   playsound( 1 ); % Play sound file
   t0 = time; % Record time at which word is presented
   logstring( t0 );
   clearkeys; % Clear all key events
   waituntil( t0+500 ); % Wait until 500ms after word was presented

   % Clear screen and wait until 2000ms after word was presented
   drawpict( 3 );
   waituntil( t0+2000 );

   % Read all key events since CLEARKEYS was called
   readkeys;
   logkeys; % Write key events to log

   % Check key pressed, set feedback, error code and reaction time
   n = countkeydown;
   [ key, t ] = lastkeydown;
   rt = 0;
   if n == 0
      % no key pressed
      message = 'no response';
      error = 1;
   elseif n > 1
      % multiple key pressed
      message = 'multiple key press';
      error = 2;
   elseif key ~= keymap.Q & key ~= keymap.P
      % keys other than Q or P pressed
```

```
      message = 'invalid key press';
      error = 3;
   elseif key == keymap.Q & code == 1
      % Q pressed when P should have been pressed
      message = 'wrong';
      error = 4;
   elseif key == keymap.P & code == 2
      % P pressed when Q should have been pressed
      message = 'wrong';
      error = 5;
   else
      % Correct key press
      message = 'correct';
      rt = t - t0;
      error = 0;
   end

   % Present feedback
   clearpict( 1 );
   preparestring( message, 1 );
   drawpict( 1 );
   wait( 1000 );

   % Clear screen
   drawpict( 3 );

   % Store code, reaction time and error code in results file
   addresults( code, rt, error );

   % Wait for 500ms
   wait( 500 );

end

stop_cogent;
```

flash30.m

flash30.m presents a flashing dartboard for 30 seconds then returns to fixation for 30 seconds. The computer display must be set up in palette mode to use the **paletteflickerrest** command and have the screen refresh frequency set to 60 Hz for 30 seconds of flashing.

```
% FLASH30.M - Flicker dartboard for 30 seconds and return to fixation
% spot for 30 seconds.
% The computer display must be in palette mode
% and the refresh frequency=60Hz (for 30 flashing).
config_display( 1, 1, [0 0 0], [1 1 1], 'Arial', 25, 4, 8 )

start_cogent

% prepare the dartboard
preparedartboard(1,20,200,10,18);
% Run paletteflickerrest to get fixation spot.
paletteflickerrest(1,0,0,0,32767);
% Wait for keyboard input
pause;

for i=1:5

   paletteflickerrest(1,4,4,225,32767);
   wait(30000);

end
stop_cogent
```

flash3.m

flash3.m presents a flashing dartboard for 3 seconds then returns to rest for 3 seconds. The computer display must be set up in direct colour mode to use the **paletteflicker** command with two buffers and have the screen refresh frequency set to 60 Hz for 3 seconds of flashing.

```
% FLASH3.M - Flicker dartboard for approx 3 seconds and rest for 3 seconds
% The computer display must be in direct color mode to use two buffers
% and the refresh frequency=60Hz (for 3 sec flashing).
config_display( 0, 1, [0 0 0], [1 1 1], 'Arial', 25, 4, 24)
start_cogent
% prepare the dartboard
preparedartboard([1 2],20,200,10,18);
% draw dartboard
paletteflicker([1 2],0,0,0,32767);
% Wait for keyboard input
pause;
for i=1:5
   paletteflicker([1 2],4,4,22,32767);
   wait(3000);
end
stop_cogent
```

5 Appendix I – List of Cogent 2000 functions

ADDRESULTS	add row of items to results file
CLEARKEYS	clears all keyboard events
CLEARMOUSE	clears mouse
CLEARPICT	clears a display buffer.
CLEARSERIALBYTES	clears bytes sent to serial port since last call to READSERIALBYTES
CONFIG_DATA	loads data file
CONFIG_DISPLAY	configures display
CONFIG_KEYBOARD	configures keyboard
CONFIG_LOG	configures log file
CONFIG_MOUSE	configures mouse
CONFIG_RESULTS	configures results file
CONFIG_SERIAL	configures serial port
CONFIG_SOUND	configures sound
COUNTDATAROWS	returns number of rows in cogent data file.
COUNTKEYDOWN	counts the number of key down events read in last call to READKEYS
COUNTKEYUP	counts the number of key up events read in last call to READKEYS
COUNTSERIALBYTES	returns the number of serial bytes read by READSERIALBYTES
DRAWPICT	copies the content of a display buffer to the screen.
GETDATA	get data item
GETKEYDOWN	returns the key IDs and times of key presses read by last call to READKEYS
GETKEYMAP	returns key IDs
GETKEYUP	gets the key IDs and times of key releases read in last call to READKEYS or CLEARKEYS
GETMOUSE	returns state of buttons and axis as read by READMOUSE
GETMOUSEMAP	return the mouse map which contains the index of axes and buttons
GETMOUSESTATE	gets mouse state (i.e. value of mouse buttons and axes)
GETRECORDING	get recording buffer and return as matrix.
GETSERIALBYTES	return values and times of serial bytes read by READSERIALBYTES
GETSOUNDFREQ	sets frequency of sound buffer
GETSOUNDVOL	gets volume of sound buffer
HELP_COGENT	lists one-line help on all Cogent functions.
LASTKEYDOWN	returns the key and time of the most recent key press
LASTKEYUP	returns the key and time of the most recent key release.
LASTSERIALBYTE	returns the value and time of the last byte to be read by READSERIALBYTES
LOADLOG	loads a text file and return an array of cell containing text field
LOADPICT	loads a bitmap and places the image in a display buffer
LOADSOUND	loads a wav file into sound buffer.
LOGKEYS	transfers all keyboard events read by READKEYS to log.
LOGSERIALBYTES	transfers serial bytes read by READSERIALBYTES to log file.

LOGSTRING	writes a time tag and string to the console and log file.
LOOPSOUND	starts a sound buffer playing in a continuous loop
PALETTEFLICKER	flickers a dartboard
PALETTEFLICKERREST	flickers a dartboard and returns to fixation
PAUSEKEY	pauses a key has been pressed and waits for another key press.
PAUSEMOUSE	execution of script if a mouse button has been press
PLAYSOUND	plays sound buffer
PREPAREDARTBOARD	places a dartboard in a display buffer
PREPAREPICT	draws a Matlab image matrix in a display buffer
PREPAREPURETONE	fill sound buffer with sin wave of specified duration and frequency
PREPARERECORDING	prepare recording buffer
PREPARESOUND	transfers a sound matrix from the matlab workspace to a Cogent sound buffer.
PREPARESTRING	places a string in a display buffer.
PREPAREWHITENOISE	fill sound buffer with white noise of specified duration
READKEYS	reads all keyboard events since last call to READKEYS or CLEARKEYS
READMOUSE	reads all mouse events
READSERIALBYTES	reads bytes sent to serial port since last call to READSERIALBYTES or CLEARSERIALBYTES.
RECORDSOUND	start recording sound
SENDSERIALBYTES	send bytes to serial port
SETFORECOLOUR	sets the foreground colour
SETPALETTECOLOURS	sets the colours used for each palette index
SETSOUNDFREQ	sets frequency of sound buffer
SETSOUNDPOSITION	sets current play position of sound buffer
SETSOUNDVOL	sets volume of sound buffer
SETTEXTSTYLE	sets font name and size
SOUNDPOSITION	returns current play position of sound buffer
START_COGENT	initialises Matlab for running Cogent 2000 commands.
STOPSOUND	stops a sound buffer playing.
STOP_COGENT	returns matlab from Cogent to normal mode.
TIME	returns current time in milliseconds since START_COGENT called.
WAIT	waits for a specified duration
WAITFRAME	wait for frame update
WAITKEYDOWN	waits for a key press
WAITKEYUP	waits for a key to be released.
WAITMOUSE	suspends execution until mouse button is clicked (i.e. pressed and then released)
WAITRECORD	wait for recording to finish
WAITSERIALBYTE	wait for byte to arrive on serial port
WAITSOUND	waits until a sound buffer has stopped playing.
WAITUNTIL	waits until specified time.

6 Appendix II – Further details about serial input

This section describes how the serial input commands work at a lower level.

The commands **config_serial** and **config_log** are required to configure the serial port and logfile.

The **start_cogent** command will now execute the low-level command cogserial('record') to start recording all serial bytes with time stamps to a serial buffer.

The **readserialbytes** command does the following:

1) Executes the low-level command CogSerial('GetEvents') which reads events from the serial buffer and then clears it.
2) Copies the events to the serial field of the cogent data structure cogent.serial (see). Any data already in the Cogent data structure will be overwritten. The following fields are written:

cogent.serial.value
cogent.serial.time
cogent.serial.number_of_responses

The **logserialbytes** command **c**opies the contents of cogent.serial to the logfile configured by config_log.

The **clearserialbytes** command reads and clears the serial buffer using the low-level command CogSerial('GetEvents'). It then clears the serial field of the Cogent data structure, cogent.serial.

The **waitserialbyte** command continuously executes **logserialbytes** followed by **readserialbytes** until the wait condition specified by this command is satisfied. Therefore while waiting for the specified condition to be satisfied, all serial bytes arriving at the serial port will be read, cleared and logged.

7 Appendix III – The cogent data structure

In Cogent 2000, a STRUCTURE ARRAY (called **cogent**) is used to communicate between high-level Cogent commands in the MATLAB workspace and low-level Cogent commands controlling stimulus presentation. This data structure is ordinarily local to the low-level commands and therefore not accessible to the MATLAB workspace. By declaring that the cogent data structure is GLOBAL (using the MATLAB command **global**), the information is also available to the MATLAB workspace. Therefore, if the line **global cogent** is included at the beginning of a Cogent 2000 script, all the information stored in this data structure is accessible.

The cogent data structure contains information (within FIELDS) about the current Cogent configuration. A field is created for each feature configured using the configuration commands (e.g. **config_display**). Each field is accessed using a '.'. (e.g. the display field of the cogent data structure can be accessed using: '**cogent.display**'). In turn each field may have one or more fields which are listed below:

config_data: cogent.data{}{} (This is a 2-dimensional cell array).

config_display: cogent.display.res, cogent.display.size, cogent.display.mode, cogent.display.bg, cogent.display.fg, cogent.display.font, cogent.display.fontsize, cogent.display.number_of_buffers, cogent.display.nbpp, cogent.display.scale.

config_sound: cogent.sound.nchannels, cogent.sound.nbits, cogent.sound.frequency, cogent.sound.number_of_buffers, cogent.sound.buffer.

config_keyboard: cogent.keyboard.id, cogent.keyboard.time, cogent.keyboard.value, cogent.keyboard.number_of_responses.

config_mouse: config_mouse.

config_log: cogent.log.filename, cogent.log.time.

config_results: cogent.results.filename.

config_serial: cogent.serial.name, cogent.serial.baudrate, cogent.serial.parity, cogent.serial.bytesize, cogent.serial.value, cogent.serial.time, cogent.serial.number_of_responses.

config_parallel: cogent.lpt.data, cogent.lpt.status.