---

void AbortCSS ()

Purpose: AbortCSS() can be used to abort out of a trial from a timing file. It is the same function that is used internally in Cortex when some huge error occurs (like the stack overflows), or if the user presses CTRL-BREAK three times to kill the running trials. It stops the clock and cleans up a bit by resetting some of the internal flags. There is no reason to use this function in a timing file under normal circumstances. Instead, it makes more sense to restructure the timing file so that the trial ends normally. Cortex automatically cleans up the necessary flags between trials.

Parameters: none

Returns: none

Platform: DOS and Windows

---

int**Platform:** DOS and Windows

---

float**Platform:** DOS and Windows

---

float asin (float **value**)

Purpose: find arc-sine

Returns: the arc-sine of **value**.

See also: acos(), atan(), atan2(), cos(), cosh(), sin(), sinh(), tan(), tanh()

Platform: DOS and Windows

---

float atan (float **value**)

Purpose: find arctangent

Returns: the arc-tangent of **value**.

See also: asin(), acos(), atan2(), cos(), cosh(), sin(), sinh(), tan(), tanh()

Platform: DOS and Windows

---

float atan2 (float **y,** float **x**)

Purpose: find arc-tangent of y/x

Returns: the arc-tangent of **y/x**. Handles equation correctly even if **x** is equal to zero.

See also: asin(), acos(), atan(), cos(), cosh(), sin(), sinh(), tan(), tanh()

Platform: DOS and Windows

---

float atof (pchar **string**)

Purpose: converts a string to a double

Returns: a float value converted from the **string.**

See also: atoi(), atol()

Platform: DOS and Windows

---

int atoi (pchar **string**)

Purpose: converts a string to an integer

Returns: an integer value converted from the **string.**

See also: atof(), atol()

Platform: DOS and Windows

---

long atol (pchar **string**)

Purpose: converts a string to a long

Returns: a long value converted from the **string.**

See also: atof(), atoi()

Platform: DOS and Windows

---

int BLOCKclear_stats(int **block_or_condition**, int **which_one**);

Purpose: clears the percent correct and circular buffer tables for a given block or condition.

Returns: 1 if valid block or condition number was made, 0 if not.

- **block_or_condition** (0 = block, 1 = condition, 2 = all blocks, 3 = all conditions)
- **which_one** (the block or condition number. If **block_or_condition** is 2 or 3, this value is ignored)

See also: BLOCKget_pct_correct(), BLOCKget_stats()

Platform: DOS and Windows

---

int BLOCKget_block_num()

Purpose: Gets current block number

Parameters: none

Returns: current block number

See also: BLOCKget_cond_num(), BLOCKset_next(), get_block_num(), get_cond_num()

Platform: DOS and Windows

---

int BLOCKget_cond_num()

Purpose: Gets current condition number

Parameters: none

Returns: current condition number

See also: BLOCKget_block_num(), BLOCKset_next(), get_block_num(), get_cond_num()

Platform: DOS and Windows

---

int BLOCKget_control_info (int **block**, pfloat **minPctOK**, pint **minTrials**, pint **recentOK**, pint **recentDone**, pint **max_errors**, pint **max_retries**, pint **recentOKtooLow**, pfloat **PctOKtooLow**)

Purpose: Gets the current values of these various variables for staircase design. At the end of each trial, these variables are used to check to see if the current block should be considered either unfinished (ie. may need to run some more depending on the parameters set in the Run:Parameters:Block/Repeat family of menus), correct (ie. finished and not to be run again), or aborted (not to be run again because the subject made too many mistakes). These are the exact names of the variables as shown in the Run:Parameters:Block/Repeat:Individual Blocks menu (with the menu length set to "full"). A value of zero for any one of these variables means that it is currently not being tested at the end of the trial.

Parameters:

- **block** the block number that is wished to be accessed [1-max_blocks]
- **minPctOK** the minimal percent correct threshold [if the block's current percent correct is higher than **minPctOK**, then the block will be considered correct]
- **minTrials** the number of trials that the block will run before testing other parameters such as minPctOK
- **recentOK** the number of correct trials required from the block's most **recentDone** trials before the block will be considered correct
- **recentDone** the number of the most recent trials that will be used to measure the monkey's progress using either **recentOK** or **recentOKtooLow**
- **max_errors** the maximum number of errors allowed before the block is considered aborted
- **max_retries** the maximum number of retries allowed when the on_error variable is set to Immediate_retry
- **recentOKtooLow** if the block's most recentDone trials have this many correct trials or less, the block will be aborted
- **PctOKtooLow** if the block's percent correct becomes this low or lower, the block will be aborted

Returns: 1 if valid block is specified, 0 if not

See also: BLOCKset_control_info()

Platform: DOS and Windows

---

int BLOCKget_max_vals (pint **max_cond)**

Purpose: Gets max_block and max_cond values. This could have been done via an external variable, but if these values were inadvertently changed, the system would crash.

Parameters: pint max_cond

Returns: the current maximum block values

See also: BLOCKset_next()

Platform: DOS and Windows

---

float BLOCKget_pct_correct (int **block_or_condition**, int **which_one**)

Purpose: Gets percent correct information for a given block or condition.

Parameters:

- **block_or_condition** (0 = block, 1 = condition)
- **which_one** (the block or condition number)

Returns: the percent correct information for a given block or condition, or -1 if unsuccessful.

See also: BLOCKclear_stats(), BLOCKget_stats(), get_block_pct_correct(), get_cond_pct_correct()

Platform: DOS and Windows

---

int BLOCKget_stats (int **block_or_condition**, int **which_one**, pint **num_correct**, pint **num_trials**, ppchar **circular_buffer**)

Purpose: gets the circular buffer, number of correct trials, and the number of total trials for a given block condition.

Parameters:

- **block_or_condition** (0 = block, 1 = condition)
- **which_one** (the block or condition number [1-max_blocks/conditions])
- **num_correct** (the number of correcttrials [so far] in the current block/condition)
- **num_trials** (the number of total trials [so far] in the current block/condition)
- **circular_buffer** (the results from the last N number of trials from the given block/condition. To set the value of N, see the Run:Parameters:Block/Repeat:Sizing menu)

Returns: a non zero value if valid selection is made, 0 if pointers are not yet set

See also: BLOCKclear_stats(), BLOCKget_pct_correct(), get_block_pct_correct(), get_cond_pct_correct()

Platform: DOS and Windows

---

int BLOCKset_control_info (int **block**, float **minPctOK**, int **minTrials**, int **recentOK**, int **recentDone**, int **max_errors**, int **max_retries**, int **recentOKtooLow**, float **PctOKtooLow**)

Purpose: sets the current values of these various variables for staircase design. At the end of each trial, these variables are used to check to see if the current block should be considered either unfinished (ie. may need to run some more depending on the parameters set in the Run:Parameters:Block/Repeat family of menus), correct (ie. finished and not to be run again), or aborted (not to be run again because the subject made too many mistakes). These are the exact names of the variables as shown in the Run:Parameters:Block/Repeat:Individual Blocks menu (with the menu length set to "full"). A value of zero for any one of these variables means that it is currently not being tested at the end of the trial.

Parameters:

- **block** (the block number that is wished to be accessed [1-max_blocks])
- **minPctOK** (the minimal percent correct threshold [if the block's current percent correct is higher than **minPctOK**, then the block will be considered correct])

- **minTrials** (the number of trials that the block will run before testing other parameters such as **minPctOK**)
- **recentOK** (the number of correct trials required from the block's most **recentDone** trials before the block will be considered correct)
- **recentDone** (the number of the most recent trials that will be used to measure the monkey's progress using either **recentOK** or **recentOKtooLow**)
- **max_errors** (the maximum number of errors allowed before the block is considered aborted)
- **max_retries** (the maximum number of retries allowed when the **on_error** variable is set to **Immediate_retry**)
- **recentOKtooLow** (if the block's most **recentDone** trials have this many correct trials or less, the block will be aborted)
- **PctOKtooLow** (if the block's percent correct becomes this low or lower, the block will be aborted)

Returns: 1 if valid block is specified, 0 if not

See also: BLOCKget_control_info()

Platform: DOS and Windows

---

int BLOCKset_next (int **block**, int **condition**)

Purpose: sets the **block** and **condition** to be run in the next trial

Parameters:

- **block** (the next **block**--remember that blocks are numbered 1-max_blocks)
- **condition** (the next **condition** within **block**--remember that conditions are numbered 1-max_conds)

Returns: 1 if successful, 0 if not.

See also: break_fixation_error()

Platform: DOS and Windows

---

void break_fixation_error()

Purpose: Records in the data file that the monkey has broken fixation

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void byte_c_out (int **byte**)

Purpose: Write the given **byte** to Port C of the PIO24 board. (Note that thisfunction will not work for the PIO24 portion of the CIO-DAS1602/12 or the PCI-DAS1602/12 boards. For those boards, please use the DEVoutp() function.)

Parameters: byte - a single byte (8 bits) of data

Returns: nothing

Platform: DOS and Windows

---

void byte_out (int **byte**)

Purpose: Write the given **byte** to Port C of the PIO24 board. (Note that thisfunction will not work for the PIO24 portion of the CIO-DAS1602/12 or the PCI-DAS1602/12 boards. For those boards, please use the DEVoutp() function.)

Parameters: byte - a single byte (8 bits) of data

Returns: nothing

Platform: DOS and Windows

---

pchar calloc (int **num_elements**, int **bytes_per_element**)

Purpose: allocates an array in memory with elements initialized to 0.

Parameters:

- **num_elements** (the number of elements or size **bytes_per_element** to allocate)
- **bytes_per_element** (the size of each element in bytes)

Returns: a pointer to the first element in the array.

See also: free(), malloc(), realloc()

Platform: DOS and Windows

---

float cart2r (float **x**, float**y**)

Purpose: A Cartesian to polar transform which computes the magnitude (rho) value for the given **x**, **y** cartesian coordinates.

Parameters: x and **y** are the Cartesian cooridinates.

Returns: the magnitude.

Platform: DOS and Windows

---

float cart2theta (float **x**, float **y**)

Purpose: A Cartesian to polar transform which computes the phase angle theta (in degrees) for a given **x**, **y** coordinate.

Parameters: x, **y** are the Cartesian coordinates

Returns: The phase angle theta polar value.

Platform: DOS and Windows

---

void cartesian2polar (float **x**, float **y**, pfloat **r**, pfloat **theta**)

Purpose: A Cartesian to polar transform which computes the phase angle **theta** (in degrees) and the magnitude **r**, for a given **x**, **y** coordinate.

Parameters:

- **x** and **y** are the Cartesian coordinates
- **r** - pointer to **r** which will store the magnitude
- **theta** - pointer to **theta** which will store the angle

Returns: nothing

Platform: DOS and Windows

---

int ceil (float **value**)

Purpose: Calculates the ceiling of a value.

Retuns: the smallest integer that is greater than or equal to **value**.

See also: floor()

Platform: DOS and Windows

---

int chdir (pchar **new_dir_name**)

Purpose: changes the current working directory to **new_dir_name**

Returns: 0 if successful.

See also: getcwd(), mkdir(), rename(), rmdir()

Platform: DOS and Windows

---

int chmod (const char ***filename**, int **pmode**)

Purpose: Change the file-permission settings. The _chmod function changes the permission setting of the file specified by filename. The permission setting controls read and write access to the file.

Parameters:

- **filename** Name of exisiting file
- **pmode** Permission setting for file

Returns: Each of these functions returns 0 if the permission setting is successfully changed. A return value of -1 indicates that the specified file could not be found, in which case errno is set to ENOENT.

Platform: DOS and Windows

---

int chsize (int **handle**, long **size**)

Purpose: Changes the file size.

Parameters:

- handle Handle referring to open file
- size New length of file in bytes

Returns: _chsize returns the value 0 if the file size is successfully changed. A return value of -1 indicates an error: errno is set to EACCES if the specified file is locked against access, to EBADF if the specified file is read-only or the handle is invalid, or to ENOSPC if no space is left on the device.

Platform: DOS and Windows

---

void ClearCSSGlobals ()

Purpose: Clears all the external variables that can be set by the user.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void clear_eog()

Purpose: clear the eog display window

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

int clip (int **value**, int **lower_limit**, int **upper_limit**)

Purpose: clip the range of an integer

Parameters:

- **value** to be clipped
- **lower_limit** of desired range
- **upper_limit** of desired range

Example:

a = clip (*a*,100,500);
returns *a* if 100 < *a* < 500
100 if *a* <= 100
500 if *a* >= 500

See also: fclip()

Platform: DOS and Windows

---

long clock()

Purpose: Calculates the processor time used by the calling process.

Parameters: none

Returns: clock returns the number of clock ticks of elapsed processor time. The returned value is the product of the amount of time that has elapsed since the start of a process and the value of the CLOCKS_PER_SEC constant.

Comments: The clock function tells how much processor time the calling process has used. The time in seconds is approximated by dividing the clock return value by the value of the CLOCKS_PER_SEC

constant In Microsoft and Watcom C/C++, the value of CLOCKS_PER_SEC is 1000. Note: In MS-DOS, clock() returns the time elapsed since the process started. This may not be equal to the actual processor time used by the process.

Platform: DOS and Windows

---

int close (int **handle**)

Purpose: closes the file indicated by **handle**

Returns: 0 if successful.

See also: dup(), dup2(), open()

Platform: DOS and Windows

---

void Cls ()

Purpose: clears the screen to black and resets the cursor position to the top left corner (0, 0). Works in text mode only (SCREENmode(1)).

Parameters: none

Returns: nothing

See also: SCREENmode()

Platform: DOS and Windows

---

int CLTactivate (int **num_entries**, int **CLTnum**, int **src_start**, int **dst_start**)

Purpose: Transfer color data from temporary to active CLTs (color lookup tables). The graphics board only has one color palette in use at a time (i.e., the active CLT). To optimize the speed of lookup table operations, any other temporary color palettes must reside in system memory. When this function call is executed, the given temporary CLT is activated on the graphics board.

Parameters:

- **num_entries Platform:** DOS and Windows

  ---

  int CLTdownload (int **CLTnum**, int **CLTstart_idx**, int **num_entries**, pchar **data**)

  Purpose: Copies color data from the given color lookup table (CLT), into the **data** variable. Parameters:

  - **CLTnum** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.
  - **CLTstart_idx** - the starting index in the CLT
  - **num_entries** - the number of entries that should be downloaded
  - **data** - internally, this parameter is a pointer to an array of BYTE_RGB structures to hold the red, green and blue values.

    typedef struct {
    Platform: DOS and Windows

    ---

int CLTget_val (int **CLTnum**, int **index**, int ***r**, int ***g**, int ***b**)

Purpose: Gets the R, G, B values for a single entry of a given color lookup table (CLT).

Parameters:

- **CLTnum** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.
- **index** - the index into the CLT for the color entry to get
- **r** - pointer to the red value at the given index
- **g** - pointer to the green value at the given index
- **b** - pointer to the blue value at the given index

Returns: 1 if successful, otherwise 0.

Platform: DOS and Windows

---

int CLTretrieve (int **num_entries**, int **CLTnum**, int **src_start**, int **dst_start**)

Purpose: This function transfers data from the graphics board's active palette to a temporary CLT in system memory. When using two computers, this is faster than GDPget_CLTS and CLTupload() to set the values in a palette. Also more convenient for CSS, which doesn't support structures, although it is equally possible to use GDPget_CLTs and know that each element is a byte, and that they are in RGB order.

Parameters:

- **num_entries** - the number of entries that should be retrieved
- **CLTnum** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.
- **src_start** - the starting index in the CLT
- **dst_start** - the ending index in the CLT

Returns: 1 if successful, otherwise 0.

Platform: DOS and Windows

---

int CLTset_val (int **CLTnum**, int **index**, int **r**, int **g**, int **b**)

Purpose: Sets the R, G, B values for a single entry of a give color lookup table (CLT).

Parameters:

- **CLTnum** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.
- **index** - the index into the CLT for the color entry to set
- **r** - the red value to set at the given index
- **g** - the green value to set at the given index
- **b** - the blue value to set at the given index

Returns: 1 if successful, otherwise 0.

Platform: DOS and Windows

---

int CLTupload (int **CLTnum**, int **CLTstart_idx**, int **num_entries**, pchar **data**)

Purpose: Copies color data from the data structure into the given color lookup table (CLT).

Parameters:

- **CLTnum** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.
- **CLTstart_idx** - the starting index in the CLT
- **num_entries** - the number of entries that should be uploaded
- **data** - internally, this parameter is a pointer to an array of BYTE_RGB structures which holds the red, green and blue values.

  typedef struct {
  Platform: DOS and Windows

---

int CMENUbool (char ***msg**)

Purpose: Prints the **msg** string on the screen, then waits for the user's response. The **msg** string is usually a question requiring a Yes/No answer.

Parameters: msg, the string to be printed on the screen

Returns: 1 if the user responds "Yes", 0 if the user responds "No", and -1 if ESCAPE is pressed.

Platform: DOS only

---

int CMENUboolRC (char ***msg**, char ***choices**)

Purpose: Prints the **msg** string on the screen, then waits for the user's response. The **msg** string is a question that requires an answer which is one of the **choices** provided.

Parameters:

- **msg** - the string to be printed on the screen as a question
- **choices** - an array of strings containing the choices of responses

Returns: 1 if a new value has been set, 0 if no change, and -1 if ESCAPE pressed.

Platform: DOS only

---

int CMENUrun (char ***message**, int **y**, int ***val**, int **num_choices**, char ***choices[]**)

Purpose: Prints the **message** string on the screen, at location (1, y), and then waits for the user\\\'s response. The **message** string is a question that requires an answer that is one of the **choices** provided.

Parameters:

- **message** - the string to be printed on the screen as a question
- **y** - the vertical location at which the string will appear
- **val** - pointer to the value chosen
- **num_choices** - the number of choices
- **choices** - an array of strings containing the choices of responses

Returns: 1 if a new value has been set, 0 if no change, and -1 if ESCAPE pressed.

Platform: DOS only

---

void collect_data (int **on_off**)

Purpose: instructs CORTEX to either begin or stop collecting spike data to place in the data file.

Parameter: BOOL (0 = stop collecting data; 1 = collect data)

Returns: nothing

Platform: DOS and Windows

---

int contact (int **cirx**, int **ciry**, int **cir_radius**, int **ulx**, int **uly**, int **lrx**, int **lry**)

Purpose: identifies when a circle and a rectangular object overlap each other

Parameters:

- **cirx** center of circle, x coordinate
- **ciry** center of circle, y coordinate
- **cir_radius** radius of circle
- **ulx** upper left corner, x coordinate of bar
- **uly** upper left corner, y coordinate of bar
- **lrx** lower right corner, x coordinate of bar
- **lry** lower right corner, y coordinate of bar

Returns: 1 if circle and rectangle overlap, 0 if no overlap.

Platform: DOS and Windows

---

float cos (float **value**)

Purpose: find the cosine of a float

Returns: the cosine of **value**.

See also: acos(), asin(), atan(), atan2(), cosh(), sin(), sinh(), tan(), tanh()

Platform: DOS and Windows

---

float cosh (float **value**)

Purpose: find the hyperbolic cosine of a float

Returns: the hyperbolic cosine of value.

See also: acos(), asin(), atan(), atan2(), cos(), sin(), sinh(), tan(), tanh()

Platform: DOS and Windows

---

void CurMov (int **row**, int **column**)

Purpose: sets the current text position to the display point (**row**, **column**). This call works in text mode only.

Returns: nothing

See also: SCREENmode(), printxy(), printf()

Platform: DOS and Windows

---

int DEVinp (int **device_number**, int **port)**

Purpose: Reads a byte from the given **port** and **deive_number**.

Parameters:

- **device_number** (set in CORTEX.CFG)
- **port** (the **port** on **device_number** to output the **data** through)

Returns: a single byte read from **port** (a part of **device_number**).

Comments: Each device listed in the CORTEX.CFG file has a corresponding device_number. The first device listed is device number zero. The *n*th device is device number *n*-1. Ports are numbered much the same way (base-0). The first parallel port on a parallel device will be port number 0, and the *n*th will be port number *n*-1 (Port A = **port** 0).

See also: DEVinpw(), DEVoutp(), DEVoutpw()

Platform: DOS and Windows

---

int DEVinpw (int **device_number**, int **port**)

Purpose: Reads two bytes from the given **port** and **device_number**

Parameters:

- **device_number** (set in CORTEX.CFG)
- **port** (the **port** on **device_number** to output the **data** through)

Returns: two bytes (16 bits) read from **port** (a part of **device_number**).

Comments Each device listed in the CORTEX.CFG file has a corresponding device_number. The first device listed is device number zero. The *n*th device is device number *n*-1. Ports are numbered much the same way (base-0). The first parallel port on a parallel device will be port number 0, and the *n*th will be port number *n*-1 (Port A = **port** 0). **See also:** DEVinp(), DEVoutp(), DEVoutpw()

Platform: DOS and Windows

---

int DEVoutp (int **device_number**, int **port**, int **data**)

Purpose: outputs a single byte of **data** on **port** (a part of **device_number**), the **data** sent if successful, else returns -1

Parameters:

- **device_number** (set in CORTEX.CFG)
- **port** (the **port** on **device_number** to output the **data** through)
- **data** (a single byte (8 bits) of data)

Comments: Each device listed in the CORTEX.CFG file has a corresponding

device_number. The first device listed is device number zero. The *n*th device is device number *n*-1. Ports are numbered much the same way (base-0). The first parallel port on a parallel device will be port number 0, and the *n*th will be port number *n*-1 (Port A = **port** 0).

See also: DEVinp(), DEVinpw(), DEVoutpw()

Backward Compatibility: byte_out()

Platform: DOS and Windows

---

int DEVoutpw (int **device_number**, int **port**, int **data**)

Purpose: Writes two bytes of **data** to the given **port** and **device_number**.

Parameters:

- **device_number** (set in CORTEX.CFG)
- **port** (the **port** on **device_number** to output the **data** through)
- **data** (two bytes (16 bits) of data)

Returns: outputs two bytes of **data** on **port** (a port of **device_number**). Returns the **data** sent if successful, else returns -1.

Comments: Each device listed in the CORTEX.CFG file has a corresponding device_number. The first device listed is device number zero. The *n*th device is device number *n*-1. Ports are numbered much the same way (base-0). The first parallel port on a parallel device will be port number 0, and the *n*th will be port number *n*-1 (Port A = **port** 0).

See also: DEVinp(), DEVoutpw(), DEVoutp()

Platform: DOS and Windows

---

void display_eye_path (int **visible**)

Purpose: show/unshow the path of the eye movement up to a given point in a trial. display_eye_path(1) will draw the eye path, and display_eye_path(0) will erase it.

Parameters: visible or invisible (1 = visible, 0 = invisible)

Returns: nothing

See also: put_data_in_eye_buf()

Backward Compatibility: display_eye_buf()

Platform: DOS and Windows

---

void display_fixspot (int **visible**)

Purpose: turns on or off the fixation spot, turns on (**visible**=1) or off (**visible**=0) the fixation spot

Parameters: visible or invisible (1 = visible, 0 = invisible)

Returns: nothing

See also: display_test(), Gon_off()

Platform: DOS and Windows

---

void display_histogram ()

Purpose: causes the histogram for the current condition to be displayed (typically called at the start of the trial)

Parameters: none

Returns: nothing

See also: display_trial_progress(), update_histogram()

Platform: DOS and Windows

---

void display_play (int **visible**)

Purpose: turns on (**visible**=1) or off (**visible**=0) the mapping stimulus in play mode

Parameters: visible or invisible (1 = visible, 0 = invisible)

Returns: nothing

See also: display_test(), Gon_off()

Platform: DOS and Windows

---

void display_sample (int **visible**)

Purpose: turns on (**visible**=1) or off (**visible**=0) the sample stimulus (the sample stimulus is defined as the item(s) in TEST0 of the current conditions file)

Parameters: visible or invisible (1 = visible, 2 = invisible)

Returns: nothing

See also: display_test(), Gon_off()

Platform: DOS and Windows

---

void display_test (int **test_screen**, int **visible**)

Purpose: turns on (**visible**=1) or off (**visible**=0) a specified test_screen (TEST1 through TEST9)

Parameters:

- test stimulus number
- visible or invisible (1= visible, 0 = invisible)

Returns: nothing

See also: display_fixspot(), display_play(), display_sample(), Gon_off()

Platform: DOS and Windows

---

void display_trial_progress (int **show_progress**)

Purpose: turns on (**show_progress**=1) or off (**show_progress**=0) the current trial's raster (the progress line) below the histogram display. Also turns on or off the placement of data into the cumulative on-line histogram. Has no effect on the raw data collection. The function is normally turned off while waiting either a random amount of time or waiting for the subject to do something.

Parameters: 0 = turn progress line off, 1 = turn it on

Returns: nothing

See also: display_histogram(), update_histogram()

Platform: DOS and Windows

---

float distance_to_line (float **x**, float **y**, float **slope**, float **DC**)

Purpose: Finds the minimum distance between a line and a point. This function is generally used to determine the amount of error the subject has made during a saccade towards a target (that lies on a line of **slope** from the origin).

Parameters:

- **x** (location along the horizontal axis of the point to be tested)
- **y** (location along the vertical axis of the point to be tested)
- **slope** (the slope of the line that passes through the origin to be tested)
- **DC** (the denominator constant of the line that passes through the origin)

Returns: the minimum distance between a line of **slope** passing through the origin (the center of the screen; 0 degrees, 0 degrees) and a point defined by (x degrees, y degrees).

See also: EYEget_dva(), find_DC(), find_slope(), in_corridor()

Platform: DOS and Windows

---

void dont_unload_conds (**void**)

Purpose: To prevent the program from unloading the current condition's worth of graphics information and to prevent the next set from being loaded.

Parameters: none.

Returns: nothing.

Comments: This function may decrease the time needed between trials (especially if the stimuli are complex), but should only be used when every trial uses the same set of graphical items and every condition uses the same items in each test_screen. The dont_unload_conds flag is reset every trial, so this routine must be called every trial if you want the same graphical environment to persist over many trials. Also, if this called on the LAST trial of a run, the graphical environment will persist into the next run

(unless you set the option in the Run:Parameters:General menu or quit CORTEX in between runs).

Platform: DOS and Windows

---

void DrawBox (float **X_center**, float **Y_center**, float **width**, float **height**, int **color**)

Purpose: draws onto the EOG_DISPLAY a box centered at (**X_center**, **Y_center**), of **width** width, **height** height and **color** color (#include css_inc.h in your state function to use its list of colors)

Parameters:

- X_center: FLOAT degrees of visual angle.
- Y_center: FLOAT degrees of visual angle.
- width: FLOAT degrees of visual angle.
- height: FLOAT degrees of visual angle.
- color: INT see css_inc.h (include it in your state function)

Returns: nothing.

See also: ITEM_POSmark_pos()

Backward Compatibility: mark_screen_pos()

Platform: DOS and Windows

---

long dsquared (int **x1**, int **y1**, int **x2**, int **y2**)

Purpose: Finds the square of the distance between two points, (**x1**, **y1**) and (**x2**, **y2**).

Parameters:

- **x1**, **y1 Platform:** DOS and Windows

  ---

  int dup (int **handle**)

  Purpose: creates a second file handle for a currently open file.

  Parameters: Takes the handle of the currently open file.

  Returns: the new file handle

  See also: close(), dup2(), open()

  Platform: DOS and Windows

  ---

  int dup2 (int **handle_1**, int **handle_2**)

  Purpose: forces **handle_2** to refer to a currently open file (referred to by **handle_1**)

  Parameters:

  - handle to open file
  - handle to be changed to refer to the current open file

Returns: 0 if successful.

See also: close(), dup(), open()

Platform: DOS and Windows

---

void encode (int **EVENT_CODE**)

Purpose: Records an event code in the cortex data file

Parameters: event code

Returns: nothing

See also: EVENT_CODE

Platform: DOS and Windows

---

void end_trial ()

Purpose: cleans up after the trial and turns off the mapping stimulus in play mode

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

int eof (int **handle**)

Purpose: checks if the current position within the file referred to by handle is end_of_file

Returns: 1 if currently at end_of_file, 0 if not

Platform: DOS and Windows

---

int EPPconvert (int **x**, int **chan**)

Purpose: takes a 12-bit, signed integer value, **x**, and a channel number, **chan**, and converts them to the format which the EPP buffer expects.

Parameters:

- **x** - data value
- **chan** - channel number

Returns: a 16-bit value (short integer) containing the 12-bit data value, and 4-bits for the channel number

Platform: DOS and Windows

---

int EPPget_chan (int **x**)

Purpose: Takes the 16-bit EPP value, **x**, and returns the 4-bit channel number

from it.

Parameters: **x**, a 16-bit value containg the 12-bit data value and the 4-bit channel number

Returns: the channel number

Platform: DOS and Windows

---

int EPPunconvert (int **x**)

Purpose: Extracts the 12-bit integer data value from the EPP buffer storage format.

Parameters: **x**, a 16-bit value containg the 12-bit data value and the 4-bit channel number

Returns: the data value

Platform: DOS and Windows

---

float exp (float **value**)

Returns: $e^{\mathbf{value}}$

See also: log(), log10(), pow()

Platform: DOS and Windows

---

void EYEget_dva (pfloat **X**, pfloat **Y**)

gets the eye position in degrees of visual angle (dva)

Purpose: gets the eye position in degrees of visual angle (dva)

Parameters:

- pointer to **X** in which to store horizontal position relative to (0, 0)
- pointer to **Y** in which to store vertical position relative to (0, 0)

Returns: nothing

Backward Compatibility: f_get_X(), f_get_Y(), get_fixation_posX(), get_fixation_posY()

Platform: DOS and Windows

---

void EYE_WINcopy (int **eye_window_number**, int **test_screen**, int **item_position**)

Purpose: copies one ITEM_POS or EYE_WIN's center and size into a new EYE_WIN. Values remain until CORTEX is exited or EYE_WINreset() called.

Parameters:

- **eye_window_number** (1-EyeWinMax--set in cortex.cfg)

- **test_screen** (0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **item_position** within that test_screen (1-x)

Returns: nothing

See also: EYE_WINreset(), EYE_WINset()

Platform: DOS and Windows

---

void EYE_WINreset ()

Purpose: clears all of the saved EYE_WINs from the EYE_WIN scratch buffer. Otherwise they remain until CORTEX is exited.

Parameters: none

Returns: nothing

See also: EYE_WINcopy(), EYE_WINset()

Platform: DOS and Windows

---

void EYE_WINset (int **eye_window_number**,float **x_center**,float **y_center**,float **x_size**,float **y_size**)

Purpose: stores a position (center and size) for future reference.

Parameters:

- **eye_window_number** (1-EyeWinMax--set in cortex.cfg)
- **x_center** (upon calling function, this pointer will store the x_center value)
- **y_center** (upon calling function, this pointer will store the y_center value)
- **x_size** (upon calling function, this pointer will store the x_size value)
- **y_size** (upon calling function, this pointer will store the y_size value)

Returns: 1 if successful, 0 if an invalid selection

Comments: Values remain until CORTEX is exited or EYE_WINreset() is called.

See also: EYE_WINcopy(), EYE_WINreset()

Backward Compatibility: set_position()

Platform: DOS and Windows

---

float fabs (float **value**)

Returns: the absolute value of value

See also: abs()

Platform: DOS and Windows

---

float fclip (float **value**, float **lower_limit**, float **upper_limit**)

Purpose: clips the range of a floating point **value**.

Parameters:

- **value** - number to be clipped
- **lower_limit** - lower end of range
- **upper_limit** - upper end of range

Returns: either the original value, the **lower limit**, or the **upper limit**.

Example: a = clip (a,100.3,500.1);
returns a if 100.3 < a <500.1
100.3 if a <= 100.3
500.1 if a = 500.1

See also: clip()

Platform: DOS and Windows

---

int fclose (plong **fp**)

Purpose: closes the open file handle **fp**.

Parameter: fp - pointer to file

Returns: 0 if the file is successfully closed, and non-zero to indicate an error.

Platform: DOS and Windows

---

int feof (plong **fp**)

Purpose: determines whether the end-of-file has been reached for the file pointed to by **fp**.

Parameter: fp - pointer to file

Returns: a nonzero value after the first read operation that attempts to read past the end of the file. It returns 0 if the current position is not end of file.

Platform: DOS and Windows

---

int ferror (plong **fp**)

Purpose: tests for a reading or writing error on the file associated with **fp**.

Parameter: fp - pointer to file

Returns: If no error has occurred on the file, ferror returns 0. Otherwise, it returns a nonzero value.

Platform: DOS and Windows

---

int fflush (plong **fp**)

Purpose: flushes a stream. If the file associated with the stream is open for output, fflush causes any unwritten data to be written to the file. If the file fp is open for input or update, the fflush function undoes the effect of any preceding ungetc operation on the stream. If the value of fp is NULL, then all files that are

open will be flushed.

Parameter: fp - pointer to file

Returns: fflush returns 0 if the buffer was successfully flushed. A non-zero return value indicates an error.

Platform: DOS and Windows

---

int fgetc (plong **fp**)

Purpose: gets the next character from the file designated by fp.

Parameter: fp - pointer to file

Returns: the character read as an int or return EOF to indicate an error or end of file.

Platform: DOS and Windows

---

pchar fgets(pchar **buf**, int **n**, plong **fp**)

Purpose: gets a string of characters from the file designated by **fp** and stores them in the array pointed to by **buf**. The fgets function stops reading characters when end-of-file is reached, or when a newline character is read, or when n-1 characters have been read, whichever comes first.

Parameters:

- **buf** - storage location for data
- **n** - maximum number of characters to read
- **fp** - pointer to file

Returns: returns **buf** if successful, otherwise 0.

Platform: DOS and Windows

---

float find_DC (float **target_x**, float **target_y**)

Purpose: finds denominator constant for distance calculation. This is the denominator constant (DC) of a line that stretches between the origin and a point in the visual field of coordinates (**target_x**, **target_y**). Using DC removes the need for costly computations of sin or cosine in trigonometric calculations, thus speeding up functions such as in_corridor() and distance_to_line() that require DC as input.

Parameters: targetx, **targety** - point in the visual field

Returns: 1.0 / sqrt(1.0 + (**target_y** / **target_x**)*(**target_y** / **target_x**)), i.e., 1/(1+slope^2)

See also: distance_to_line(), find_slope(), in_corridor()

Platform: DOS and Windows

---

float find_slope (float **target_x**, float **target_y**)

Purpose: find slope of a line formed from the point (target_x, target_y) and the origin (0,0).

Parameters: targetx, **targety** - point in the visual field

Returns: the slope (**target_y** / **target_x**) of a line stretched between the origin and point (**target_x**, **target_y**).

See also: distance_to_line(), find_DC(), in_corridor()

Platform: DOS and Windows

---

float floor (float **value**)

Purpose: computes the largest integer not greater than value.

Parameter: value - floating-point value to be manipulated

Returns: a floating-point value representing the largest integer that is less than or equal to value.

Platform: DOS and Windows

---

float fmax (float **value_1**, float **value_2**)

Returns: the maximum of two floating point **values**

See also: fmin(), max(), min()

Platform: DOS and Windows

---

float fmin (float **value_1**, float **value_2**)

Returns: the minimum of two floating point **values**

See also: fmax(), max(), min()

Platform: DOS and Windows

---

plong fopen(pchar **filename**, pchar **mode**)

Purpose: opens the file specified by filename.

Parameters:

- **filename** - filename
- **mode** - type of access permitted
  - "r" - open file for reading
  - "w" - open file for writing
  - "a" - append
  - "t" - text
  - "b" - binary

Returns: a pointer to the open file. A null pointer value indicates an error.

Platform: DOS and Windows

---

int foreback_wins (int **fore_test_screen**, int **fore_speed**, int **fore_direction**, int **back_test_screen**, int **back_speed**, int **back_direction**)

Purpose: to execute simultaneous motion of two test_screens. This function has largely been replaced by Gscroll().

Parameters:

- **fore_test_screen** (TEST0 through 9)
- **fore_speed** (of movement, in units of 1/100 of deg per second)
- **fore_direction** (of relative motion, in 1/100 of deg of angle)
- **back_test_screen** INT background workstation #
- **back_speed** INT speed of movement for second workstation.
- **back_direction** INT direction of APPARANT motion for second workstation.

Returns: time remaining (in msec). Returns 0 when done, and turns off both test_screens, waiting until they are actually off before returning.

NOTE: init_foreback() must be called prior to this function.

See also: init_foreback()

Platform: DOS and Windows

---

int fprintf (plong **fp**, pchar **format**)

Purpose: writes output to the file pointed to by **fp** under control of the argument **format**. The format string has the same syntax and use as in printf().

Parameters:

- **fp** - pointer to file
- **format** - format-control string

Returns: the number of bytes written. Otherwise, returns a negative value if an output error occurs.

Platform: DOS and Windows

---

int fputc (int **c**, plong **fp**)

Purpose: writes the character specified by c to the output stream designated by **fp**.

Parameters:

- **c** - character to be written
- **fp** - pointer to file

Returns: the character written; or, if a write error occurs, returns EOF.

Platform: DOS and Windows

---

int fputs (pchar **buf**, plong **fp**)

Purpose: writes the character string pointed to by **buf** to the output stream designated by **fp**.

Parameters:

- **buf** - character string to be written
- **fp** - pointer to file

Returns: returns EOF if an error occurs otherwise, it returns a non-negative value.

Platform: DOS and Windows

---

int fread (pchar **buffer**, int **size**, int **count**, plong **fp**)

Purpose: reads **count** elements of **size** bytes each from the file specified by **fp** into the buffer specified by **buffer**.

Parameters:

- **buffer** - storage location for data
- **size** - item size in bytes
- **count**- maximum number of items to be read
- **fp** - pointer to file

Returns: the number of complete elements actually read, which may be less than count if an error occurs or if the end of the file is encountered before reaching count.

Platform: DOS and Windows

---

void free (pchar **memory_block**)

Purpose: frees (un-allocates) the currently allocated memory pointed to by **memory_block**. Be careful to be sure that the pointer you are freeing is the correct pointer. A misspelling with this function can crash the system or worse.

Parameter: memory_block - previously allocated memory block to be freed

Returns: nothing

See also: calloc(), malloc(), realloc()

Platform: DOS and Windows

---

float freespace (**void**)

Purpose: calculates the amount of freespace in kilobytes, on the drive containing the Cortex data file.

Parameters: none

Returns: the amount of free space in kilobytes.

Platform: DOS and Windows

---

plong freopen (pchar **filename**, pchar **mode**, plong **fp**)

Purpose: closes the file currently associated with **fp**. Then, it opens the file named **filename** and associates this new file with **fp**.

Parameters:

- **filename** - name of new file
- **mode** - type of access permitted (see fopen())
- **fp** - pointer to file

Returns: a pointer to the newly opened file. If an error occurs, the original file is closed and the function returns a NULL pointer value.

Platform: DOS and Windows

---

int fscanf (plong **fp**, pchar **msg**)

Purpose: scans input from the file designated by **fp** under control of the argument **format**.

Parameters:

- **fp** - pointer to file
- **format** - format-control string

Returns: the number of input arguments for which values were successfully scanned and stored. Otherwise, returns EOF when the scanning is terminated by reaching the end of the input stream.

Platform: DOS and Windows

---

int fseek (plong **fp**, long **offset**, int **where**)

Purpose: changes the read/write position of the file specified by **fp**. The argument **offset** is the position to seek to relative to one of three positions specified by the argument **where**.

Parameters:

- **fp** - pointer to the file
- **offset** - position to seek to
- **where** - the offset will be relative to this location. Allowable values:
  - **SEEK_SET** - The new file position is computed relative to the start of the file. The value of offset must not be negative.
  - **SEEK_CUR** - The new file position is computed relative to the current file position. The value of offset may be positive, negative or zero.
  - **SEEK_END** - The new file position is computed relative to the end of the file.

Returns: 0 if successful, otherwise non-zero.

Platform: DOS and Windows

---

long ftell (plong **fp**)

Purpose: returns the current read/write position of the file specified by **fp**.

Parameters: fp - pointer to file

Returns: returns the current read/write position of the file if successful; otherwise, it returns -1 on error.

Platform: DOS and Windows

---

int fwrite (pchar **buffer**, int **size**, int **count**, plong **fp**)

Purpose: writes **count** elements of **size** bytes each to the file specified by **fp**.

Parameters:

- **buffer** - storage location for data
- **size** - item size in bytes
- **count** - maximum number of items to be written
- **fp** - pointer to file

Returns: the number of complete elements successfully written. This value will be less than the requested number of elements only if a write error occurs.

Platform: DOS and Windows

---

pchar GactivateCLT (int **num_entries**, int **CLTsource**, int **src_start**, int **dst_start**)

Purpose: Delayed (Gflush()able) version for changing a set of colors. Transfers the colors from a temporary CLT to the active CLT.

Parameters:

- **num entries** - the number of CLT entries to load
- **CLTsource** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.
- **src_start** - the starting index in the source (temporary) CLT
- **dst_start** - the starting index in the destination (active) CLT

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

Platform: DOS and Windows

---

pchar Gadd (int **test_screen/index**, int **operation**, int **repetitions**, float **arg1**, float **arg2**, int **arg3**)

Purpose: General interface to graphics kernel. All of the other kernel functions are converted into Gadd() calls. Basically, Gadd() is the mother of all graphics kernel functions. Must be followed by Gflush().

Parameters:

- **test_screen/index** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h"; or color index)
- **operation** (G_xy_MOVE_ABS, ..., see table below; #include "css_inc.h")
- **repetitions** (must be =1 units of Gflush() calls: allows "automatic" repeats)

- **arg1** (depends on **operation**, see below)
- **arg2** (depends on **operation**, see below)
- **arg3** (depends on **operation**, see below)

Arguments for **Gadd()**:

General Graphical Operations

Operation	arg1	arg2	notes
G_COLOR_ABS	red*256 +blue	green	Changes requested **index** value to r,g,b
G_COLOR_LUT	0	0	Changes the current LUT to **index**
G_COLOR_REL	red*256 +blue	green	Adds r,g,b to the requested **index** in the LUT
G_MOVIE_run_FORE	pause_on_each+1	pause_on_each	Plays a movie forwards
G_MOVIE_run_REV	pause_on_each+1	pause_on_each	Plays a movie backwards
G_MOVIE_step	frames	bounds	Adds **frames** to the current frame of the movie
G_PRIORITY	priority	0	Changes the priority of **test_screen**
G_VISIBLE	on or off (1 or 0)	0	Turns **test_screen** on or off

Operations using degrees per visual angle coordinates

Operation	arg1	arg2	notes
G_xy_MOVE_ABS	x	y	Moves **test_screen** to x,y offset from center of the screen
G_xy_MOVE_ABSref	x	y	Moves **test_screen** to x,y offset from the reference point
G_xy_MOVE_ABSorig	x	y	Moves **test_screen** to x,y offset from the lower left corner of the screen
G_xy_MOVE_REL	x	y	Moves **test_screen** to x,y offset from current position
G_xy_PAN_ABS	x	y	Pans **test_screen** within its window to absolute offset
G_xy_PAN_REL	x	y	Pans **test_screen** within its window relative to current position
G_xy_WINSIZE_ABS	x	y	Changes the size of **test_screen**'s window to x,y
G_xy_WINSIZE_REL	x	y	Adds x,y to the dimensions of **test_screen**'s window

Operations using pixel coordinates

Operation	arg1	arg2	notes
G_pix_MOVE_ABS	x	y	Moves **test_screen** to x,y offset from center of the screen
G_pix_MOVE_ABSref	x	y	Moves **test_screen** to x,y offset from the reference point

G_pix_MOVE_ABSorig	x	y	Moves **test_screen** to x,y offset from the lower left corner of the screen
G_pix_MOVE_REL	x	y	Moves **test_screen** to x,y offset from current position
G_pix_PAN_ABS	x	y	Pans **test_screen** within its window to absolute offset
G_pix_PAN_REL	x	y	Pans **test_screen** within its window relative to current position
G_pix_WINSIZE_ABS	x	y	Changes the size of **test_screen**'s window to x,y
G_pix_WINSIZE_REL	x	y	Adds x,y to the dimensions of **test_screen**'s window

Operations using radial coordinates

Operation	arg1	arg2	notes
G_rt_MOVE_ABS	radius	theta	Moves **test_screen** to r,t offset from center of the screen
G_rt_MOVE_ABSref	radius	theta	Moves **test_screen** to r,t offset from the reference point
G_rt_MOVE_ABSorig	radius	theta	Moves **test_screen** to r,t offset from the lower left corner of the screen
G_rt_MOVE_REL	radius	theta	Moves **test_screen** to r,t offset from current position
G_rt_PAN_ABS	radius	theta	Pans **test_screen** within its window to absolute offset
G_rt_PAN_REL	radius	theta	Pans **test_screen** within its window relative to current position
G_rt_WINSIZE_ABS	radius	theta	Changes the size of **test_screen**'s window to r,t
G_rt_WINSIZE_REL	radius	theta	Adds r,t to the dimensions of **test_screen**'s window

Returns: pointer to the thread allocated by Gadd()

See Also: Gadd_with_wait(), Gcheck(), Gdel(), Gflush(), Gpurge()

Platform: DOS and Windows

---

pchar Gadd_with_wait (int **test_screen/index**, int **operation**, int **repetitions**, float **arg1**, float **arg2**, int **wait_frames**)

Purpose: General interface to graphics kernel. This is the same as Gadd(), except that it will wait a specified number of Gflush() calls before executing. Thus, if the user calls Gflush() every screen refresh (to enact real-time animation or movies, for instance), the Gadd_with_wait() operation will be performed in **wait_frames** screen refreshes. This call is helpful when the user would like to start and end a graphical operation at a variable random times and does not wish to keep track of how much time has passed between the start and end of the operation. Must be followed by Gflush().

Parameters:

- **test_screen/index** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h"; or color index)
- **operation** (G_xy_MOVE_ABS, ..., see table below; #include "css_inc.h")
- **repetitions** (must be =1 units of Gflush calls: allows "automatic" repeats)
- **arg1** (depends on message, see Gadd())
- **arg2** (depends on message, see Gadd())
- **arg3** (depends on message, see Gadd())
- **wait_frames** (number of Gflush() calls until function will be executed)

Returns: pointer to the thread allocated by Gadd_with_wait()

See also: Gadd(), Gcheck(), Gdel(), Gflush(), Gpurge()

Platform: DOS and Windows

---

long Gcheck (pchar **active_thread**)

Purpose: Returns the time remaining on a Gadd() thread in milliseconds.

Parameters:

- **active_thread** - the value returned by a previous Gadd()

Returns: the time remaining for a graphical operation to run (in milliseconds). The value **active_thread** is the value returned by many of the New Graphical Routines (the ones that are not instantaneous in duration) and it allows Gcheck() to trace the operation's remaining time on the queue.

See also: Gadd()

Platform: DOS and Windows

---

void GcolorABS (int **index**, int **red**, int **green**, int **blue)**;

sets the color of one **index** within the current lookup table. To find out which index of the LUT contains the color information for a certain item, call ITEM_POSlut_index(). Must be followed by Gflush().

- **index** (index within color LUT)
- **red** (0-255)
- **green** (0-255)
- **blue** (0-255)

See also: GcolorLUT(), GcolorREL(), ITEM_POSlut_index(), load_CLT(), set_CLT_load_index(), set_colorABS(), set_colorREL()

Platform: DOS and Windows

---

void GcolorLUT (int **index**)

Purpose: Delayed (Gflush()able) version for changing a set of colors. Transfers the entire palette of colors (i.e., 256 colors) from a temporary CLT to the active CLT.

Parameters:

- **CLTsource** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

Platform: DOS and Windows

---

pchar GcolorLUTsubset (int **num_entries**, int **CLTsource**, int **src_start**, int **dst_start**)

Purpose: Delayed (Gflush()able) version for changing a set of colors. Transfers the colors from a temporary CLT to the active CLT.

Parameters:

- **num entries** - the number of CLT entries to load
- **CLTsource** - number of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.
- **src_start** - the starting index in the source (temporary) CLT
- **dst_start** - the starting index in the destination (active) CLT

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

Platform: DOS and Windows

---

void GcolorREL (int **index**, int **red**, int **green**, int **blue**)

Purpose: resets the color of an item one color lookup table index at a time by changing a single value within a color lookup table. Adds the values added_red, added_green, and added_blue to the current values for that entry in the color lookup table.

Parameters:

- **index** - index within color LUT [use ITEM_POSlut_index()]
- **added_red** - offset from the current red value, can be positive or negative
- **added_green** - offset from the current green value, can be positive or negative
- **added_blue** - offset from the current blue value, can be positive or negative

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: GcolorABS(), GcolorLUT(), ITEM_POSlut_index(), load_CLT(), set_CLT_load_index(), set_colorABS(), set_colorREL()

Platform: DOS and Windows

---

void Gdel (pchar **active_thread**)

Purpose: removes a graphical operation that has been added to the stack by one of the New Graphics Routines. Only needed if the graphical operation has a duration that is non-instantaneous.

Parameters:

- **active_thread** (pointer to the graphical operation to be deleted; returned by Gadd())

Returns: the number of threads deleted. Note that this can be dangerous if it isn't currently part of the structure, so descend that structure instead of just using next/previous pointers

See Also: Gadd(), Gpurge()

Platform: DOS and Windows

---

pchar GDPget_CLT (int **num_colors**, int **bstart_index**, pchar **colors**)

Purpose: gets the values of the active color lookup table (CLT) which is in use by graphics board

Parameters:

- **num_colors** - the number of colors to be retrieved
- **start_index** - the starting index in the active CLT
- **colors** - the color data values. Internally, this parameter is a pointer to an array of BYTE_RGB structures which hold the red, green and blue values.

typedef struct { **Platform:** DOS and Windows

---

void GDPset_CLT (int **num_colors**, int **start_index**, pchar **colors**)

Purpose: sets the values of the active color lookup table (CLT) which is in use by graphics board

Parameters:

- **num_colors** - the number of colors to be set
- **start_index** - the starting index in the active CLT
- **colors** - the color data values. Internally, this parameter is a pointer to an array of BYTE_RGB structures which hold the red, green and blue values.

typedef struct {
Platform: DOS and Windows

---

int GetAKey ()

Purpose: waits for and returns a key press.

Parameters: none

Returns: casting as (char) will provide the key that was pressed, and the high byte contains special attributes (such as SHIFT or CTRL). Or, one can compare directly with the #defines in css_inc.h (be sure to #include "css_inc.h"). Internally, this function is the same as KeyGet().

See Also: KeyGet()

Platform: DOS and Windows

---

int getch()

Purpose: Get a character from the console without echo

Parameters: none

See also: gets()

Platform: DOS only

---

void getCndsFileName(pchar **filename**)

Purpose: This function will return the name of the Cortex conditions file through a parameter. The timing file must allocate space for the file name.

Parameters: pointer to a string that will hold the filename

Returns: nothing.

Platform: Windows only

---

pchar getcwd (pchar **current_directory**, int **max_path_length**)

Purpose: gets the name of the currently set working directory.

Parameters:

- allocates memory of at least size **max_path_length**
- returns the pointer to **current_directory**.

Returns: a pointer to the newly allocated **current_directory**, which is filled with the requested path string

See also: chdir(), mkdir(), rename(), rmdir()

Platform: DOS and Windows

---

void**Platform:** DOS and Windows

---

pchar getenv (pchar **varname**)

Purpose: searches the list of environment variables for an entry corresponding to varname

Parameter: varname - Environment variable name

Returns: pointer to that entry in the environment table

Platform: DOS and Windows

---

void getExternsFileName(pchar **filename**)

Purpose: This function will return the name of the Cortex external variables file through a parameter. The timing file must allocate space for the file name.

Parameters: pointer to a string that will hold the filename

Returns: nothing.

Platform: Windows only

---

void getItemsFileName(pchar **filename**)

Purpose: This function will return the name of the Cortex items file through a parameter. The timing file must allocate space for the file name.

Parameters: pointer to a string that will hold the filename

Returns: nothing.

Platform: Windows only

---

pchar gets (pchar **string**)

Purpose: gets a line from the keyboard and stores it in **string**. The characters are echoed to the screen as they are typed in (text mode only). The gets() function replaces the newline character '\n' with a NULL character '\0'. The user is responsible for allocating enough memory for the incoming string....otherwise there may be a catastrophe (such as an array overwrite or system crash).

Parameter: buffer Storage location for input string

Returns: the **string**, if succesful

See Also: getch(), SCREENmode()

Platform: DOS only

---

void getTimeDateString(pchar **timStr**)

Purpose: This function will return the current time and date through a string parameter. The timing file must allocate space for the time/date. The size of the string will be 26 chars. NOTE: This function was added to DOS for version 5.9.6.

Parameters: pointer to a string that will hold the time/date string

Returns: nothing.

Platform: DOS and Windows

---

void getTimingFileName(pchar **filename**)

Purpose: This function will return the name of the Cortex timing file through a parameter. The timing file must allocate space for the file name.

Parameters: pointer to a string that will hold the filename

Returns: nothing.

Platform: Windows only

---

int get_a_input (int **val**)

Purpose: read a byte from Port A of the PIO24 board.

Parameters:

- **val** - used as a mask for the input byte. If **val** < 8, it is multiplied by 2, and used as a mask. Otherwise, if val >=8, the input is not read and -1 is returned. (I have no idea why this masking is done.)

Returns: input from Port A of PIO24, masked with 2***val** if **val** < 8. Otherwise, returns -1.

Platform: DOS and Windows

---

int get_bar_state()

Purpose: get the current state of the subject's response bar.

Parameters: none

Returns: bar state. If this is a bar up/bar down paradigm, then bar state returns 0 for bar up, 1 for bar down. If this is a bar left/right paradigm, it returns 1 for bar_centered (neither left nor right) 2 for bar_left, 3 for bar_right, and 4 for bar extra. This function assumes that the bar inputs are connected as specified in CORTEX manual.

See Also: get_block_num()

Platform: DOS and Windows

---

int get_block_num ()

Purpose: Returns the number of the given block

Parameters: none

Returns: the number of the current block being tested. The first block is 0 not 1.

See also: BLOCKget_block_num(), BLOCKget_cond_num(), BLOCKset_next(), get_cond_num()

Platform: DOS and Windows

---

int get_block_pct_correct (int **block_id**)

Purpose: Returns the percent correct (0-10000) of a given block. This function has been, for the most part, replaced by BLOCKget_pct(). The first block is 0 not 1.

Parameter:

- **block_id** - the block number (must be a number from 0 to the maximum number of blocks)

Returns: The average percentage of correct trials (multiplied by 100) of all the conditions in block number block_id.

See also: BLOCKclear_stats(), BLOCKget_pct_correct(), get_cond_pct_correct()

Platform: DOS and Windows

---

int get_CODEbuf (int **index**)

Purpose: get the current CODEbuf value at the given index.

Parameters: index

Returns: the current CODEbuf value at the given index

Platform: Windows only

---

int get_CODE_ISImax ()

Purpose: get the current CODE_ISImax value.

Parameters: none

Returns: the current CODE_ISImax value

Platform: Windows only

---

int get_CODE_ISIoverflow ()

Purpose: get the current CODE_ISIoverflow value.

Parameters: none

Returns: the current CODE_ISIoverflow value

Platform: Windows only

---

int get_CODE_ISIsize ()

Purpose: get the current CODE_ISIsize value.

Parameters: none

Returns: the current CODE_ISIsize value

Platform: Windows only

---

int get_cond_num ()

Purpose: Returns the number of the given condition

Parameters: none

Returns: the current condition being tested. The first condition is 0 not 1.

See also: BLOCKget_cond_num(), BLOCKget_block_num(),BLOCKset_next(), get_block_num()

Platform: DOS and Windows

---

int get_cond_pct_correct (int **cond_id**)

Purpose: Returns the percent correct (0-10000) of a given condition

Parameters:

- **cond_id** - the condition number (must be a number from 0 to the maximum number of conditions)

Returns: the percentage of correct trials (multiplied by 100) in condition **cond_id**. The first condition is 0 not 1.

See also: BLOCKclear_stats(), BLOCKget_pct_correct(), get_block_pct_correct()

Platform: DOS and Windows

---

int get_digital_input (int **val**)

Purpose: read a byte from Port C of the PIO24 board.

Parameters:

- **val** - used as a mask for the input byte. If val < 8, it is multiplied by 2, and used as a mask. Otherwise, if val >=8, the input is not read and -1 is returned. (I have no idea why this masking is done.)

Returns: input from Port C of PIO24, masked with 2***val** if **val** < 8. Otherwise, returns -1.

Platform: DOS and Windows

---

int get_EOGbuf (**int index**)

Purpose: get the current EOGbuf value at the given index.

Parameters: index

Returns: the current EOGbuf value at the given index

Platform: Windows only

---

int get_EOGdynamic_fixwin_size ()

Purpose: get the current params.dynamic_eyewin_size value.

Parameters: none

Returns: the current params.dynamic_eyewin_size value

Platform: Windows only

---

float get_EOGfixwin_size_x ()

Purpose: get the current params.window_x value.

Parameters: none

Returns: the current params.window_x value

Platform: Windows only

---

float get_EOGfixwin_size_y ()

Purpose: get the current params.window_y value.

Parameters: none

Returns: the current params.window_y value

Platform: Windows only

---

int get_EOGgain ()

purpose: get the current params.eog_gain value.

Parameters: none

Returns: the current params.eog_gain value

Platform: Windows only

---

int get_EOGmax ()

Purpose: get the current EOGmax value.

Parameters: none

Returns: the current EOGmax value

Platform: Windows only

---

int get_EOGnew_x ()

Purpose: get the current EOGnew_x value.

Parameters: none

Returns: the current EOGnew_x value

Platform: Windows only

---

int get_EOGnew_y ()

Purpose: get the current EOGnew_y value.

Parameters: none

Returns: the current EOGnew_y value

Platform: Windows only

---

int get_EOGoffset_x ()

Purpose: get the current params.eog_xoffset value.

Parameters: none

Returns: the current params.eog_xoffset value

Platform: Windows only

---

int get_EOGoffset_y ()

Purpose: get the current params.eog_yoffset value.

Parameters: none

Returns: the current params.eog_yoffset value

Platform: Windows only

---

int get_EOGoverflow ()

Purpose: get the current EOGoverflow value.

Parameters: none

Returns: the current EOGoverflow value

Platform: Windows only

---

float get_EOGsaccade ()

Purpose: get the current params.mc value.

Parameters: none

Returns: the current params.mc value

Platform: Windows only

---

int get_EOGsize ()

Purpose: get the current EOGsize value.

Parameters: none

Returns: the current EOGsize value

Platform: Windows only

---

int get_EPPbuf (int **index**)

Purpose: get the current EPPbuf value at the given **index**.

Returns: the current EPPbuf value at the given **index**

Platform: Windows only

---

int get_EPPmax ()

Purpose: get the current EPPmax value.

Parameters: none

Returns: the current EPPmax value

Platform: Windows only

---

int get_EPPnew_x ()

Purpose: get the current EPPnew_x value.

Parameters: none

Returns: the current EPPnew_x value

Platform: Windows only

---

int get_EPPnew_y ()

Purpose: get the current EPPnew_y value.

Parameters: none

Returns: the current EPPnew_y value

Platform: Windows only

---

int get_EPPoverflow ()

Purpose: get the current EPPoverflow value.

Parameters: none

Returns: the current EPPoverflow value

Platform: Windows only

---

int get_EPPsize ()

Purpose: get the current EPPsize value.

Parameters: none

Returns: the current EPPsize value

Platform: Windows only

---

int get_eye_storage_rate ()

Purpose: gets the eye storage rate that was set in the Run:

Parameters: General:EOG_storage_rate menu. It is also the value that is stored in the header of the Cortex output data file.

Parameters: none

Returns: the eye data storage rate.

Platform: DOS and Windows

---

int get_fixation_state()

Purpose: find out whether the subject's eye is within the fixation window.

Parameters: none

Returns: fixation state (1 if subject is fixated, 0 if not).

See Also: ITEM_POSeye_ishere(), ITEM_POSeye_iswithin()

Platform: DOS and Windows

---

long get_ISIbuf (int **index**)

Purpose: get the current ISIbuf value at the given index.

Parameters: index

Returns: the current ISIbuf value at the given index

Platform: Windows only

---

int get_keep_current_conds ()

Purpose: get the current params.keep_current_conds value.

Parameters: none

Returns: the current params.keep_current_conds value

Platform: Windows only

---

int get_kHz_resolution ()

Purpose: gets the kHz resolution value that is stored in the header of the Cortex output data file.

Parameters: none

Returns: the kHz resolution value.

Platform: DOS and Windows

---

int get_ms_reward_duration ()

Purpose: get the current params.ms_reward_duration value.

Parameters: none

Returns: the current params.ms_reward_duration value

Platform: Windows only

---

float get_param (pchar **name**, float **default_value**, pchar **parfile**)

Purpose: finds the default value of a parameter in a file **parfile** which contains semi-colons at the ends of the lines, and contains an equal sign after the **name** but before the **default_value**. (I am not sure of the real use of this function.)

Parameters:

- **name** - name of the parameter that you are looking for
- **default_value** - default value for the parameter (not used)
- **parfile** - name of the file to search

Returns: the default value of the parameter

Platform: DOS and Windows

---

int get_repeat_num ()

Purpose: gets the repeat number, as it was stored in the header of the Cortex output data file.

Parameters: none

Returns: the repeat number.

Platform: DOS and Windows

---

int get_saccade_state ()

Purpose: Find out whether the subject's eye is within the fixation window. If set_saccade_tolerance() was called first, saccades will be

checked for at the desired rate. Otherwise, get_saccade_state() is equal to get_fixation_state(). When there is a saccade error, can't distinquish it from a get_fixation_state error, so a stricter get_fixation_state.

Parameters: none

Returns: 0 if not fixated, 1 if fixated and not saccading, 2 if fixated but saccading too much

See Also: get_fixation_state()

Platform: DOS and Windows

---

long get_TIMER100us_counter ()

Purpose: get the current TIMER100us_counter value.

Parameters: none

Returns: the current TIMER100us_counter value

Platform: Windows only

---

long get_TIMERms_counter ()

Purpose: get the current TIMERms_counter value.

Parameters: none

Returns: the current TIMERms_counter value

Platform: Windows only

---

int get_trial_num()

Purpose: Returns the current trial_number

Parameters: none

Returns: current trial number

Platform: DOS and Windows

---

int get_trial_type ()

Purpose: gets the trial type that was set in the TRIAL_TYPE column of the conditions file. It is also the value that is stored in the header of the Cortex output data file as the expected_response.

Parameters: none

Returns: the trial type.

Platform: DOS and Windows

---

int Gflush (int **synchronize**)

Purpose: After queueing a number of graphics commands, you need to flush them to ensure that they are acted upon. You can also specify whether or not to synchronize on them (i.e. wait until the graphics card is finished the requested operations before continuing with the calling program.

Parameter: synchronize (1 = synchronize, 0 = don't synchronize).

Returns: number of threads left in queue to be processed.

Note: If you are using the DirectX version of the receive program, the synchronize parameter of the Gflush() function has no effect. The program will always behave as if Gflush(1) was called. The reason for this behavior is that in a DirectX application, the vertical refreshing behavior must occur at initialization, and cannot be changed dynamically. Since it was assumed that most users would want to synchronize with the vertical refresh, this was programmed to be the default behavior.

See Also: GmoveABS()

Platform: DOS and Windows

---

void GmoveABS (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: Moves the center of a **test_screen** to an absolute position. (Moves the window's center relative to (0,0).) Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gdel(), Gflush(), GmoveREL(), Gpurge()

Platform: DOS and Windows

---

pchar GmoveABSorig (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: Moves the center of a **test_screen** to another position. (Moves the window's center relative to it's original center.) Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")

- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See Also: Gdel(), Gflush(), GmoveREL(), Gpurge()

Platform: DOS and Windows

---

pchar GmoveABSref (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: Moves the center of a **test_screen** to another position. (Moves the window's center relative to the reference point as defined in the Cortex Item:Reference menu option.) Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See Also: Gdel(), Gflush(), GmoveREL(), Gpurge()

Platform: DOS and Windows

---

void GmoveREL (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: Move the center of a test_screen to a location relative to its current location. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gdel(), Gflush(), GmoveABS(), Gpurge()

Platform: DOS and Windows

---

int Gmove_fixwin (int **movetype**, float **arg1**, float **arg2**)

Purpose: Move the fixation window to a new location. This function draws the new fixation window on the user's screen and updates the eye boundary parameters.

Parameters:

- **movetype** - specify how the fixwin will be moved. Can be any one of the following values (must #include "css_inc.h"):
  - G_xy_MOVE_ABS, G_xy_MOVE_ABSref, G_xy_MOVE_ABSorig, G_xy_MOVE_REL, G_pix_MOVE_ABS, G_pix_MOVE_ABSref, G_pix_MOVE_ABSorig, G_pix_MOVE_REL, G_rt_MOVE_ABS, G_rt_MOVE_ABSref, G_rt_MOVE_ABSorig, G_rt_MOVE_REL
- **arg1** - depends on the movetype parameter; refer to Gadd() for the necessary parameters.
- **arg2** - depends on the movetype parameter; refer to Gadd() for the necessary parameters.

Returns: 1 if successful.

Platform: DOS and Windows

---

pchar Gmovie (int **test_screen**, long **duration**, int **pause_each_frame**, int **starting_frame**, int **direction**, int **auto_on_off**)

Purpose: The Gmovie() function tells CORTEX to display a movie in the specified test_screen. CORTEX will not actually display a frame of the movie until the Gflush() function is called. In order for each new frame of the movies to be drawn, Gflush() must be called. Thus, if your monitor is updated at 60Hz, you should call Gflush() at least once every 1/60 seconds. The easiest way to handle this is to put the Gflush() function within a while-loop immediately after the Gmovie() function is called. If the fifth argument of the Gmovie() function is set to 1, the movies will automatically turn off at the correct time as set by the duration parameter.

Note that Gflush() will return the value 1 if Gmovie() or any other graphical function is pending. Gflush() will return the value 0 if there are no pending graphical functions pending. Therefore, one easy way to guarantee that you will display the movie until the duration has elapsed is with code like the following example:

Platform: DOS and Windows

---

pchar Gmovie_one_time (int **test_screen**, long **duration**, int **pause_each_frame**, int **starting_frame**, int **direction**, int **auto_on_off**)

Purpose: Runs a movie one time through one complete cycle of all of the frames of the movie, and then stops. If the **starting_frame** is a value other than the first frame of the movie (i.e., frame number 0), then the movie will be displayed starting at the specified frame, and will play through all of the frames of the movie, wrapping around until it reaches the **starting_frame** - 1. If there is only one frame in the movie, it just calculates the number of frames based upon the time (like Gmovie() does). Gmovie_one_time() must be

followed by Gflush(). Refer to the Gflush() information in the description of Gmovie().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **duration** (milliseconds) - only used if total number of frames is zero or negative
- **pause_each_frame** (number of frames to wait between updates)
- **starting_frame** (relative to current one: ...-1 = last, 0 = current, 1 = next..., enter a large negative or positive number to get to FIRST or LAST frame if the desired offset is unknown)
- **direction** (1 = run forwards, 0 = run backwards)
- **auto_on_off** (if != 0 then automatically turn on and off at appropriate time)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gmovie(), Gmovie_step(), init_movie(), run_movie()

Platform: DOS and Windows

---

void Gmovie_step (int **test_screen**, int **next_frame**, int **bounds**)

Purpose: Steps a movie to a new position. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **next_frame** (relative to current one: ...-1 = last, 0 = current, 1 = next..., enter a large negative or positive number to get to FIRST or LAST frame if the desired offset is unknown)
- **bounds** (one of following:)
  MOVIE_FIRST_FRAME - goto first frame (ignore "next_frame") MOVIE_LAST_FRAME - goto last frame (ignore "next_frame") MOVIE_STEP_WRAP - add "next_frame" to current position. Allow wrap. MOVIE_STEP_BOUNDED - add "next_frame" to current, stopping @ bounds MOVIE_STEP_IF_VALID - only add "next_frame" to current if within bounds

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gmovie(), init_movie(), run_movie()

Platform: DOS and Windows

---

void Gon_off (int **test_screen**, int **visible**)

Purpose: Turn a test_screen on or off. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **visible** (1 = on, 0 = off)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: display_fixspot(), display_sample(), display_test(), Gdel(), Gflush(), Gpurge()

Platform: DOS and Windows

---

pchar Gpan (int **test_screen**, float **speed**, float **direction**, long **duration**, int **auto_on_off**)

Purpose: The Gpan() function tells CORTEX to display and pan a test_screen across its center. CORTEX will not actually display a frame until the Gflush() function is called. In order for each new frame to be drawn, Gflush() must be called. The easiest way to handle this is to put the Gflush() function within a while-loop immediately after the Gpan() function is called. If the fifth argument of the Gpan() function is set to 1, the item will automatically turn off at the correct time as set by the duration parameter.

Note that Gflush() will return the value 1 if Gpan() or any other graphical function is pending. Gflush() will return the value 0 if there are no pending graphical functions pending. Therefore, one easy way to guarantee that you will display the pan until the duration has elapsed is with code like the following example:

```
Gpan(TEST0, 2, 2, 1000, 1);
while (Gflush(1));
```

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed** (degrees of visual angle per second)
- **direction** (degrees)
- **duration** (milliseconds)
- **auto_on_off** (if != 0 then automatically turn on and off at appropriate time)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gflush(), GpanABS(), GpanREL(), Gpurge(), init_pan(), pan_win(), pan_wkstABS(), pan_wkstREL()

Platform: DOS and Windows

---

void GpanABS (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: Pans the item across absolute coordinates while its window stays still. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gflush(), Gpan(), GpanREL(), Gpurge(), init_pan(), pan_win(), pan_wkstABS(), pan_wkstREL()

Platform: DOS and Windows

---

void GpanREL (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: Pans the item across coordinates relative to its current position while its window stays still. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gflush(), Gpan(), GpanABS(), Gpurge(), init_pan(), pan_win(), pan_wkstABS(), pan_wkstREL()

Platform: DOS and Windows

---

void Gpriority (int **test_screen**, int **priority**)

Purpose: window with highest priority is on top in case of overlap. The default is that the FIXSPOT has the highest priority, followed the PLAY stimulus, followed by TEST0, etc. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **priority** - the maximum priority is 700. The fixspot, since it is highest priority, is also 700. The play screen has a priority of 650. All the test screens have a priority assigned by decrements of 50. In other words, TEST0 has a priority of 600, TEST1 has a priority of 550, TEST2 has a priority of 500,

etc. The higher the number, the higher the priority.

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel()

See also: init_foreback()

Platform: DOS and Windows

---

void Gpurge ()

Purpose: In situations where one needs to abort a trial, and is not certain which commands might remain in queue, Gpurge() will clear the entire buffer. In general, it is a good idea to specifically Gdel() the commands from the queue, but calling Gpurge() at the end of a timing file (before exit()) may be beneficial. This call is basically the same as Gdel(), but it deletes ALL pending graphics operations at once.

Parameters: none

Returns: number of threads deleted.

See also: Gdel()

Platform: DOS and Windows

---

void GRAPHICSclose ()

Purpose: shutdown all the graphics operations and release all of the resources. Should not need to call this function, since Cortex calls this automatically when the trial is over. May not work with the two-computer version of Cortex.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void GRAPHICSdegenerate ()

Purpose: erases the graphics from the screen, and deallocates the storage for them. Should not need to call this function, since Cortex calls this automatically when the trial is over.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void GRAPHICSdraw (int **who**, int **item_id**)

Purpose: Draw an item onto an offscreen surface. All objects/groups are drawn with respect to the reference point except

for the reference point itself, which is drawn relative to the center of the screen (0,0). Should not need to call this function, since Cortex calls this internally when Gon_off() is called. (Use Gon_off() instead!!)

Parameters:

- **who** - TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h"
- **item_id** - the number of the item to be drawn

Returns: nothing

Platform: DOS and Windows

---

void GRAPHICSdraw_background (int **item_id**)

Purpose: Sets the background color to be the color of the given item number, as specified in the items file RGB columns. Should not need to call this function, since Cortex calls this internally when the trial is started.

Parameters: item_id - number of the item to be used

Returns: nothing

Platform: DOS and Windows

---

void GRAPHICSdraw_condition ()

Purpose: Draws the items of the current condition number into an offscreen surface. Should not need to call this function, since Cortex calls this internally when the trial is started.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void GRAPHICSgenerate (int **cond**)

Purpose: Draws the items of the given condition number into an offscreen surface. Can not be called if graphics have already been generated on the subject's screen. Must issue a GRAPHICSdegenerate() call first. Should not need to call this function, since Cortex calls this internally when the trial is started.

Parameters: cond - condition number

Returns: nothing

Platform: DOS and Windows

---

int GRAPHICSopen (int **bits_per_pixel**, float **hpixels_per_dva**, float **vpixels_per_dva**, int **fps**, int **Xdim**, int **Ydim**)

Purpose: initiate graphics capabilities of the given specifications on the subject's monitor. Should not need to call this function, since Cortex calls this internally when it starts up. May not work with the two-computer version of Cortex.

Parameters:

- **bits_per_pixel** - valid values: 1, 2, 4, 8, 16, 24, 32
- **hpixels_per_dva** - number of horizontal pixels per degree of visual angle
- **vpixels_per_dva** - number of vertical pixels per degree of visual angle
- **fps** - number of frames per second
- **Xdim** - resolution of the screen in the X dimension
- **Ydim** - resolution of the screen in the Y dimension

Returns: 1 if successful, otherwise 0.

Platform: DOS and Windows

---

void GRAPHICSread_color (int **color**, int ***r**, int ***g**, int ***b**)

Purpose: get the color value for the specified index in the active color lookup table (CLT).

Parameters:

- **color** - index into the active CLT
- **r** - pointer to the red value
- **g** - pointer to the red value
- **b** - pointer to the red value

Returns: nothing

Platform: DOS and Windows

---

int GRAPHICSread_color_palette(char ***palette_name**)

Purpose: read the given Cortex color lookup table (CLT) file, and load it into the active CLT which is in use by the graphics board.

Parameters: palette_name - filename of the CLT (the file must be in the Cortex CLT format. Refer to the Cortex User's Manual for the file format.)

Returns: 1 if successful

Platform: DOS and Windows

---

void GRAPHICSset_color (int **color**, int **r**, int **g**, int **b**)

Purpose: set the color value for the specified index in the active color lookup table (CLT).

Parameters:

- **color** - index into the active CLT
- **r** - pointer to the red value
- **g** - pointer to the red value
- **b** - pointer to the red value

Returns: nothing

Platform: DOS and Windows

---

pchar Gscroll (int **test_screen**, float **speed**, float **direction**, long **duration**, int **auto_on_off**)

Purpose: The Gscroll() function tells CORTEX to display and scroll a test_screen across its center. CORTEX will not actually display a frame until the Gflush()function is called. In order for each new frame to be drawn, Gflush() must be called. The easiest way to handle this is to put the Gflush() function within a while-loop immediately after the Gscroll() function is called. If the fifth argument of the Gscroll() function is set to 1, the item will automatically turn off at the correct time as set by the duration parameter.

Note that Gflush() will return the value 1 if Gscroll() or any other graphical function is pending. Gflush() will return the value 0 if there are no pending graphical functions pending. Therefore, one easy way to guarantee that you will display the scroll until the duration has elapsed is with code like the following example:

```
Gscroll(TEST0, 2, 2, 1000, 1);
while (Gflush(1));
```

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed** (degrees of visual angle per second)
- **direction** (degrees)
- **duration** (milliseconds)
- **auto_on_off** (if != 0 then automatically turn on and off at appropriate time)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gflush(), Gpurge(), init_scroll(), scroll_win(), scroll_win_with_fix()

Platform: DOS and Windows

---

pchar Gsweep (int **test_screen**, float **speed**, float **direction**, long **duration**, int **auto_on_off**)

Purpose: The Gsweep() function tells CORTEX to display and sweep a test_screen across its center. CORTEX will not actually display a frame until the Gflush() function is called. In order for each new frame to be drawn, Gflush() must be called. The easiest

way to handle this is to put the Gflush() function within a while-loop immediately after the Gsweep() function is called. If the fifth argument of the Gsweep() function is set to 1, the item will automatically turn off at the correct time as set by the duration parameter.

Note that Gflush() will return the value 1 if Gsweep() or any other graphical function is pending. Gflush() will return the value 0 if there are no pending graphical functions pending. Therefore, one easy way to guarantee that you will display the sweep until the duration has elapsed is with code like the following example:

```
Gsweep(TEST0, 2, 2, 1000, 1);
while (Gflush(1));
```

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed** (degrees of visual angle per second)
- **direction** (degrees)
- **duration** (milliseconds)
- **auto_on_off** (if != 0 then automatically turn on and off at appropriate time)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gflush(), Gpurge(), init_sweep(), sweep_win(), sweep_win_with_fix()

Platform: DOS and Windows

---

void Gtransparancy (int **on_off**)

Purpose: turn transparency on or off for all subsequent calls

Parameter: 1 = on, 0 = off

Returns: nothing

Platform: DOS and Windows

---

void GwinSizeABS (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: changes the size of the window surrounding a **test_screen** to an absolute size. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or

Gdel().

See also: Gflush(), Gdel(), Gpurge(), GwinSizeREL()

Platform: DOS and Windows

---

void GwinSizeREL (int **test_screen**, float **horizontal**, float **vertical**)

Purpose: changes the size of the window surrounding **test_screen** to an size relative to its current size. Must be followed by Gflush().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **horizontal** (degrees of visual angle)
- **vertical** (degrees of visual angle)

Returns: pointer to thread added. Can be passed to Gcheck() or Gdel().

See also: Gflush(), Gdel(), Gpurge(), GwinSizeABS()

Platform: DOS and Windows

---

void histogram_Ctik (int **color**)

Purpose: draws a colored tik mark on the current bin of the histogram(s). Typically for indicating where events have occurred in the trial. This is the newer version of histogram_tik(), which has been left in CORTEX for backwards compatibility reasons.

Parameters: color (desired color of tik mark -- #include "css_inc.h")

Returns: nothing.

See also: histogram_tik()

Platform: DOS and Windows

---

void histogram_tik ()

Purpose: draws a tik mark on the current bin of the histogram(s). Typically for indicating where events have occurred in the trial. This has been replaced by histogram_Ctik() and has been left in for backwards compatibility. Paramters: none.

Returns: nothing.

See also: histogram_Ctik()

Platform: DOS and Windows

---

int init_foreback (int **fore_test_screen**, int **back_test_screen**, int **milliseconds**)

Purpose: simultaneously move the images inside of two windows (foreground/background motion).

Parameters:

- **fore_test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **back_test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- duration of foreback movement, in **milliseconds**

Returns: number of milliseconds remaining. If successful, turns on both stimuli, and waits for the stimuli to appear before returning.

See also: init_movie()

Platform: DOS and Windows

---

int init_movie (int **test_screen**, int **duration**, int **rate**, int **first_frame**)

Purpose: to start a movie at frame # **first_frame** and run it at desired **speed** for specified **duration**.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **duration** of movie in milliseconds
- **rate** of movie (if frames_per_second [fps] is set at 60: 1 = 60 fps, 2 = 30 fps, 3 = 20 fps)
- **first_frame** (starting frame) (0 = first, 1 = last, 2 = current, 3 = next 4 = previous)

Returns: 0 if failed to initialize movie, else number of milliseconds left (a multiple of the refresh rate).

See also: Gmovie(), run_movie()

Platform: DOS and Windows

---

int init_pan (int **test_screen**, int **duration**)

Purpose: pans a **test_screen** within a stationary window.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **duration** of panning, in milliseconds

Returns: 0 if failed to initialize panning, else number of milliseconds left (a multiple of the refresh rate).

See also: Gpan(), GpanABS(), GpanREL(), Gpurge(), pan_win(), pan_wkstABS(), pan_wkstREL()

Platform: DOS and Windows

---

int init_scroll (int **test_screen**, int **duration**)

Purpose: move a test_screen around the screen

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **duration** of scrolling, in milliseconds

Returns: 0 if failed to initialize scrolling, else number of milliseconds left (a multiple of the refresh rate)

See also: Gscroll(), scroll_win(), scroll_win_with_fix()

Platform: DOS and Windows

---

int init_sweep (int **test_screen**, int **speed**, int **direction**, int **duration**)

Purpose: move a **test_screen** across the screen so that it passes through the center of the **test_screen's** current location.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed** (in 1/100 of deg per sec)
- **direction** (in 1/100 of deg of angle)
- **duration** of sweeping, in milliseconds

Returns: 0 if failed to initialize sweeping, else number of milliseconds left (a multiple of the refresh rate)

See Also: Gsweep(), init_sweep_win_with_fix(), sweep_win(), sweep_win_with_fix()

Platform: DOS and Windows

---

int init_sweep_with_fix (int **test_screen**, int **speed**, int **direction**, int **duration**)

Purpose: set up the initialization parameters for moving a window across the screen so that it goes through the center of the **test_screen's** location (i.e. the center of a receptive field). Fixspot will sweep with the window.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed** (in 1/100 of deg per sec)
- **direction** (in 1/100 of deg of angle)
- **duration** of sweeping, in milliseconds

Returns: 0 if failed to initialize sweeping, else number of milliseconds left (a multiple of the refresh rate)

See Also: Gsweep(), init_sweep(), sweep_win(), sweep_win_with_fix()

Platform: DOS and Windows

---

int init_toggle (int **first_test_screen**, int **second_test_screen**, int **duration**, int **rate**) <

Purpose: flip between two test_screens

Parameters:

- **first_test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **second_test_screen**
- **duration** of toggling, in milliseconds
- **rate** of toggling (if frames_per_second [fps] is set at 60: 1 = 60 Hz toggling, 2 = 30 Hz toggling, 3 = 20 Hz toggling)

Returns: 0 if failed to initialize sweeping, else number of milliseconds left (a multiple of the refresh rate) and waits for the stimulus to appear before returning (syncs).

See also: toggle_wins()

Platform: DOS and Windows

---

int inp (int **port**)

Returns: a single byte read from port

See also: inpw(), outp(), outpw()

Platform: DOS and Windows

---

int inpw (int **port**)

Returns: two bytes (16 bits) read from port

See also: inp(), outp(), outpw()

Platform: DOS and Windows

---

int in_corridor (float **eye_horizontal**, float **eye_vertical**, int **corridor_width**, float **slope**, float **DC**)

Purpose: A distance value is computed for the indicated position of the eye. The distance is the minimum distance between the straight line trajectory (defined here as an infinitely long line of slope = **slope** starting at the origin (0,0)) and the current eye position. If this distance is within a preset tolerance (the **corridor_width**), the function returns 1, else 0.

Parameters:

- **eye_horizontal** (eye location in horizontal axis)
- **eye_vertical** (eye location in vertical axis)
- **corridor_width** (width of requested corridor)
- **slope** (slope of requested corridor, see find_slope())
- **DC** (denominator constant of corridor, see find_DC())

Returns: 1 = in corridor, 0 = outside corridor

See also: find_slope(), find_DC(), distance_to_line()

Platform: DOS and Windows

---

int in_window (int **test_x**, int **test_y**, int **target_x**, int **target_y**, int **target_size**)

Purpose: test if an point (**test_x**, **test_y**) is within a square target area

Parameters:

- **test_x** - horizontal coordinate of test point
- **test_y** - vertical coordinate of test point
- **target_x** - horizontal coordinate of center of square target area
- **target_y** - vertical coordinate of center of square target area
- **target_size** - size of one side of the square target area

Returns: 1= in the target area, 0= not in the target area

See also: ITEM_POSbind_fixspot()

Platform: DOS and Windows

---

int ITEM_POSbind_fixspot (int **test_screen**, int **item_position**)

Purpose: moves the fixation window to a specified **item_position** (or eye_window position) within a given test_screen.

Parameters:

- **test_screen** (TEST0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN; see css_inc.h)
- **item_position** (1-x)

Returns: 1 if success, 0 if fails

See also: move_eye_window(), move_fixspot(), set_fix_params()

Backward Compatibility: EYEactivate_eyewin(), EYEactivate_item()

Platform: DOS and Windows

---

int ITEM_POSeye_delta (int **test_screen**, int **item_position**, pfloat **x_from_center**, pfloat **y_from_center**, pint **within_target**)

Purpose: get distance of eye_spot from an item's center, quadrant info, and whether eye_pos is within item's size window.

Parameters:

- **test_screen** (TEST0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **item_position** within that test_screen (1-x)
- **x_from_center**
- **y_from_center**
- **within_target** == 1 if (**x_from_center**, **y_from_center**) is within (target center+=target size/2), else 0

Returns:

- 0 if invalid selection
- ITEM_POS_QUAD_UR = upper right quadrant (#include "css_inc.h" to get these codes)
- ITEM_POS_QUAD_UL = upper left
- ITEM_POS_QUAD_LL = lower left
- ITEM_POS_QUAD_LR = lower right

See Also: EYEget_dva(), ITEM_POSeye_ishere(), ITEM_POSeye_iswithin()

Backward Compatibility: get_fixation_posX(), get_fixation_posY()

Platform: DOS and Windows

---

int ITEM_POSeye_ishere (int **test_screen,** int **item_position**)

Purpose: checks if the eye position is within the bounds of an item (or the smallest rectangle that can surround a item of complex shape)

Parameters:

- **test_screen** (TEST0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **item_position** (1-x)

Returns: 1 if eye is within bounds of item, 0 if it is not

See also: EYEget_dva(), ITEM_POSeye_delta(), ITEM_POSeye_iswithin()

Backward Compatibility: EYEis_at_eyewin(), EYEis_at_item()

Platform: DOS and Windows

---

int ITEM_POSeye_iswithin (int **test_screen,** int **item_position,** float **x_size**, float **y_size**)

Purpose: checks if the eye position is within bounds specified by a rectangle with width of **x_size** and height of **y_size** centered around the middle of the specified target. This is a more conservative version of ITEM_POSeye_ishere().

Parameters:

- **test_screen** (TEST0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **item_position** (1-x)
- **x_size** limits x size
- **y_size** limits y size<

Returns: 1 if eye is within specified bounds (center +-size/2), 0 if it is not

See also: EYEget_dva(), ITEM_POSeye_isdelta(), ITEM_POSeye_ishere()

Platform: DOS and Windows

---

int ITEM_POSget (int **test_screen**, int **item_position**, pfloat **x_center**, pfloat **y_center,** pfloat **x_size**, pfloat **y_size**)

Purpose: gets the center and size (in degrees of visual angle) of any item. ITEM_POS dynamically tracks all items, accurately recording their position even when involved in complicated sweeping or other procedures. (Although ITEM size information returned may not be accurate for all frames of a movie with varyingly sized frames).

Parameters:

- **test_screen** (TEST0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **item_position** within that **test_screen** (1-n)
- **x_center** (upon calling function, this pointer will store the x_center value here)
- **y_center** (upon calling function, this pointer will store the y_center value here)
- **x_size** (upon calling function, this pointer will store the x_size value here)
- **y_size** (upon calling function, this pointer will store the y_size value here)

Returns: 1 if successful, 0 if an invalid selection

Backward Compatibility: get_posX(), get_posY()

Platform: DOS and Windows

---

int ITEM_POSlut_index (int **test_screen**, int **item_position**)

Purpose: ONLY way to get the index which has to be passed to set_color() and other color-related functions.

Parameters:

- **test_screen** (TEST0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **item_position** (1-x)

Returns: the index in the current LUT in which the requested item's color is stored. This function is used when the user wishes to change the color of an item, followed by a call to GcolorABS(), GcolorLUT(), GcolorREL(), set_CLT_load_index(), set_colorABS(), or set_color_REL().

See Also: GcolorABS(), GcolorLUT(), GcolorREL(), load_CLT(), set_CLT_load_index(), set_colorABS(), set_color_REL()

Platform: DOS and Windows

---

void ITEM_POSmark_pos (int **test_screen,** int **item_position,** int **color**)

Purpose: draws a box marking the center and size of the item on the USER screen. Is NOT dynamically updated if the item changes position (such as when it is sweeping).

Parameters:

- **test_screen** (TEST0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **item_position** (1-x)
- **color** (#include "css_inc.h" and use the colors listed there)

Returns: nothing

See also: DrawBox()

Backward Compatibility: mark_eyewin(), mark_item(), mark_pos()

Platform: DOS and Windows

---

int KeyGet ()

Purpose: waits for a key press and returns the key and attributes.

Parameters: none.

Returns: (unsigned) casting as (char) will provide the key that was pressed, and the high byte contains special attributes (such as SHIFT or CTRL). Or, one can compare directly with the #defines in css_inc.h (be sure to #include "css_inc.h"). Internally, this function calls the GetAKey() function.

See Also: GetAKey()

Platform: DOS and Windows

---

int KeyHit ()

Purpose: checks if a key has been pressed. Removes the key from the stdin buffer and erases it.

Parameters: none

Returns: 1 if TRUE, 0 if FALSE

See also: KeyPressed()

Platform: DOS and Windows

---

int KeyPressed ()

Purpose: checks if a key has been pressed. Leaves the pressed key in the stdin buffer, so GetAKey() can retrieve it.

Parameters: none

Returns: 1 if TRUE (anything in char buffer), 0 if FALSE

See Also: GetAKey(), KeyHit()

Platform: DOS and Windows

---

void load_CLT (int **index**)

Purpose: changes the entire color palette mid-trial using preloaded CLTs (from the CLT menu). Refer to them by their index position (1-n).

Parameters:

- **index** - number (1 through n) of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.

Returns: nothing.

See also: GcolorABS(), GcolorREL(), GcolorLUT(), ITEM_POSlut_index(), set_CLT_load_index(), set_colorABS(), set_colorREL()

Platform: DOS and Windows

---

int load_CLT_subset (int **num_entries**, int **CLTsource**, int **src_start**, int **dst_start**)

Purpose: loads part of a preloaded CLT (from the CLT menu) into the active CLT.

Parameters:

- **num_entries** - the number of entries that should be loaded
- **CLTsource** - number (1 through n) of the CLT as it was loaded into Cortex, through the LUT:Get:From_Disk menu. The first CLT number should be 1.

- **src_start** - the starting index in the source (temporary) CLT, a number between 0 and 255
- **dst_start** - the starting index in the destination (active) CLT, a number between 0 and 255

Returns: 1 if successful, 0 if error (e.g. invalid CLT or range)

Platform: DOS and Windows

---

float log (float **value**)

Purpose: The function returns the logarithm of x.

Returns: natural log of **value**

See also: exp(), log10()

Platform: DOS and Windows

---

float log10 (float **value**)

Purpose: The function returns the base 10 logarithm of x.

Returns: logarithm (base 10) of **value**

See also: log()

Platform: DOS and Windows

---

long lseek (int **handle**, long **offset**, int **origin**)

Purpose: moves the current file pointer (referred to by **handle**) position to **offset** from **origin**. **origin** must equal one of the following constants: SEEK_SET (beginning of file); SEEK_CUR (current position of pointer); SEEK_END (end_of_file). These constants are defined in css_inc.h so be sure to #include "css_inc.h" in your timing file.

Parameters:

- **handle** (handle to a currently open file)
- **offset** (number of bytes from origin to set new pointer position)
- **origin** (see description in purpose section of this page)

Returns: the offset in bytes of the new position of the pointer from the beginning of the file. Returns -1L if there is an error.

See also: tell()

Platform: DOS and Windows

---

pchar malloc (int **number_of_bytes**)

Purpose: allocates a block of memory. Note that the sizeof() command may be used to assess the size of any data type

automatically.

Parameters:

- **number_of_bytes** (the number of bytes to allocate)

Returns: a pointer to the allocated space, or 0 if there is insufficient memory available.

See Also: calloc(), free(), realloc()

Platform: DOS and Windows

---

int max (int **value_1**, int **value_2**)

Purpose: find maximum of two integers

Parameters:

- **value_1** first value to be compared
- **value 2** second value to be compared

Returns: the value that is larger

See Also: min()

Platform: DOS and Windows

---

pchar memchr (pchar **buf**, int **c**, int **count**)

Purpose: Finds characters in a buffer.

Parameters:

- **buf** - buffer to search
- **c** - character to look for
- **count** - number of characters to check

Returns: If successful, a pointer to the first location of **c** in **buf**; otherwise, it returns 0.

Platform: DOS and Windows

---

int memcmp (pchar **buf1**, pchar **buf2**, int **count**)

Purpose: Compare characters in two buffers.

Parameters:

- **buf1** - first buffer
- **buf2** - second buffer
- **count** - number of characters to compare

Returns: The return value indicates the relationship between the buffers:

- < 0 indicates buf1 less than **buf2**

- = 0 indicates but1 identical to **buf2**
- >0 indicates buf1 greater than **buf2**

Platform: DOS and Windows

---

pchar memcpy (pchar **dest**, pchar **src**, int **count**)

Purpose: Copies characters between buffers.

Parameters:

- **dest** - new buffer
- **src** - buffer to copy from
- **count** - number of characters to copy

Returns: the value of **dest**.

Platform: DOS and Windows

---

pchar memmove (pchar **dest**, pchar **src**, int **count**)

Purpose: Moves one buffer to another.

Parameters:

- **dest** - destination buffer
- **src** - source buffer
- **count** -number of bytes of characters to copy

Returns: the value of **dest**.

Platform: DOS and Windows

---

pchar memset (pchar **dest**, int **c**, int **count**)

Purpose: sets the first count bytes of **dest** to the character **c**.

Parameters:

- **dest** - destination buffer
- **c** - character to use for filling the buffer
- **count** - number of characters to set

Returns: the value of **dest**.

Platform: DOS and Windows

---

void MessageFloat (int **message_field**, float **value**)

Purpose: prints a floating point value to the USER status screen. To take advantage of this, you add the appropriate GETSmessage lines to CORTEX.CFG.

Parameters:

- **message_field** in which to put the value (1-5) (corresponds to GETSmessage1,2...)

- the floating-point **value** to be displayed

Returns: nothing.

See Also: MessageInt(), MessageLong(), MessageString(), Mprintf()

Platform: DOS and Windows

---

void MessageInt (int **message_field**, int **value**)

Purpose: prints an int to the USER status screen. To take advantage of this, you add the appropriate GETSmessage lines to CORTEX.CFG.

Parameters:

- **message_field** in which to put the value (1-5) (corresponds to GETSmessage1,2...)
- the integer **value** to be displayed

Returns: nothing.

See Also: MessageFloat(), MessageLong(), MessageString(), Mprintf()

Platform: DOS and Windows

---

void MessageLong (int **index**, long **message**)

Purpose: Prints a message to the user's status screen. To take advantage of this, you add the appropriate GETSmessage lines to cortex.cfg.

Parameters:

- **message_field** in chich to put the value (1-5) (corresponds to GETSmessage1,2...)
- the number to be shown in the **message** window.

Returns: nothing.

Platform: DOS and Windows

---

void MessageString (int **message_field**, pchar **string**)

Purpose: prints a string to the USER status screen. To take advantage of this, you add the appropriate GETSmessage line to CORTEX.CFG.

Parameters:

- **message_field** in which to put the value (1-5) (corresponds to GETSmessage1,2...)
- the pointer to the **string** to be displayed

Returns: nothing.

See Also: MessageFloat(), MessageInt(), MessageLong(), Mprintf()

Platform: DOS and Windows

---

int min (int **value_1**, int **value_2**)

Purpose: find the smaller of two values

Returns: the minimum of two integers value_1 and value_2

See Also: max()

Platform: DOS and Windows

---

int mkdir (pchar **dirname**)

Purpose: creates the directory indicated by **dirname**

Parameter: dirname - path for new directory

Returns: 0 if successful

See also: chdir(), getcwd(), rename(), rmdir()

Platform: DOS and Windows

---

int MouseMoved (pint **deltaX**, pint **deltaY**, pint **buttons**)

Purpose: checks the distance deltaX and deltaY of the mouse since last call. Paramaters:

- **deltaX** (points to the value filled by MouseMoved() and represents the distance in the X-axis the mouse has been moved.)
- **deltaY** (points to the value filled by MouseMoved() and represents the distance in the Y-axis the mouse has been moved.)
- **buttons** (points to the value filled by MouseMoved() and can be compared to values listed in css_inc.h to recover which buttons are currently being pressed: LEFT_BUTTON, RIGHT_BUTTON, BOTH_BUTTONS. Be sure to #include "css_inc.h" in you timing file.)

Returns: TRUE if at least one button is being pressed, FALSE if none

See also: MousePressed()

Platform: DOS and Windows

---

int MousePressed (pint **buttons**)

Purpose: checks to see if any of the mouse buttons are currently being pressed. Paramaters:

- **buttons** (points to the value filled by MousePressed() and can

be compared to values listed in css_inc.h to recover which buttons are currently being pressed: LEFT_BUTTON, RIGHT_BUTTON, BOTH_BUTTONS. Be sure to #include "css_inc.h" in you timing file.)

Returns: TRUE if at least one button is being pressed, FALSE if none

See also: MouseMoved()

Platform: DOS and Windows

---

void move_eye_window (int **horiz_offset**, int **vert_offset**)

Purpose: move the fixation window without moving the fixspot

Parameters:

- **horiz_offset** from reference point (in 1/100 of deg of visual angle)
- **vert_offset** from reference point (in 1/100 of deg of visual angle)

Returns: nothing

See also: ITEM_POSbind_fixspot(), move_fixspot(), set_fixwin_params()

Platform: DOS and Windows

---

void move_fixspot (int **visible**, int **horiz_offset**, int **vert_offset**)

Purpose: moves and turns on or off the fixation spot. The fixation window automatically moves with the fixspot. To separate the fixspot and fixation window, call move_eye_window().

Parameters:

- **visible** (0 = turn fixspot off, 1 = turn it on)
- **horiz_offset** (from reference point in 1/100 of deg of visual angle)
- **vert_offset** (from reference point in 1/100 of deg of visual angle)

Returns: nothing

See also: GmoveREL(), ITEM_POSbind_fixspot(), move_eye_window(), set_fixwin_params()

Platform: DOS and Windows

---

void move_sample (int **visible**, int **horiz_offset**, int **vert_offset**)

Purpose: moves and turns on or off the sample stimulus (the sample stimulus is defined as TEST0)

Parameters:

- **visible** (0 = turn TEST0 off, 1 = turn it on)
- **horiz_offset** (from reference point in 1/100 of deg of visual angle)
- **vert_offset** (from reference point in 1/100 of deg of visual angle)

Returns: nothing

See also: GmoveREL(), move_test()

Platform: DOS and Windows

---

void move_test (int **test_screen**, int **visible**, int **horiz_offset**, int **vert_offset**)

Purpose: turns on or off the specified **test_screen** and moves it

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **visible** (0 = turn test off; 1 = turn test on)
- **horiz_offset** from reference point (in 1/100 of deg of visual angle)
- **vert_offset** from reference point (in 1/100 of deg of visual angle)

Returns: nothing

See also: GmoveREL(), move_sample()

Platform: DOS and Windows

---

void Mprintf (int **message_field**, pchar **string**)

Purpose: prints a string to the USER status screen. To take advantage of this, you add the appropriate GETSmessage? line to CORTEX.CFG. Unlike MessageString(), Mprintf() will accept strings using the format flags used in printf().

Parameters:

- **message_field** in which to put the value (1-5) (corresponds to GETSmessage1,2...)
- the pointer to the **string** to be displayed, or string with format flags in quotes variable this depends on if there are flags in argument (2) to define...just as in printf()

Returns: nothing.

See Also: MessageFloat(), MessageInt(), MessageLong(), MessageString()

Platform: DOS and Windows

---

long MS_TIMERcheck (int **timer_number**)

Purpose: checks the number of milliseconds left in the count-down of timer number **timer_number**

Parameters:

- **timer_number** (the timer_number set in MS_TIMERset())

Returns: number of MS left (0 when finished)

See also: MS_TIMERset(), set_timer(), timer_expired()

Platform: DOS and Windows

---

int MS_TIMERset (int **timer_number**, long **milliseconds**)

Purpose: sets the timer numbered **timer_number** to **milliseconds**. There are a maximum of 32 timers which can be set in this way besides the timer set by set_timer(). This allows more flexibility in time management than in earlier versions of CORTEX, but beware that the only way to retrieve the current countdown for a given MS_TIMER is with the MS_TIMERcheck() command. timer_expired() will not check these timers.

Parameters:

- **timer_number** (maximum value can be 32)
- **milliseconds** (number of milliseconds in countdown)

Returns: 1 if successful

See also: MS_TIMERcheck(), set_timer(), timer_expired()

Platform: DOS and Windows

---

void no_fixation ()

Purpose: Records in the data file that the subject never achieved fixation. This function also displays a "no fixation" message in the *status* of the USER screen.

Parameters: none

Returns: nothing

See also: encode()

Platform: DOS and Windows

---

int open (pchar **filename**, int **oflag**, int **permission_mode**)

Purpose: opens a file named filename and bestows upon it the attributes set with **oflag** and **permission_mode**. Detailed descriptions of the constants to be used with **oflag** and permission_mode can be found in any ANSI manual. The oflag

constants can be ored (|) with each other to make various combinations. They are called, briefly: _O_APPEND (repositions the file pointer to the end-of-file before writing); _O_BINARY (opens file in binary mode); _O_CREAT (creates and opens a new file, fails if **filename** already exists); _O_EXCL (returns an error if filename already exists--used only in conjunction with _O_CREAT); _O_RDONLY (opens a file for reading only...cannot be given simultaneously as _O_RDWR or _O_WRONLY); _O_RDWR (opens a file for both reading and writing...cannot be given simultaneously as _O_RDONLY or _O_WRONLY); _O_TEXT (opens a file in text mode); _O_TRUNC (opens and erases a file...cannot be given simultaneously as _O_RDONLY); _O_WRONLY (opens a file for writing only...cannot be given simultaneously as _O_RDWR or _O_RDONLY). The **permission_mode** constants are ignored unless the _O_CREAT **oflag** is called and can be ored (|) with each other to allow for both reading and writing. They are called, briefly: _S_IREAD (reading permitted, only); _S_IWRITE (writing permitted, only).

Parameters:

- **filename** (name of file to be opened)
- **oflag** (file attributes)
- **permission_mode** (file permissions. Only used with the _O_CREAT file attribute.)

Returns: the file handle for the opened file. return value of -1 indicates an error.

See also: close(), dup(), dup2()

Platform: DOS and Windows

---

int outp (int **port**, int **data**)

Purpose: outputs a single byte of data on port

Parameters:

- **port** (the **port** on **device_number** to output the **data** through)
- **data** (a single byte (8 bits) of data)

Returns: the **data** sent if successful, else -1

See also: inp(), inpw(), outpw()

Platform: DOS and Windows

---

int outpw (int **port**, int **data**)

Purpose: outputs a two bytes of **data** on **port**

Parameters:

- **port** (the port on device_number to output the data through)
- **data** (two bytes (16 bits) of data)

Returns: the **data** sent if successful, else -1

See also: inp(), inpw(), outp()

Platform: DOS and Windows

---

int pan_win (int **test_screen**, int **speed**, int **direction**)

Purpose: pan a **test_screen** inside a window

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed** of movement, in units of 1/100 of deg per second (e.g. 60 = 1 deg/sec)
- **direction** of panned motion, in 1/100 of deg of angle (e.g. 1800 = 30 degrees)

Returns: time remaining (a multiple of the refresh rate in milliseconds). Returns 0 when panning done.

See also: Gpan(), GpanABS(), GpanREL(), init_pan(), pan_wkstABS(), pan_wkstREL()

Platform: DOS and Windows

---

void pan_wkstABS (int **test_screen**, int **X_offset**, int **Y_offset**)

Purpose: to have the **test_screen's** window originate at new absolute coordinates. NOTE: The image moves, but the window stays put. This function is normally used for manual panning of **test_screens** within windows. It is much easier to use the pan_win() function if possible, since it does all of the size calculations and repositioning itself.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **X_offset** from lower left of test_screen (in 1/100 of deg)
- **Y_offset** from lower left of test_screen (in 1/100 of deg)

Returns: nothing

See Also: Gpan(), GpanABS(), GpanREL(), init_pan(), pan_win(), pan_wkstREL()

Platform: DOS and Windows

---

void pan_wkstREL (int **test_screen**, int **X_offset**, int **Y_offset**)

Purpose: to have the test_screen window originate at new absolute coords. NOTE: that the image moves, but the window stays put. This function is normally used for manual panning of test_screen within windows. It is much easier to use the pan_win() function if possible,

since it does all of the size calculations and repositioning itself.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **X_offset** from lower left of current position (in 1/100 of deg)
- **Y_offset** from lower left of current position (in 1/100 of deg)

Returns: nothing

See also: Gpan(), pan_win(), pan_wkstABS()

Platform: DOS and Windows

---

void polar2cartesian (float **r**, float **theta**, pfloat **x**, pfloat **y**)

Purpose: A polar to Cartesian transform which computes the x and y, given the phase angle **theta** (in degrees) and the magnitude **r**.

Parameters:

- **r** - the magnitude
- **theta** - the angle
- **x** and **y** are pointers to the Cartesian coordinates

Returns: nothing

Platform: DOS and Windows

---

float polar2x (float **r**, float **theta**)

Purpose: find x position from polar coordinates, **r** and **theta**

Parameters:

- **r** - the magnitude
- **theta** - the angle

Returns: the Cartesian x coordinate

Platform: DOS and Windows

---

float polar2y (float **r**, float **theta**)

Purpose: find y position from polar coordinates, **r** and **theta**

Parameters:

- **r** - the magnitude
- **theta** - the angle

Returns: the Cartesian y coordinate

Platform: DOS and Windows

---

float pow (float **x**, float **y**)

Purpose: find x^y

Returns: floating point power function $\mathbf{x^y}$

See also: exp(), log(), log10(), sqrt()

Platform: DOS and Windows

---

int printf (pchar **string**)

Purpose: prints a string to the text screen at the current cursor position. Formatting in the **string** (ie. '\n') behaves as in ANSI C printf() calls. The USER screen must be in text mode for this call to be effective. SCREENmode(MODE_TEXT) will change the screen from graphics mode to text mode. To return the user screen to normal, you must call SCREENmode(MODE_GRAPHICS), followed by SCREENdraw_entire_screen(). printf() will technically print to the screen in MODE_GRAPHICS screen as well as MODE_TEXT, but it will ignore the current graphics and make a mess.

Parameters:

- **string** (with a variable number of extra parameters, depending of the formatting of **string**)

Returns: the number of characters printed, or -1 if there is an error

See Also: printxy(), putchar(), puts(), SCREENmode()

Platform: DOS only

---

void printxy (pchar **string**, int **column**, int **row**, int **attributes**)

Purpose: prints a **string** to the text screen at cursor position (**row**, **column**) and with the text attributes (**attributes**). Formatting in the **string** (ie. '\n') behaves as in ANSI C printf() calls. The USER screen must be in text mode for this call to be effective. SCREENmode(MODE_TEXT) will change the screen from graphics mode to text mode. To return the user screen to normal, you must call SCREENmode(MODE_GRAPHICS), followed by SCREENdraw_entire_screen(). NOTE: text mode attribute table (add 16 to these values for blinking characters)

Parameters:

- **string** (with a variable number of extra parameters, depending of the formatting of **string**)
- **column**, **row** - location to print
- **attributes** - If css_inc.h is included in the timing file, then the following color attributes are recognized: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN,

LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE. (In DOS Cortex, add 16 to these values for blinking characters.)

Returns: nothing

Example:
Platform: DOS only

---

int putchar (int **character**)

Purpose: writes a single **character** to the current position of the cursor on the screen (text mode only).

Returns: the **character**, if successful; otherwise, it returns EOF if it is an error or an end-of-file condition

See also: printf(), printxy(), puts(), SCREENmode()

Platform: DOS only

---

int puts (pchar **string**)

Purpose: writes a **string** to the current position of the cursor on the screen (text mode only)

Parameter:

- **string** Output string

Returns: non-zero if successful, 0 if failed

See also: printf(), printxy(), putchar(), SCREENmode()

Platform: DOS only

---

void put_eye_data_in_buf (int **on_off**)

Purpose: starts (**on_off**=1) or stops (**on_off**=0) storing the eye movement data in a buffer. NOTE: you must still set the "save eye data" variable in the Run:

Parameters: General menu if you want the eye buffer to be stored to disk (without this the data will be stored just long enough to draw it to screen). If you turn the buffer storage on, it is good form to turn it off before the end of the trial. Also, be sure to call encode() with the correct code for starting/stopping the storage (i.e. encode(START_EYE_DATA) or encode(END_EYE_DATA)). These codes will let the analysis programs know when the eye data started and stopped. The purpose of this function is to allow you to limit saving eye data to just the portion of trial that is relevant. Eye data is always sampled - this function simply saves it to an internal buffer. If you save the buffer to the disk data file, this function will limit the amount of disk space required for saving eye data. NOTE: In previous versions of CORTEX, all eye data was stored to the buffer whether you asked for it or not. The "save eye data" variable must still be set in the menus for the buffer to be written to the disk data

file.

Parameters: start or stop (1 = start, 0 = stop) e.g. put_eye_data_in_buf(1)

Returns: nothing

See Also: display_eye_path()

Platform: DOS and Windows

---

int rand()

Purpose: Generates a pseudorandom number between 0 and RAND_MAX (32767). Use the srand() function to seed the pseudorandom-number generator before calling rand. This function is internally seeded inside Cortex, but may be seeded in a timing file with srand(). rand() and srand() are the standard C function for generating a pseduo-random number.

Returns: a pseudorandom number. There is no error return.

See also: srand()

Platform: DOS and Windows

---

int rand2 ()

Purpose: Generates a pseudorandom number between 0 and RAND_MAX (32767). Use the srand2() function to seed the pseudorandom-number generator before calling rand2. rand2() and srand2() use a different algorithm than rand() and srand(), and may provide better pseudo-random numbers than those obtained from rand() and srand().

Returns: a pseudorandom number. There is no error return.

See also: srand2()

Platform: DOS and Windows

---

int random (int **minimum**, int **maximum**)

Purpose: returns a random number; typically used for randomizing a time interval. (e.g. typical use: set_timer(random(50,100));) This function internally calls srand2(). It provides a way of specifying a maximum and minimum range for the pseudo-random numbers.

Parameters:

- **minimum** size of random number.
- **maximum** size of random number

Returns: a random integer number between **minimum** and (**maximum**-1).

See also: srand2()

Platform: DOS and Windows

---

int read (int **handle**, pchar **buffer**, int **count**)

Purpose: reads data from a file

Parameters:

- **handle** - the file handle of an open file
- **buffer** - pointer to previously allocated storage location for incoming information
- **count** - number of bytes to read

Returns: number of bytes actually read

See also: open(), write()

Platform: DOS and Windows

---

pchar realloc (pchar **memory_block**, int **number_of_bytes**)

Purpose: Reallocate memory blocks by changing the size of a currently allocated **memory_block** to **number_of_bytes**. Use sizeof() to assess the size of any data type.

Parameters:

- **memory_block** - Pointer to previously allocated memory block
- **number_of_bytes** - New size in bytes

Returns: a pointer to the first byte in the newly allocated array if succesful, otherwise '\0'.

See Also: calloc(), free(), malloc()

Platform: DOS and Windows

---

int recent_block_status (int **minimum_correct**, int **last_several_trials**)

Purpose: Tells whether at least X of last Y trials were correct.

Parameters:

- **minimum_correct** - minimum number of correct trials
- **last_several_trials** - number of trials to examine

Returns: 0 if the number correct in the **last_several_trials** < **minimum_correct** 1 if the number correct in the **last_several_trials** **minimum_correct** -1 if the number of total trials < **last_several_trials** (indicating an error)

Platform: DOS and Windows

---

int remove (pchar **filename**)

Purpose: deletes the file indicated by **filename**

Parameters: file to remove

Returns: 0 if successful

See also: chdir(),getcwd(), mkdir(), rename(), rmdir()

Platform: DOS and Windows

---

int rename (pchar **oldname**, int **newname**)

Purpose: renames a file or directory from **oldname** to **newname**. The old name must be the path of an existing file or directory. The new name must not be the name of an existing file or directory. You can use rename to move a file from one directory or device to another by giving a different path in the newname argument. However, you cannot use rename to move a directory. Directories can be renamed, but not moved.

Parameters:

- **oldname** Pointer to old name
- **newname** Pointer to new name

Returns: 0 if successful, non-zero on an error

See also: chdir(), getcwd(), mkdir(), remove(), rmdir()

Platform: DOS and Windows

---

void repeat_block_if_pct_correct (int **direction**, int **limiting_pct**)

Purpose: Repeats the current block if its percent correct is greater than or less than an amount.

Parameters:

- **direction** - direction of the relational operator. That is, 0 means "<=", and 1 means ">="
- **limiting_pct** - a number between 0-10000 to represent the percent correct

Returns: nothing

Example: repeat_block_if_pct_correct(0, 5000) means repeat this block if its percent correct <= 50 percent.

Platform: DOS and Windows

---

void repeat_cond_if_pct_correct (int **direction**, int **limiting_pct**)

Purpose: Repeats the current condition if its percent correct is greater than or less than an amount. Can also be used to

unconditionally repeat the condition if an error occurs.

Parameters:

- **direction** - direction of the relational operator. That is, 0 means "<=", and 1 means ">="0
- **limiting_pct** - a number between 0-10000 to represent the percent correct

Returns: nothing Example: repeat_block_if_pct_correct(0, 5000) means repeat this condition if its percent correct <= 50 percent.

Platform: DOS and Windows

---

void response_before_test (int **response**)

Purpose: Records in the *response_error* field of the trial header, the error code (#7) for a behavioral response before the presentation of the test stimulus (i.e. an aborted trial). Also records in the *response* field header the code for the response for this trial. The status field in the USER screen will reflect this message. (The *expected_response* field of the header gets set during the trial from the value under the TRIAL_TYPE heading of the conditions file.)

Parameters:

- **response** (*one of the following):*

```
#: Meaning

0: NO_TRIAL_TYPE
1: MOVING
2: IMMEDIATE_RELEASE
3: DELAYED_RELEASE
4: TEST1_LEFT_LEVER
5: TEST1_RIGHT_LEVER
6: TEST2_LEFT_LEVEL
7: TEST2_RIGHT_LEVER
8: TEST3_LEFT_LEVER
9: TEST3_RIGHT_LEVER
10: TEST4_LEFT_LEVER
11: TEST4_RIGHT_LEVER
12: TEST5_LEFT_LEVER
13: TEST5_RIGHT_LEVER
14: ON_OFF
15: EYE_MOVEMENT
16: TEST6_LEFT_LEFT
17: TEST6_RIGHT_LEVER
18: TEST7_LEFT_LEVER
19: TEST7_RIGHT_LEVER
20: TEST8_LEFT_LEVER
21: TEST8_RIGHT_LEVER
22: TEST9_LEFT_LEVER
23: TEST9_RIGHT_LEVER
24: TEST1_EXTRA_LEVER
25: TEST2_EXTRA_LEVER
26: TEST3_EXTRA_LEVER
27: TEST4_EXTRA_LEVER
28: TEST5_EXTRA_LEVER
29: TEST6_EXTRA_LEVER
30: TEST7_EXTRA_LEVER
31: TEST8_EXTRA_LEVER
32: TEST9_EXTRA_LEVER
```

Platform: DOS and Windows

---

void response_correct (int **response**)

Purpose: Records in the *response_error* field of the trial header, the error code (#0) for a correct behavioral response. Also records in the *response* field header the code for the **response** for this trial. The status field in the USER screen will reflect this message. (The *expected_response field* of the header gets set during the trial from the value under the TRIAL_TYPE heading of the conditions file.)

Parameters:

- **response** (*see response codes* in response_before_test())

Returns: nothing

Platform: DOS and Windows

---

void response_early (int **response**)

Purpose: Records in the *response_error* field of the trial header the code (#5) for behavioral response earlier than expected (used, for instance, when the subject anticipates the stimulus). Also records in the *response* field header the code for the **response** for this trial. The status field in the USER screen will reflect this message. (The *expected_response* field of the header gets set during the trial from the value under the TRIAL_TYPE heading of the conditions file.)

Parameters:

- **response** (*see response codes* in response_before_test())

Returns: nothing

Platform: DOS and Windows

---

void response_late (int **response**)

Purpose: Records in the *response_error* field of the trial header the code (#2) for a behavioral response later than expected (used, for instance, to see if the monkey is responding too slowly to a stimulus). Also records in the *response* field header the code for the **response** for this trial. The status field in the USER screen will reflect this message. (The *expected_response* field of the header gets set during the trial from the value under the TRIAL_TYPE heading of the conditions file.)

Parameters:

- **response** (*see response codes* in response_before_test())

Returns: nothing

Platform: DOS and Windows

---

void response_missing (int **response**)

Purpose: Records in the *response_error* field of the trial header the code (#1) for the absence of any behavioral response in the trial. Also records in the *response* field header the code for the **response** for this trial. The status field in the USER screen will reflect this message. (The *expected_response* field of the header gets set during the trial from the value under the TRIAL_TYPE heading of the conditions file.)

Parameters:

- **response** (*see response codes* in response_before_test())

Returns: nothing

Platform: DOS and Windows

---

void response_no_bar_down (int **response**)

Purpose: Records in the *response_error* field of the trial header the code for a response of no bar down at the start of the trial. Also records in the *response* field header the code for the **response** for this trial. The status field in the USER screen will reflect this message.**Platform:** DOS and Windows

---

void response_wrong (int **response**)

Purpose: Records in the *response_error* field of the trial header the code (#6) for an unexpected or incorrect response (i.e.: a left response on a trial when right was expected). Also records in the *response* field header the code for the **response** for this trial. The status field in the USER screen will reflect this message.**Platform:** DOS and Windows

---

void reward ()

Purpose: gives the subject a reward (assumes that the solenoid is connected as specified in the Cortex User's Manual).

Parameters: none

Returns: nothing.

Platform: DOS and Windows

---

int rmdir (pchar **dirname**)

Purpose: deletes the directory indicated by **dirname**. The directory must be empty and must not be either the current working directory or the root directory.

Parameter:

- **dirname** Path of directory to be removed

Returns: 0 if successfully deleted. A return value of -1 indicates an error.l

See also: chdir(), rename(), mkdir()

Platform: DOS and Windows

---

int run_movie (int **test_screen**, int **direction**)

Purpose: starts the movie running

Parameters:

- **test_screen** (0 = TEST0, 1 = TEST1, ...)
- **direction of movement** (0 = forwards, 1 = backwards)

Returns: number of milliseconds left (a multiple of the refresh rate; 0 when finished). Turns the move off and waits for it to disappear when finished.

See also: init_movie(),

Platform: DOS and Windows

---

int scanf (pchar **string**)

Purpose: Read formatted data from the standard input stream. Formatting in the **string** (ie. '\n') behaves as in ANSI C scanf() calls.

Parameters:

- **string** (with a variable number of extra parameters, depending of the formatting of **string**)

Returns: the number of characters successfully read. The return value is EOF for an error or if the end-of-file character or the end-of-string character is encountered in the first attempt to read a character.

Platform: DOS only

---

void SCREENcalc_fixwin (**void**)

Purpose: Draw boundaries and crosshairs of active eyewindow on screen

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void SCREENdraw_box_on_eog (float **centx**, float **centy**, float **width**, float **height**, int **color**)

Purpose: Draw a box of the given size and color on the eog area of

the user's screen.

Parameters:

- **centx** - x coordinate of the center of the box, in degrees of visual angle
- **centy** - y coordinate of the center of the box, in degrees of visual angle
- **width** - width of the box, in degrees of visual angle
- **height** - width of the box, in degrees of visual angle
- **color** - the color of the box to be drawn. If css_inc.h is included in the timing file, then the following color values are recognized: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE

Returns: nothing

Platform: DOS and Windows

---

void SCREENdraw_entire_screen ()

Purpose: draws the entire USER screen (when it is in graphics mode). This is useful when returning from text mode to graphics mode (ie. with a call to SCREENmode(MODE_GRAPHICS)). In that case, the screen will be blank, so the histograms, rasters, etc. need to be redrawn.

Parameters: none

Returns: nothing

See also: SCREENmode()

Platform: DOS and Windows

---

void SCREENdraw_eye_path (int **show**)

Purpose: Draw the path on the user screen of the eye movement, up to a given point in a trial. Also, called from within display_eye_path() function.

Parameters:

- **show** - show or erase the eye path (1 = show, 0 = erase)

Returns: nothing

Platform: DOS and Windows

---

void SCREENdraw_fixwin (int **color**)

Purpose: Draw the fixation window on the user screen, in the specified color.

Parameters:

- **color** - the color of the box to be drawn. If css_inc.h is included in the timing file, then the following color values are recognized: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE

Returns: nothing

Platform: DOS and Windows

---

void SCREENdraw_histograms ()

Purpose: Draw the histograms on the user screen.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void SCREENdraw_raster_tables ()

Purpose: Draw the raster tables on the user screen.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void SCREENerase_histograms ()

Purpose: Erase the histograms. This function only erases the graph. The tik marks remain until the rasters are cleared.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void SCREENerase_raster_tables ()

Purpose: Erase the raster tables.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void SCREENerase_rasters ()

Purpose: Clears the rasters and the entire histogram, including the tiks.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void SCREENmode (int **mode**)

Purpose: Sets the mode of the USER screen to text (**mode**=1) or graphics (**mode**=0). This is useful if there is user interaction during a trial with keyboard input (text mode is best here), then reset the USER screen to graphics mode (followed by a call to SCREENdraw_entire_screen()) when histograms/eye- position data need to be supervised.

Parameters:

- **mode** - MODE_TEXT for text mode, MODE_GRAPHICS for graphics mode (must #include "css_inc.h")

Returns: nothing

See also: SCREENdraw_entire_screen()

Platform: DOS only

---

void SCREENshow_eye_path ()

Purpose: Draws the path on the user screen of the eye movement, up to a given point in a trial. When the function is called, the user screen will only display the EOG window. Cortex will wait for the user to press any key to display the eye path. When the user presses another key, the eye path will be erased. To return the user screen to normal, you must call SCREENmode(MODE_GRAPHICS), followed by SCREENdraw_entire_screen(). SCREENshow_eye_path() function is actually the function that is called internally by Cortex when *eye_Trail* is chosen from the menu.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

void SCREENupdate_eye_pos (int **x**, int **y**)

Purpose: Updates the eye position on the user screen. This function should not need to be called in the timing file, since this operation is done automatically by Cortex. The style of the dots is determined by the EOG_STYLE setting in the Cortex.cfg file.

Parameters:

- **x** - the x coordinate of the eye position
- **y** - the y coordinate of the eye position

Returns: nothing

Platform: DOS and Windows

---

void SCREENuser_display(int **on_off**)

Purpose: Turns the user display off or on, so that the display can be turned it off without setting the MONITOR_TYPE parameter in Cortex.cfg to NONE. This same functionality can also be accomplished with the "User Graphical Display on?" setting in the Run:

Parameters: General menu option.

Parameters:

- **on_off** - turn the display on (1) or off (0)

Returns: nothing

Platform: DOS only

---

int scroll_win (int **test_screen**, int **speed**, int **direction**)

Purpose: scroll a **test_screen** across the screen. Unlike sweep(), the motion starts at the item location rather than offset from the center of motion. The init_scroll() function must be called first.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed of scrolling**, in units of 1/100 of deg per second (e.g. 100 = 1 deg/sec)
- **direction** of motion, in 1/100 of deg of angle (e.g. 100 = 1 deg)

Returns: time remaining (in milliseconds; a multiple of the refresh rate). returns 0 when scrolling is done.

See also: Gscroll(), init_scroll(), scroll_win_with_fix()

Platform: DOS and Windows

---

int scroll_win_with_fix (int **test_screen**, int **speed**, int **direction**)

Purpose: scroll a **test_screen** across the screen. Unlike sweep(), the motion starts at the item location rather than offset from the center of motion. The fixspot tracks the scrolling, but it need not be centered on same point. The init_scroll() function must be called first.

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **speed** of scrolling, in units of 1/100 of deg per second (e.g. 100 = 1 deg/sec)
- **direction** of motion, in 1/100 of deg of angle (e.g. 100 = 1 deg)

Returns: time remaining (in milliseconds; a multiple of the refresh rate). Returns 0 when scrolling is done.

See also: Gscroll(), init_scroll(), scroll_win()

Platform: DOS and Windows

---

void send_termination_signal(int **port_base**, int **baud**, int **configuration**, pchar **message**);

Purpose: This function sets up a flag and the other parameters, that tell Cortex to send a string out of the given serial port when the trials are over. If this flag is set, the port will be opened with Serial_Open(), the string will be written, one byte at a time, with Serial_Write(), and then the port will be closed with Serial_Close(). If the flag has not been set, the internal Cortex logic will just proceed as it has in previous versions, i.e., without a termination signal.

Parameters:

- **port_base** - the base address of the port to be opened. Must use one of the following defined macro names for this parameter (and #include "css_inc.h"):

  COM_1 which is equivalent to 0x3F8 (the base port address of port 0)
  COM_2 which is equivalent to 0x2F8 (the base port address of port 1)
  COM_3 which is equivalent to 0x3E8 (the base port address of port 2)
  COM_4 which is equivalent to 0x2E8 (the base port address of port 3)

- **baud** - the baud rate divisor for the port to be opened. Must use one of the following defined macro names for this parameter (and #include "css_inc.h"):

  SERIAL_BAUD_1200 which is equivalent to a baud rate divisor of 96
  SERIAL_BAUD_2400 which is equivalent to a baud rate divisor of 48
  SERIAL_BAUD_4800 which is equivalent to a baud rate divisor of 24
  SERIAL_BAUD_9600 which is equivalent to a baud rate divisor of 12

SERIAL_BAUD_19200 which is equivalent to a baud rate divisor of 6
SERIAL_BAUD_38400 which is equivalent to a baud rate divisor of 3
SERIAL_BAUD_57600 which is equivalent to a baud rate divisor of 2
SERIAL_BAUD_115200 which is equivalent to a baud rate divisor of 1

- **configuration** - set how the port should be configured (stop_bits | send_bits | parity). Must use the following defined macro names for this parameter (and #include "css_inc.h"):

  for the stop bits:
  SERIAL_STOP_1 - configure port for 1 stop bit per character
  SERIAL_STOP_2 - configure port for 2 stop bits per character

  for the send bits:
  SERIAL_BITS_5 - configure port for sending 5 bit characters
  SERIAL_BITS_6 - configure port for sending 6 bit characters
  SERIAL_BITS_7 - configure port for sending 7 bit characters
  SERIAL_BITS_8 - configure port for sending 8 bit characters

  for the parity:
  SERIAL_PARITY_NONE - configure port for no parity
  SERIAL_PARITY_ODD - configure port for odd parity
  SERIAL_PARITY_EVEN - configure port for even parity

- **message** - is the string that will be sent out the serial port when the trials are finished (or if you Escape out).

Returns: 1 if successful, otherwise 0 if an error.

Comments: This function is only available in the DOS 5.9.6 version. It is only necessary to call this function once per "Run:Start" of Cortex. Therefore, if you have logic in your timing file that is only executed on the first trial, you can put it there. If you don't have such logic, it is not a problem. Each time you call send_termination_signal() it will just overwrite the last parameters that were stored. In other words, if you call it the same way every time, it will just have the same parameters.

See also: Serial_Open(), Serial_Write(), Serial_Close()

Platform: DOS only

---

int Serial_Close ()

Purpose: Closes the serial port that was opened with the Serial_Open() function.

Parameters: none

Returns: 1 if successful, otherwise it returns 0.

Platform: DOS and Windows

---

void Serial_Flush ()

Purpose: Flushes out the serial buffer, for the serial port that was opened with the Serial_Open() function.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

int Serial_Open (int **port_base**, int **baud**, int **configuration**)

Purpose: Opens the specified serial port for communication. After the port has been successfully opened, the other Serial_*() functions can be used.

Parameters:

- port_base - the base address of the port to be opened. Must use one of the following defined macro names for this parameter (and #include "css_inc.h"):

Platform: DOS and Windows

---

void Serial_Print (pchar **string**)

Purpose: Write a string to the serial port that was opened with the Serial_Open() function. The string may contain standard C formatting codes, such as '\n'.

Parameters:

- **string** - the string to be printed.

Returns: nothing

Platform: DOS and Windows

---

int Serial_Read ()

Purpose: Reads a byte from the port that was opened with Serial_Open(), if there is a byte waiting in the buffer.

Parameters: none

Returns: the byte that was read, if successful; otherwise, it returns a -1

Platform: DOS and Windows

---

int Serial_Ready ()

Purpose: Checks the buffer of the serial port that was opened with Serial_Open() to see if there is a character waiting to be read.

Parameters: none

Returns: 1 if there are any characters waiting to be read, or a 0 if the buffer is empty.

Platform: DOS and Windows

---

int Serial_SetPortHardware (int **standard_port_addr**, int **new_port_addr**, int **new_IRQ**)

Purpose: changes the COM port's address to a non-standard IRQ and/or address. The **standard_port_addr** must be given, since the underlying GreenLeaf communications library functions require this as a parameter. In the timing file, the call to Serial_SetPortHardware() **MUST** be followed by a call to Serial_Open(), where the standard_port_addr is given as the

parameter for the "port_base".

Parameters:

- **standard_port_addr** - the standard address for the port (as defined in \cortex\source\include\serial.h)
  - standard_port_addr for COM1 is 0x3F8
  - standard_port_addr for COM2 is 0x2F8
  - standard_port_addr for COM3 is 0x3E8
  - standard_port_addr for COM4 is 0x2E8
- **new_port_addr** - the new address to be assigned to the port
- **new_IRQ** - the new IRQ to be assigned to the port

Returns: 1 if successful, otherwise 0.

Platform: DOS and Windows

---

void Serial_Write (char **ch**)

Purpose: Writes one byte to the port that was opened with Serial_Open().

Parameters:

- **ch** - byte to be written

Returns: nothing

Platform: DOS and Windows

---

void set_CLT_load_index (int **CLT_number**)

Purpose: sets the color lookup table index (default is 0). This function refers to which LUT will be used (CORTEX has the ability to load multiple LUTs at a single time) and the term "index" does not refer to a specific index within an LUT.

Parameters:

- **CLT_number** (0-255)

Returns: nothing.

See also: GcolorABS(), GcolorLUT(), GcolorREL(), ITEM_POSlut_index(), load_CLT(), set_colorABS(), set_colorREL()

Platform: DOS and Windows

---

void set_CODEbuf (int **index**, int **value**)

Purpose: sets the CODEbuf value at the given index.

Parameters: new value for CODEbuf at the given index

Returns: nothing

Platform: Windows only

---

void set_CODE_ISImax (int **value**)

Purpose: sets the current CODE_ISImax value.

Parameters: new value for CODE_ISImax

Returns: nothing

Platform: Windows only

---

void set_CODE_ISIoverflow (int **value**)

Purpose: sets the current CODE_ISIoverflow value.

Parameters: new value for CODE_ISIoverflow

Returns: nothing

Platform: Windows only

---

void set_CODE_ISIsize (int **value**)

Purpose: sets the current CODE_ISIsize value.

Parameters: new value for CODE_ISIsize

Returns: nothing

Platform: Windows only

---

void set_colorABS (int **index**, int **red**, int **green**, int **blue**)

Purpose: resets the color lookup table one color at a time by changing a single value within a color lookup table. To find out which index of the current color lookup table contains the color information for the desired item, call ITEM_POSlut_index(). This function operates immediately and does not wait for the screen refresh (unlike GcolorABS()).

Parameters:

- **index** - index within color LUT [use ITEM_POSlut_index()]
- **red** (0-255)
- **green** (0-255)
- **blue** (0-255)

Returns: nothing.

See also: GcolorABS(), GcolorLUT(), GcolorREL(), ITEM_POSlut_index(), load_CLT(), set_CLT_load_index(), set_colorREL()

Platform: DOS and Windows

---

void set_colorREL (int **index**, int **added_red**, int **added_green**, int **added_blue**)

Purpose: resets the color of an item one color lookup table index at a time by changing a single value within a color lookup table. Adds the values added_red, added_green, and added_blue to the current values for that entry in the color lookup table. To find out which index of the current color lookup table contains the color information for the desired item, call ITEM_POSlut_index(). This function operates immediately and does not wait for the screen refresh (unlike GcolorREL()).

Parameters:

- **index** - index within color LUT [use ITEM_POSlut_index()]
- **added_red** - offset from the current red value, can be positive or negative
- **added_green** - offset from the current green value, can be positive or negative
- **added_blue** - offset from the current blue value, can be positive or negative

See also: GcolorABS(), GcolorLUT(), GcolorREL(), ITEM_POSlut_index(), load_CLT(), set_CLT_load_index(), set_colorABS()

Platform: DOS and Windows

---

void set_EOGbuf (int **index**, int **value**)

Purpose: sets the EOGbuf value at the given index.

Parameters: new **value** for EOGbuf at the given index

Returns: nothing

Platform: Windows only

---

void set_EOGdynamic_fixwin_size (int **value**)

Purpose: sets the current params.dynamic_eyewin_size value.

Parameters: new **value** for params.dynamic_eyewin_size

Returns: nothing

Platform: Windows only

---

void set_EOGfixwin_size_x (float **value**)

Purpose: sets the current params.window_x value.

parameters: new **value** for params.window_x

Returns: nothing

Platform: Windows only

---

void set_EOGfixwin_size_y (float **value**)

Purpose: sets the current params.window_y value.

Parameters: new **value** for params.window_y

Returns: nothing

Platform: Windows only

---

void set_EOGgain (int **value**)

Purpose: sets the current params.eog_gain value.

Parameters: new **value** for params.eog_gain

Returns: nothing

Platform: Windows only

---

void set_EOGmax (int **value**)

Purpose: sets the current EOGmax value.

Parameters: new **value** for EOGmax

Returns: nothing

Platform: Windows only

---

void set_EOGnew_x (int **value**)

Purpose: sets the current EOGnew_x value.

Parameters: new **value** for EOGnew_x

Returns: nothing

Platform: Windows only

---

void set_EOGnew_y (int **value**)

Purpose: sets the current EOGnew_y value.

Parameters: new **value** for EOGnew_y

Returns: nothing

Platform: Windows only

---

void set_EOGoffset_x (int **value**)

Purpose: sets the current params.eog_xoffset value.

Parameters: new **value** for params.eog_xoffset

Returns: nothing

Platform: Windows only

---

void set_EOGoffset_y (int **value**)

Purpose: sets the current params.eog_yoffset value.

Parameters: new **value** for params.eog_yoffset

Returns: nothing

Platform: Windows only

---

void set_EOGoverflow (int **value**)

Purpose: sets the current EOGoverflow value.

Parameters: new **value** for EOGoverflow

Returns: nothing

Platform: Windows only

---

void set_EOGsaccade (float **value**)

Purpose: sets the current params.mc value.

Parameters: new **value** for params.mc

Returns: nothing

Platform: Windows only

---

void set_EOGsize (int **value**)

Purpose: sets the current EOGsize value.

Parameters: new **value** for EOGsize

Returns: nothing

Platform: Windows only

---

void set_EPPbuf (int **index**, int **value**)

Purpose: sets the EPPbuf value at the given index.

Parameters: new **value** for EPPbuf at the given index

Returns: nothing

Platform: Windows only

---

void set_EPPmax (int **value**)

Purpose: sets the current EPPmax value.

Parameters: new **value** for EPPmax

Returns: nothing

Platform: Windows only

---

void set_EPPnew_x (int **value**)

Purpose: sets the current EPPnew_x value.

Parameters: new **value** for EPPnew_x

Returns: nothing

Platform: Windows only

---

void set_EPPnew_y (int **value**)

Purpose: sets the current EPPnew_y value.

Parameters: new **value** for EPPnew_y

Returns: nothing

Platform: Windows only

---

void set_EPPoverflow (int **value**)

Purpose: sets the current EPPoverflow value.

Parameters: new **value** for EPPoverflow

Returns: nothing

Platform: Windows only

---

void set_EPPsize (int **value**)

Purpose: sets the current EPPsize value.

Parameters: new **value** for EPPsize

Returns: nothing

Platform: Windows only

---

void**Platform:** DOS and Windows

---

void set_fixwin_params (int **static_or_dynamic**, float **horizontal_extent**, float **vertical_extent**)

Purpose: The fixation window is the window which is compared to the eye_spot in the calls get_fixation_state(), get_saccade_state(), and ITEM_POSeye_ishere(BOUND_FIXWIN,...). The size of the fixation window can either be static (based on windowX, and windowY values found in the RUN:

PARAMETERS:GENERAL menu), or dynamic (set as the size of the fixation point item drawn on the screen). This function lets you specify dynamic vs. static nature of the fixation window with the **static_or_dynamic** argument, and sets the **horizontal_extent** and **vertical_extent** of the fixation window if static, which updates the windowX and windowY fields of the RUN:

PARAMETERS:GENERAL menu. Since updating the fixation window on the USER screen can take a long period of time (in terms of computing speed) and the **static_or_dynamic** toggle may need to be changed quickly, you may want to speed things up by bypassing this function altogether and setting the ExternVars EOGfixwin_size_x, EOGfixwin_size_y, and EOGdynamic_fixwin_size instead (which is a more direct but less organized way of adjusting the fixation window size).

Parameters:

- **static_or_dynamic** (static = 0, dynamic = 1)
- **horizontal_extent** (in degrees of visual angle. Only applies when static_or_dynamic = 0. Ignored otherwise)
- **vertical_extent** (in degrees of visual angle. Only applies when static_or_dynamic = 0. Ignored otherwise)

Returns: nothing

See also: get_fixation_state(), get_saccade_state(), ITEM_POSbind_fixspot(), ITEM_POSeye_ishere(), move_eye_window(), move_fixspot()

Platform: DOS and Windows

---

void set_ISIbuf (int **index**, long **value**)

Purpose: sets the ISIbuf value at the given index.

Parameters: new **value** for ISIbuf at the given index

Returns: nothing

Platform: Windows only

---

void set_keep_current_conds (int **value**)

Purpose: sets the current params.keep_current_conds value.

Parameters: new **value** for params.keep_current_conds

Returns: nothing

Platform: Windows only

---

void set_ms_reward_duration (int **value**)

Purpose: sets the current params.ms_reward_duration value.

Parameters: new **value** for params.ms_reward_duration

Returns: nothing

Platform: Windows only

---

void set_random_interval (int **minimum**, int **maximum**, int **step_interval**)

Purpose: randomly set timer to count-down from a number which is between **minimum** and **maximum** in steps of **step_interval**. For example, set_random_interval(100, 1000, 200) would set the timer for either 100, 300, 500, 700, or 900 milliseconds, randomly).

Parameters:

- **minimum** - minimum time, in milliseconds
- **maximum** - maximum time, in milliseconds
- **step_interval** - intervals between min and max, in milliseconds

Returns: nothing.

See also: MS_TIMERset(), set_random_timer(), set_timer()

Platform: DOS and Windows

---

void set_random_timer (int **maximum**)

Purpose: sets a random timer for timing some event, up to **maximum** number of milliseconds. Normally followed with get_timer()

Parameters: maximum time of random interval, in milliseconds.

Returns: nothing.

See also: MS_TIMERset(), set_random_interval(), set_timer()

Platform: DOS and Windows

---

void**Platform:** DOS and Windows

---

void**Platform:** DOS and Windows

---

void set_saccade_tolerance (long **min_ms_between**, float **x**, float **y**)

Purpose: Setup parameters for saccade testing.

Parameters:

- **min_ms_between** - minimum milliseconds between saccade tests. Set it to 0 for no checking.
- **x** - x tolerance in dva
- **y** - y tolerance in dva

Returns: nothing

Platform: DOS and Windows

---

void set_timer (int **time**)

Purpose: sets a timer to time milliseconds. Normally followed with get_timer();

Parameters: time, in milliseconds.

Returns: nothing.

See also: MS_TIMERset(), set_random_interval(), set_random_timer()

Platform: DOS and Windows

---

void set_TIMER100us_counter (long **value**)

Purpose: sets the current TIMER100us_counter value.

Parameters: new **value** for TIMER100us_counter

Returns: nothing

Platform: Windows only

---

void set_TIMERms_counter (long **value**)

Purpose: sets the current TIMERms_counter value.

Parameters: new **value** for the TIMERms_counter

Returns: nothing

Platform: Windows only

---

void**Platform:** DOS and Windows

---

float sin (float **value**)

Purpose: find the sine of a float value

Parameter:

- **value** Angle in radians

Returns: the sine of value.

See also: acos(), asin(), atan(), atan2(), cos(), cosh(), sinh(), tan(), tanh()

Platform: DOS and Windows

---

float sinh (float **value**)

Purpose: find the hyperbolic sine of value

Parameter:

- **value** Angle in radians

Returns: the hyperbolic sine of value.

See also: acos(), asin(), atan(), atan2(), cos(), cosh(), sin(), tan(), tanh()

Platform: DOS and Windows

---

int SMENUrun (int **yorg**, char ***select**, char **type**, char **bounds**, void ***item**, int **min**, int **max**)

Purpose: Prompts the user to enter a certain parameter.

Parameters:

- **yorg** - specifies the line on the screen where the text should appear
- **selec**t - string containing the message which will be displayed to the user
- **type** - type of the parameter to be input (must use the PMENU_ definitions from css_inc.h)
- **bounds** -specifies the bounds values that one may use (must use the PMENU_ definitions from css_inc.h)
- **item** - the variable that will hold the value to be set
- **min** -minimun length of the queried parameter
- **max** - maximum length of the queried parameter

Returns: whether or not the original value was changed by the user. If SMENUrun() returns a 0, the value has not been changed from its original setting. If SMENUrun() returns a 1, the value has been changed.

Platform: DOS only

---

int SOUNDload(int **i**, pchar **soundfile**)

Purpose: causes the sound file to be loaded as sound number i (0 - 255) and readied to play. This function is only available for two-computer Cortex.

Parameters:

- **i** - number to be associated with the sound file. Can be a value between 0 and 255.
- **soundfile** - file name of the sound file.

Platform: DOS and Windows

---

int SOUNDplay (int **i**, [int **looping**])

Purpose: plays the loaded sound file. If using the DirectX receive program, the **looping** parameter is used to specify whether to play the wave file once (0), or to keep playing it continously (1) until SOUNDstop() is called. The Scitech receive program can not use the looping parameter. This function is only available for two-computer Cortex.

Parameters:

- **i** - number that was associated with the file by calling SOUNDload(). Can be a value between 0 and 255.
- **looping** - parameter which specifies (in the DirectX receive version only) whether to play the .wav file once or to loop continuously until SOUNDstop() is called. 0 specifies to play the file once, 1 specifies to play the file continuously.

Returns: 1 if successful, 0 if not. Note: The SOUNDplay() call internally invokes the same code as SOUNDprep() and SOUNDstart(). Therefore, if you use SOUNDplay(), you do not need to call SOUNDprep() or SOUNDstart().

Platform: DOS and Windows

---

int SOUNDprep (int **i**)

Purpose: prepares sound number i to be started. This function is only available for two-computer Cortex.

Parameters: i - number that was associated with the file by calling SOUNDload(). Can be a value between 0 and 255.

Returns: 1 if successful, 0 if not. Note: The SOUNDplay() call internally invokes the same code as SOUNDprep() and SOUNDstart(). Therefore, if you use SOUNDplay(), you do not need to call SOUNDprep() or SOUNDstart().

Platform: DOS and Windows

---

int SOUNDstart(int **i**, int **looping**)

Purpose: starts a sound i that has already been prepared with

SOUNDprep(). If using the DirectX receive program, the looping parameter is used to specify whether to play the wave file once (0), or to keep playing it continously (1) until SOUNDstop() is called. The Scitech receive program can not use the looping parameter. This function is only available for two-computer Cortex.

Parameters:

- **i** - number that was associated with the file by calling SOUNDload(). Can be a value between 0 and 255.
- **looping** - parameter which specifies (in the DirectX receive version only) whether to play the .wav file once or to loop continuously until SOUNDstop() is called. 0 specifies to play the file once, 1 specifies to play the file continuously.

Returns: 1 if succesful, 0 if not. Note: The SOUNDplay() call internally invokes the same code as SOUNDprep() and SOUNDstart(). Therefore, if you use SOUNDplay(), you do not need to call SOUNDprep() or SOUNDstart().

Platform: DOS and Windows

---

void SOUNDstop (int **i**)

Purpose: stops the sound from playing. For the Scitech version, only one sound can be played at a time, so you do not need to specify the sound number. For the DirectX version, multiple sounds can be played at the same time, so you must specify the number that is to be stopped. . This function is only available for two-computer Cortex.

Parameters:

- **i** - number that was associated with the file by calling SOUNDload(). Can be a value between 0 and 255. (Only used in this function with the DirectX version of Cortex.)

Returns: nothing.

Platform: DOS and Windows

---

void SOUNDvol (int **left**, int **right**, int **i**)

Purpose: sets the left and right mixer volume in each speaker (valid values, 0-31). For the Scitech version, this call sets the parameters for all sounds. For the DirectX version, the volume can be set for each sound. Therefore, you must specify the sound number i. . This function is only available for two-computer Cortex.

Parameters:

- **left** - mixer volume for the left speaker. Valid values are

0 - 31, where 0 is low volume and 31 is high volume.
- **right** - mixer volume for the right speaker. Valid values are 0 - 31, where is low volume and 31 is high volume.
- **i** - number that was associated with the file by calling SOUNDload(). Can be a value between 0 and 255. (Only used in this function with the DirectX version of Cortex.)

Returns: nothing

Platform: DOS and Windows

---

int sprintf (pchar **string**)

Purpose: formats and stores a series of characters and values in **string**, into **buffer**. Formatting in the string (ie. '\n') behaves as in ANSI C printf() calls. A null character is added to the end of the characters written, but not counted in the return value. See a standard C library reference for further help.

Parameters:

- **buffer** - Storage location for output
- **string** - format control string and arguments

Returns: the number of bytes stored in **string**, not counting the terminating null character.

See also: printf()

Platform: DOS and Windows

---

float sqrt (float **value**)

Returns: the square root of **value**

See Also: pow()

Platform: DOS and Windows

---

void srand (unsigned int **seed**)

Purpose: Sets the seed value for the rand() function. rand() and srand() are the standard C function for generating a pseudo-random number.

Parameter:

- **seed** - Seed for random-number generation

Example:
```
long current;
srand( time( & current ) );
```

Returns: nothing

See also: rand()

Platform: DOS and Windows

---

void srand2 (unsigned int **seed**)

Purpose: Sets the seed value for the random() and rand2() functions. rand2() and srand2() use a different algorithm than rand() and srand(), and may provide better pseudo-random numbers than those obtained from rand() and srand().

Parameter:

- **seed** - Seed for random-number generation

Example:

```
long current;
srand2( time( & current ) );
```

Returns: nothing

See also: random(), rand2()

Platform: DOS and Windows

---

int sscanf (pchar **buffer**, pchar **string**)

Purpose: reads data from **buffer** into the location given by each argument in **string**. See a standard C library reference for further help.

Parameters:

- **buffer** - holds the string of data
- **string** - format control string and arguments

Returns: the number of fields successfully converted and assigned A return value of 0 indicates that no fields were assigned.

Platform: DOS and Windows

---

void start_trial (int **bin_width**)

Purpose: initializes the spike input flip flops and sets the **bin_width** (in ms) for the on-line cumulative histogram display (divide length of trial by 256 bins).

Parameters: binwidth, in milliseconds.

Returns: nothing.

See also: end_trial()

Platform: DOS and Windows

---

pchar strcat (pchar **string1**, pchar **string2**)

Purpose: appends **string2** to **string1** and terminates the resultant string with a Null character ('\0').

Parameters:

- **string1** Null-terminated destination string
- **string2** Null-terminated source string

Returns: pointer to the concatenated string

See Also: strchr(), strcmp(), strcpy()

Platform: DOS and Windows

---

pchar strchr (pchar **string**, int **character**)

Purpose: Find a character in a string.

Parameters:

- **string** Null-terminated source string
- **character** Character to be located

Returns: pointer to the character within string

See Also: strcat(), strcmp(), strcpy(), strchr(), strstr()

Platform: DOS and Windows

---

int strcmp (pchar **string1**, pchar **string2**)

Purpose: compares string2 to string1 lexicographically.

Parameters:

- string1, string2 Null-terminated strings to compare

Returns: <0 if **string1** < **string2**
0 if **string1** = **string2**
0 if **string1** **string2**

See also: strcat(), strcmp(), strcpy(), strchr()

Platform: DOS and Windows

---

pchar strcpy (pchar **string1**, pchar **string2**)

Purpose: copies **string2** to **string1**. Enough space for **string1** must have been previously allocated (unlike strdup()).

Parameters:

- string1 Destination string
- string2 Null-terminated source string

Returns: returns **string1**

See also: strcat(), strcmp(), strchr(), strdup()

Platform: DOS and Windows

---

pchar strdup (pchar **string**)

Purpose: allocates storage space for a duplicate of **string** and returns it a pointer to the storage space

Parameters: Null-terminated source string

Returns: returns a pointer to the new storage space filled with a duplicate of string

See also: strcat(), strcmp(), strcpy(), strchr()

Platform: DOS and Windows

---

int strlen (pchar **string**)

Purpose: Get the length of a **string**.

Parameter:

- **string** Null-terminated string

Returns: the length of string

See also: strstr()

Platform: DOS and Windows

---

pchar strncat (pchar **strDest**, pchar **strSource**, int **count**)

Purpose: Append characters of a string.

Parameters:

- **strDest** - Null-terminated destination string
- **strSource** - Null-terminated source string
- **count** - Number of characters to append

Returns: a pointer to the destination string.

Platform: DOS and Windows

---

int strncmp (pchar **string1**, pchar **string2**, int **count**)

Purpose: Compare characters of two strings.

Parameters:

- **string1**, **string2** - Strings to compare
- **count** - Number of characters to compare

Returns: The return value indicates the relation of the substrings of string1 and string2 as follows.

- <0 if string1 substring less than string2 substring
- 0 if string1 substring identical to string2 substring
- >0 if string1 substring greater than string2 substring

Platform: DOS and Windows

---

pchar strncpy (pchar **strDest**, pchar **strSource**, int **count**)

Purpose: Copy characters of one string to another.

Parameters:

- **strDest** - Destination string
- **strSource** - Source string
- **count** - Number of characters to be copied

Returns: the destination string, strDest.

Platform: DOS and Windows

---

pchar strpbrk (pchar **string**, pchar **strCharSet**)

Purpose: Scan strings for characters in specified character sets.

Parameters:

- **string** - Null-terminated, searched string
- **strCharSet** - Null-terminated character set

Returns: a pointer to the first occurrence of any character from strCharSet in string, or a NULL pointer if the two string arguments have no characters in common.

Platform: DOS and Windows

---

pchar strrchr (pchar **string**, int **c**)

Purpose: Scan a string for the last occurrence of a character.

Parameters:

- **string** - Null-terminated string to search
- **c** - Character to be located

Returns: a pointer to the last occurrence of c in string, or NULL if c is not found.

Platform: DOS and Windows

---

int strspn (pchar **string**, pchar **strCharSet**)

Purpose: Find the first substring.

Parameters:

- **string** - Null-terminated string to search
- **strCharSet** - Null-terminated character set

Returns: an integer value specifying the length of the substring in string that consists entirely of characters in strCharSet. If string begins with a character not in strCharSet, the function returns 0.

Platform: DOS and Windows

---

pchar strstr (pchar **string1**, pchar **string2**)

Purpose: Find a substring.

Parameters:

- **string1** - Null-terminated string to search
- **string2** - Null-terminated string to search for

Returns: a pointer to the first occurrence of **string2** in **string1** or NULL if string2 does not appear in string1.

See also: strcat(), strcmp(), strcpy(), strchr()

Platform: DOS and Windows

---

pchar strtok (pchar **strToken**, pchar **strDelimit**)

Purpose: Find the next token in a string.

Parameters:

- **strToken** - String containing token(s)
- **strDelimit** - Set of delimiter characters

Returns: a pointer to the next token found in strToken. It returns NULL when no more tokens are found. Each call modifies strToken by substituting a NULL character for each delimiter that is encountered.

Platform: DOS and Windows

---

int sweep_win (int **test_screen**)

Purpose: sweeps a test_screen for the initialized (using init_sweep()) time, direction of motion, and speed. Unlike the pan function, the center of the sweep will be at the window location and the start of the sweep will be appropriate for the direction of motion. Because this call requires an initial call to init_sweep(), it has been replaced by Gsweep().

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY;

#include "css_inc.h")

Returns: time remaining (in milliseconds; a multiple of the refresh rate). Returns 0 when sweep done.

See also: Gsweep(), init_sweep(), sweep_win_with_fix()

Platform: DOS and Windows

---

int sweep_win_with_fix (int **test_screen**)

Purpose: sweeps a test_screen for the initialized (using init_sweep()) time, direction of motion, and speed. Unlike the pan function, the center of the sweep will be at the window location and the start of the sweep will be appropriate for the direction of motion. Fixspot tracks scroll (but need not be centered on same point).

Parameters:

- **test_screen** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")

Returns: time remaining (in milliseconds; a multiple of the refresh rate). Returns 0 when sweep done.

See also: Gsweep(), init_sweep(), sweep_win()

Platform: DOS and Windows

---

int system (pchar **DOS_command**)

Purpose: executes **DOS_command** as an operating system command

Parameter:

- command Command to be executed

Returns: 0 if successful

Platform: DOS and Windows

---

float tan (float **value**)

Purpose: find the tangent of a value

Parameter:

- x Angle in radians

Returns: tangent of **value**

See also: acos(), asin(), atan(), atan2(), cos(), cosh(), sin(), sinh(), tanh()

Platform: DOS and Windows

---

float tanh (float **value**)

Purpose: Calculate the hyperbolic tangent (tanh).

Parameter:

- x Angle in radians

Returns: the hyperbolic tangent of **value**.

See also: acos(), asin(), atan(), atan2(), cos(), cosh(), sin(), sinh(), tan()

Platform: DOS and Windows

---

long tell (int **handle**)

Purpose: reports the current position of the file pointer within the file referred to by **handle**

Parameter:

- handle Handle referring to open file

Returns: current position of the file pointer from the beginning of the file in bytes

See also: lseek()

Platform: DOS and Windows

---

long THREADadd (int **class**, int **ms_tiks**, long **max_count**, void (***fn**)(void))

Purpose: Add a user-defined thread.

Parameters:

- **class** - type of thread (refer to the THREAD_ definitions in \source\css.h)
- **ms_tiks** - number of milliseconds between executions of the thread
- **max_count** - the number of times for the thread to run. Zero means that it should run forever. Any other number specifies the number of times it should run before it is deleted
- **fn** - function to be called when the thread executes

Returns: The id of the thread

Platform: DOS and Windows

---

int THREADdel (long **id**)

Purpose: Delete a user-defined thread.

Parameters:

- **id** - thread id of the thread to be deleted. The thread id is returned by the THREADadd function.

Returns: 1 if successful, 0 if unsuccessful

Platform: DOS and Windows

---

int THREADrun (int **class**)

Purpose: Run a user-defined thread.

Parameters:

- **class** - type of thread (refer to the THREAD_ definitions in \source\css.h)

Returns: The number of threads executed of this class.

Platform: DOS and Windows

---

int THREADstart_seqs ()

Purpose: Runs threads of all the different types.

Parameters: none

Returns: The number of threads executed.

Platform: DOS and Windows

---

void THREADstop_seqs ()

Purpose: Stops and removes all interrupt threads. Internally, calls TIMERpurge().

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

long time (plong **timer**)

Purpose: Returns the number of seconds elapsed since midnight (00:00:00), January 1, 1970, according to the system clock. The return value is stored in the location given by **timer**. This function is useful for providing a value for seeding the random number function.

Parameter:

- **timer** - the time

Returns: The time in elapsed seconds. There is no error return.

Platform: DOS and Windows

---

int TIMERadd (int **ms_tiks**, long **max_count**, void (***fn**)(**void**))

Purpose: Add a thread of type TIMER_THREAD.

Parameters:

- **ms_tiks** - number of milliseconds between executions of the thread
- **max_count** - the number of times for the thread to run. Zero means that it should run forever. Any other number specifies the number of times it should run before it is deleted
- **fn** - function to be called when the thread executes

Returns: The id of the thread, if successful. Otherwise, it returns a 0.

Platform: DOS and Windows

---

int TIMERaddCSSfn (int **ms_tiks**, long **max_count**, int **css_start**)

Purpose: Add a thread of type TIMER_THREAD containing a Cortex system function.

Parameters:

- **ms_tiks** - number of milliseconds between executions of the thread
- **max_count** - the number of times for the thread to run. Zero means that it should run forever. Any other number specifies the number of times it should run before it is deleted
- **css_start** - CSSFN starting index of the function to be run

Returns: The id of the thread, if successful. Otherwise, it returns a 0.

Platform: DOS and Windows

---

int TIMERaddINT (int **ms_tiks**, long **max_count**, void (***thread**)(**void**))

Purpose: Add a thread of type TIMER_INT08.

Parameters:

- **ms_tiks** - number of milliseconds between executions of the thread

- **max_count** - the number of times for the thread to run. Zero means that it should run forever. Any other number specifies the number of times it should run before it is deleted
- **thread** - function to be called when the thread executes

Returns: The id of the thread, if successful. Otherwise, it returns a 0.

Platform: DOS and Windows

---

int TIMERchange_rate (int **id**, int **new_ms_tiks**)

Purpose: Change the rate of the thread.

Parameters:

- **id** - thread id
- **new_ms_tiks** - new number of milliseconds between executions of the thread

Returns: 1 if successful, 0 otherwise.

Platform: DOS and Windows

---

int TIMERdel (int **id**)

Purpose: Delete a user-defined thread.

Parameters:

- **id** - id of the thread to be deleted

Returns: 1 if successful, otherwise 0.

Platform: DOS and Windows

---

long TIMERget_count (int **id**)

Purpose: Returns the number of times that a particular thread is actually run.

Parameters:

- **id** - thread id

Returns: The number of times that a thread is run.

Platform: DOS and Windows

---

long TIMERget_ms_count ()

Purpose: Returns the current trial time which is held in the global variable, TIMERms_counter

Parameters: none

Returns: The value of TIMERms_counter.

Platform: DOS and Windows

---

void TIMERpurge ()

Purpose: Kills all timer threads, but doesn't free malloced space for them.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

int TIMERstart_clock (int **speed_class**)

Purpose: Starts the interrupt timer running.

Parameters:

- **speed_class** - speed class of the timer

Returns: 1 if successful, 0 otherwise

Platform: DOS and Windows

---

void TIMERstop_clock ()

Purpose: Stops the interrupt timer.

Parameters: none

Returns: nothing

Platform: DOS and Windows

---

int timer_expired ()

Purpose: Function returns whether or not the timer has expired. The timer must have previously been set by set_timer. The typical use is: while(!timer_expired()) { do something }. Note: It is only during the timer_expired call that the play routine is activated

Parameters: none.

Returns: whether or not (0 = no; 1 = yes) the timer has expired. The timer must have previously been set by the set_timer() family of routines.

See also: set_random_interval(), set_random_timer(), set_timer()

Platform: DOS and Windows

---

int toggle_wins (int **test_screen_1**, int **test_screen_2**)

Purpose: toggles between two test_screens

Parameters:

- **test_screen_1** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")
- **test_screen_2** (TEST0,TEST1,...,FIXSPOT,PLAY; #include "css_inc.h")

Returns: time remaining (in milliseconds; a multiple of the refresh rate). Returns 0 when done, and turns off both stimuli, waiting until they are actually off before returning.

Platform: DOS and Windows

---

int touch_item (int **test**, int **position**, float **radius**)

Purpose: compare the location of touch screen coordinate to the location of an item on the screen; if the difference is less than the radius specified by the user, return TRUE, else return FALSE.

Parameters:

- **test** - test# (0-9,FIXSPOT,PLAY,EYE_WIN,BOUND_FIXWIN)
- **position** - position within that test (1-x)
- **radius** - distance for comparison

Returns: 1 if difference in location is less than radius (i.e., the touch was within the correct item); 0 if the touch was not in the correct item location; -1 if there was an error or if no touch occurred.

Platform: DOS and Windows

---

void update_histogram ()

Purpose: update the histogram with the current trial's data. Typically called at the end of the trial.

Parameters: none.

Returns: nothing.

See Also: display_histogram(), display_trial_progress()

Platform: DOS and Windows

---

int write (int **handle**, pchar **buffer**, int **count**)

Purpose: writes data to a currently open file.

Parameters:

- **handle** (a currently open file)
- **buffer** (data to be written to file)
- **count** (number of bytes to be written)

Returns: the number of bytes actually written

See also: open(), read()

Platform: DOS and Windows

---

int _stricmp (pchar **string1**, pchar **string2**)

Purpose: Perform a lowercase comparison of strings.

Parameters:

- **string1**, **string2** Null-terminated strings to compare

Returns: indicates the relation of string1 to string2 as follows.

- <0 if string1 less than string2
- 0 if string1 identical to string2
- >0 if string1 greater than string2

Platform: DOS and Windows

Backward compatibility

Several functions that were available in CORTEX version 4 have been replaced in version 5 with upgrades. In some cases we have added new and better functions (but have not removed the old ones) and in these cases we felt that the old functions were so frequently used that removing them would cause hardship for the user, regardless of an upgrade option. This section of the CORTEX Timing File Reference Manual is devoted only to guiding the user to the new function in instances where the old function has been removed.

Old Functions New Functions display_eye_buf() display_eye_path() EYEactivate_eyewin() ITEM_POSbind_fixspot() EYEactivate_item() ITEM_POSbind_fixspot() EYEis_at_eyewin() ITEM_POSeye_ishere() EYEis_at_item() ITEM_POSeye_ishere() fget_X() EYEget_dva() fget_Y() EYEget_dva() get_fixation_posX() EYEget_dva() or ITEM_POSeye_delta() get_fixation_posY() EYEget_dva() or ITEM_POSeye_delta() get_posX() ITEM_POSget() get_posY() ITEM_POSget() mark_eyewin() ITEM_POSmark_pos() mark_item() ITEM_POSmark_pos() mark_pos() ITEM_POSmark_pos() mark_screen_pos() DrawBox() set_block_pick_type() BLOCKset_control_info() set_cond_pick_type() BLOCKset_control_info() set_position() EYE_WINset() set_trial_duration() not replaced

Back to top