



Script reference manual

Eye and Pen version 3.0

(May 2018)

"IF ALL ELSE FAILS, READ THE MANUAL "

© D. Chesnet & D. Alamargot, 2004-2012

© D. Chesnet, 2012-2018

Maison des Sciences de l'Homme et de la Société (MSHS) de Poitiers
Centre de Recherches sur la Cognition et l'Apprentissage (CeRCA)

The CNRS and the University of Poitiers own the rights to the "Eye and Pen" software
(IDDN.FR.001.470020.000.S.C.2004.000.31235)

A Script-based acquisition: introduction	6
Reminder : script-based acquisition dialog box.....	6
Script commands: some rules and definition	7
Relative file paths	8
Release notes	9
Script commands, by category	10
Files and directories	10
SetPicsDirectory	10
SetDataDirectory	10
SetModelsDirectory	10
Recording	11
SetSafeRec	11
OpenRec	11
CloseRec	12
Log.....	12
AddToLog	12
OpenMyLog	12
AddToMyLog	13
CloseMyLog	13
Text file.....	13
CreateTextFile	13
AddToTextFile	13
CloseTextFile	14
CopyFileTo	14
Labels	15
:label	15
ResetLabelCounter	16
SetLabelCounter	16
AddToLabelCounter	17
SubtractFromLabelCounter	17
List, items and elements	18
AddToList	18
LoadList	18
ResetList	18
RandomizeList	18
RandomizeListRange	19
AddElement	19
GetElements	19
Screen Display	20
SetDisplayWindowCoord	20
SetBackgroundColor	21
SetFont	21
SaveScreenToBMP	22
SaveScreenAreaToBMP	22
Masking.....	23
SetUnmaskFile	23
SetMaskingMode	23
SetMaskingFillPic	24
SetMaskingFillColor	24
SetMaskingBlurLevel	24
ActivateFeedbackMasking	24
DefineUnmaskZone	25
ClearUnmaskZones	25

SetUnmaskZonesRange	25
SetUnmaskZonesBackTracking	26
MustWriteToActivateUnmaskZone	26
Stimuli	27
PlayModel	27
Message	28
DisplayMsg	28
HideMessage	29
Text.....	29
DisplayText	29
HideText	30
Picture	30
DisplayPic	30
HidePicture	30
DisplayImageList	31
PreloadPics	31
UnloadPics	32
SetLightSensorPosition	32
DisplayPicWithSensor	32
Video	33
DisplayAVI	33
StopAVI	34
Audio	34
Beep	34
SystemBeep	34
PlaySound	35
StopSound	35
SetVolume	35
Wait.....	36
WaitFor	36
WaitForKeyPress	36
CountDown	36
Jumps	36
JumpTo	36
JumpToIfKeyPressIs	37
JumpToIfTextIs	38
JumpToIfGroupIs	39
JumpToIfNumberIs	40
JumpToIfLabelIs	40
LoopIfLabelIsBelow	41
Jumps in the script triggered by a tablet zone.....	42
DefineTabZone	42
WaitForTabZones	42
WaitForPenOut	43
ClearZones	45
Eye tracker	45
SetCalibrationCoord	45
TestCalibration	45
TestDrift	45
Writing display	46
SetPenColor	46
SetPenWidth	46
Time shift writing display	46
SetTabTimeShiftDelay	46
ActivateTabTimeShift	47
Transform writing display	47
SetTabOffset	47
SetTabRatio	47

Write until.....	48
WaitForTabZoneAt	48
WriteUntilKeypress	49
MaxWritingDuration	49
Using the “Simple” acquisition mode	50
RecStandard	50
RecNewUsages	50
RecNewPics	51
RecNewPics&Usages	52
SetPicsZones	53
RestoreOriginalPicsZones	53
RecNewAll	53
ShowSimpleTrace	55
SetRecStandardMaxDisplay	55
NetSync	56
WaitForNetSync	56
SendMessageToNetSync	56
Network messaging.....	56
UseNetMessaging	56
SendNetMessage	57
I/O	57
UseParallelPort	57
SendToParallelPort	57
SetParallelPortLines	57
WaitForParallelPortValue	58
Keywords.....	58
%EValue%	58
%ECount%	58
%G%	59
%I%	59
%K%	59
%L%	59
%LCount%	60
%PFolder%	60
%RValue%	60
%S%	61
%?:Label%	61
Command menu tree view	62
Script examples	64
1. Have a task re-done	64
2. Mask production (Simple).....	65
3. Mask production with unmasking locations	66
4. Load a list and show its content	67
5. Randomize sub-blocks of a list	68
6. Use two label counters with one shifted.....	68
7. Have a list of words copied, with NetSync.....	69
8. Dictation of a text	70
9. Copying along with a mental load	70

A SCRIPT-BASED ACQUISITION: INTRODUCTION

Sometimes (often), easy to use (“one-click”) build-in tools doesn’t help to perform an experiment. More flexibility is needed.

The Eye and Pen way is to offer scripting capabilities. Scripting means telling Eye and Pen what it has to do, with commands (“orders”). Yes, this is programming. But with a limited number of words.

Thus: acquisition based on script relies on a mini programming language.

The good news is that every command can be added to script with the help of dialog boxes (assistant) where you just have to set values in fields, or select value in list, etc.. It has a syntax coloring feature.

Script editor usage is described with more details in the User Manual.

This volume will be dealing with commands and their options.

REMINDER : SCRIPT-BASED ACQUISITION DIALOG BOX

(File/Acquisition/Script menu)

When launching the script-based acquisition mode, a “start” dialog box is displayed on the screen.

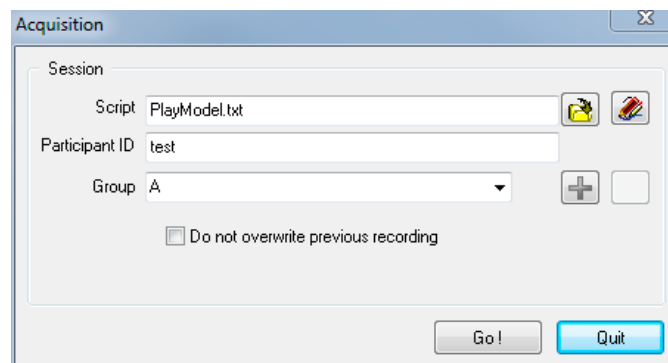




Figure 1: Script-based acquisition dialog box.

Two fields of this dialog box are of particular interest to script:

LABEL	DESCRIPTION
Script	Displays the name of the current script.
	Allows you to select the script you want to use. Caution: a script file must be saved in "text-only" file format (".txt") if it is to be understood by Eye and Pen. By default, the last used script is proposed.
	Modifies the current script or create a new one in the script editor.

Participant ID	Choose a name for the data file(s).
Group	Optionally, select a group to which the participant belongs to, in the scrolling list. This selection may be later retrieved from a script. This list may be modified by hand.
Do not overwrite previous recording	This option allows you to avoid losing data from a previous recording if you mistakenly choose the same name. If this option is ticked and a data file with the same name already exists in the same directory, a warning will be displayed (recording is canceled).
Go !	Launches acquisition. First, the script is checked against major defects (syntax faults, missing files), then it is executed.
Quit	Closes the acquisition dialog box and returns to the main Eye and Pen screen.

SCRIPT COMMANDS: SOME RULES AND DEFINITION

The pseudo language is made up of a list of **commands**.

These commands have to be written in a **text-only file “.txt”** (Windows ANSI coding).

A set of commands in a text file is called a script.

A script follows a number of rules, listed below.

There must be only one command per line. No spaces (blanks) are allowed inside a command.

In a script, a line may contain three sorts of items:

- a **command**. It tells Eye and Pen to perform some action.
- a **comment**: defined by a semi-colon at the beginning of the line and followed by text (spaces allowed). A comment has no action. It is just text.
- a **label**: defined by “:” followed by a single word. It is a location “mark” inside a script.

Example:

```

Command1
; my comment is here      ← A comment
command2
:Tag1                     ← A label
Command3

```

Script uses two kinds of commands:

- commands comprising a single word.

Example: Command1

- commands comprising a command word followed by parameters between brackets, separated by a comma.

Example: `Command1 (parameter1, parameter2)`


When using the second type of command, replace the parameter definitions (between brackets) with their values.

A few rules:

- A single command, comment or label to a line.
- The script analyzer isn't "case sensitive", i.e. upper- and lower-case letters are the same to it.
- Since comma is used to separate parameters, if you wish to insert a comma into a text, first insert a backslash before the comma ("\\,"). This way, it won't be interpreted as a separator.

About coordinates:

X1 and Y1, X2 and Y2 are the coordinates of the two opposite corners of a rectangular area (e.g. a screen, a picture, a zone on a tablet, etc.)

LABEL	DESCRIPTION
	X1 and Y1 coordinates correspond to the upper left-hand corner of the rectangle. X2 and Y2 coordinates correspond to the lower right-hand corner of the rectangle.
X1	Coordinate of the left edge of the rectangle.
Y1	Coordinate of the upper edge of the rectangle.
X2	Coordinate of the right edge of the rectangle
Y2	Coordinate of the bottom edge of the rectangle.

RELATIVE FILE PATHS

In a number of script commands, you will have to mention filenames.

By default, the software will look for pictures in Eye and Pen's "stimuli" folder (or in the folder you selected in your configuration or redefined through a script command).

However, for reasons of your own, you may want to designate a file that is not in the default folder. There is a solution: the relative path.

An example:

```
SetUnmaskFile(..\cursors\Mask_oval_asym_right170.bmp)
```

Explanation

Folder structure:

Eyeandpen

|-> Stimuli

|-> Cursor

The picture file is located not in the Stimuli folder but in the Cursor folder. Accordingly, the relative path will start from the Stimuli folder:

1. “ .. “ allows you to move up to the next level in the folder tree, i.e. to leave the "Stimuli" folder and move to the "Eyeandpen" folder.
2. The first “ \ “ allows you to move down to the "Cursor" subfolder.
3. The next “ \ “ allows you to designate the picture file contained in the "Cursor" folder.

Relative paths from a default folder can be used anywhere that a file has to be loaded from a default folder.

Note: in this manual, the terms “folder” and “directory” mean exactly the same thing.

RELEASE NOTES

A few commands have been deprecated because they were compound commands:

- WaitForKeyPressMsg
- WaitForKeyPressText
- WaitForKeyPressPic

They will not be explained nor proposed in the script editor, although they're still supported during script execution.

Please note that the keyword **%M%** (which give the number of items in the list) **is deprecated** and replaced by **%LCount%** (stands for “List Count”, which is more explicit).

SCRIPT COMMANDS, BY CATEGORY

The script commands are presented and explained below, divided into different function categories.

Files and directories

Redefines the stimuli default directory (pictures, texts, sounds, videos)

SetPicsDirectory (DirectoryPath)

This command redefines the directory of stimuli used in the scripts, initially defined in the “Script” tab of the acquisition configuration panel.

Replace “**DirectoryPath**” with the path to the directory containing your stimuli.

Example

`SetPicsDirectory(c:\mypictures\)` gives the “mypictures” directory on the “C” drive of the PC as the new default stimuli directory.

Redefines the default data files directory

SetDataDirectory (DirectoryPath)

This command redefines the directory in which data files will be recorded (initially defined in the “Script” tab of the acquisition configuration panel).

Replace “**DirectoryPath**” with the path to the directory you want to use.

Example

The command `SetDataDirectory(c:\mydata\)` gives the “mydata” directory on the “C” drive of the PC as the new data recording saving directory.

Redefines the default model files directory

SetModelsDirectory (DirectoryPath)

This command redefines the directory of models files used in the scripts, initially defined in the “Script” tab of the acquisition configuration panel.

Replace “**DirectoryPath**” with the path to the directory containing your models.

Example

The command `SetModelsDirectory(c:\myModels\)` gives the “myModels” directory on the “C” drive of the PC as the new default models directory.

Recording

Can a participant's existing data file be overwritten?

SetSafeRec (DontErase)

This command redefines the “*No recording overwrite*” option in the dialog box.

Replace “*DontErase*” by:

- TRUE: no recording allowed if a file already exists.
- FALSE: allow data overwrite if one already exists.

Example

If you insert the command `SetSafeRec(False)` into a script, after this command, there will be no further overwrite checks.

If, a few lines further, you insert the command `SetSafeRec(True)`, **from that line onwards**, data will not be erased and it will be impossible to record them with the same name.

Hint: deactivate this checking for the training trials (to limit the number of useless files) and reactivate it for the following experimental trials.

Opens tablet and/or eye-tracking data recording file

OpenRec (AddToParticipantName)

This command opens one/two file(s) to record the participant's data: one for the tablet data and one for the eye tracker data, if an eye tracker is used.

The “*AddToParticipantName*” parameter allows you to define a suffix that will be added to the data filename (defined in the acquisition start dialog box).

Caution: each file opened with “*OpenRec*” must be closed with “*CloseRec*” (see next command description). If not, you may lose data.

Hint: to avoid overwriting files if you are using a loop (e.g. when the participant has to perform the same task repeatedly), add “*%I%*” at the end of “*AddToParticipantName*”. This special option will be replaced by the number of times the script has read the last label (for a definition, p. 15).

Example 1

For a participant called “Toto” (you have defined his name in the start dialog box), the command `OpenRec(_series1)` will record tablet data in the file named “*toto_series1.tab*”.

Example 2

You have used the command `OpenRec(_series1_%I%)`. Participant Toto has passed the same label in the script twice. You will therefore end up with two recording files: “*toto_series1_1.tab*” and “*toto_series1_2.tab*”.

Closes the opened data recording file(s)

CloseRec

This command closes the current participant's recording data file(s) (tablet “.TAB” file and optional eye tracker “.EYE”)

Log

Logging is a convenient way to store information in memory. When the log is closed, it is written to a text file on disk.

Eye and Pen creates and manages a default log for each recording session (in which each information is tagged with the session timer), which is named from the participant Id.

A custom log may be explicitly created (only one at a time), filled and saved. It is possible to issue successive custom logs.

Add a text to the “default” recording session log

AddToLog (TextMessage)

This command allows to record your own text into the log that Eye and Pen manages during a recording session.

Replace “*TextMessage*” by the text you want to be appended.

Example

If you insert the command `AddToLog(Here we made a pause)` into a script, the sentence “Here we made a pause” will be inserted into the session log file, timestamped with the time elapsed since the beginning of the session.

Creates a custom log

OpenMyLog (AddToParticipantName)

This command allows to create your own custom log.

The “*AddToParticipantName*” parameter allows you to define a suffix that will be added to the data filename (defined in the acquisition start dialog box).

Example

For a participant called “Toto” (you have defined his name in the start dialog box), the command `OpenMyLog(_series1)` will finally record your own log content in the text file named “*toto_series1.MyLog*”.

If you set a log name that already exists, the log file will be overwritten.

OpenMyLog works like OpenRec: the file will be saved in the data directory.

Calling OpenMyLog when a custom log is already opened immediately closes the current custom log and opens the new one.

Add a text to a custom log

AddToMyLog (TimerMode, TextMessage)

This command allows to add information to your own custom log (for this command to work, the log must have been first created with OpenMyLog).

“**TimerMode**” allows to select a timing source to timestamp the information that will be added to the log. Value can be:

- 0 = no timing
- 1 = current recording timer
- 2 = session timer

Replace “**TextMessage**” by the “sentence” you want to add to your custom log. For example, this may be an item name, the light detector activation, participant’s writing start, etc.

Example: `AddToMyLog(1, item number %I%)`

Saves a custom log to file

CloseMyLog

This command closes the custom log file (previously opened with OpenMyLog) and write its content from memory to a text file.

Text file

Creates a text file

CreateTextFile (Filename)

This command opens a text file in which you may add text.

Replace “**Filename**” by the name of the text file your want to open. The file will be named <Filename>.TXT and created in the Data folder.

If the file already exists, new information will added at its end, the file not be overwritten.

Add some text to a text file

AddToTextFile (TextMessage)

This command allows to save your own text into a text file (previously opened with OpenTextFile).

Replace “**TextMessage**” by the text you want to be appended. It is immediately written to file.

Please remember that keywords inserted in text are interpreted (replaced by their value), such as participant Id, Group name, Label, etc.

You may use this command to save information created during an experiment, for later re-use, for example.

Close a text file

CloseTextFile

Closes the currently opened text file.

Copy a file to another location

CopyFileTo (Source, Destination)

This command allows to save your own text into a text file (previously opened with OpenTextFile).

Replace “**Source**” by the full path name of the file you wish to copy.

Replace “**Destination**” by the full path name of the file you wish to “receive”. The file name may be different from the source, thus allowing to “rename” the file.

Example

c:\data\toto_capture.jpg may be copied to c:\stimuli\pictureA.jpg

Note

Jokers may be used, such as “*.*”, “*.bmp”, etc.

Location-related keywords may also be use (see Keywords, p. 58), such as %PIMAGES% (Images directory path).

This command may be use for a range of purposes, including copying a screen capture from the Data to the Stimuli folder, to later re-use it as a screen background, for example.

Defines a label

:label

A label identifies a particular place in the script to which you will later refer or for which you will establish a link. Strictly speaking, this is not a command, but a mark in the script. For this reason, each label in the script must be unique.

The “label” notion can be likened to the notion of “bookmark” used in a number of common applications (word processor, etc.) or even with the “anchor” notion used in HTML (Web) documents.

Different “jump” commands (see below) allow the script to discontinue its sequential execution (one line after the other) and go straight to a particular label and then continue executing commands from this location (see other commands in this section).

When “jumping”, commands located between the jump command and the destination label are ignored.

Replace the term “*label*” with a word of your choice.

You may define as many labels as you like.

Notion of “counter”

Each label has an internal counter.

The first time a label is “read” by the script interpreter, it sets the label counter to 1. Each time the interpreter reads this label again (like a command, i.e. the label is on a line of its own) for example, because of a jump command, its counter value is increased by 1 (it will then be set at 2, and the next time at 3, etc.).

A label counter’s value represents the number of iterations (the action of repeating same processes).

Operations

Some specific commands (see below) allow you to explicitly modify a counter’s value. Please remember that performing operations on counter values may lead to negative values, which are not suitable for indexing a list, for example.

Syntax

You can insert spaces in front of the labels or commands (indentation) in order to make your script easier to read.

Example

```
Command1
Command2
    :label
        Command3
        Command4
etc.
```

Hints:

1. a label can be used as a convenient way of memorizing a value while executing a script. In classic programming languages, this is done with a “variable” when the value is likely to change, and with a “constant” when the value will not be modified.
2. to check a label counter’s value, you can use the DisplayMsg command. For example, to display the value of the “Label1” label counter, you can use `DisplayMsg(Label1 value is: %I:Label1%,2000,-1,-1,TRUE)`

Resets a label counter to zero

ResetLabelCounter (Label)

This command allows you to initialize or re-initialize a label’s iteration counter. Replace the word “**Label**” with the name of the label you wish to reset (minus “:”).

Example

```
:Loop
  ResetLabelCounter (label1)
:Label1
  JumpToIfNumberIs (5,Next, FALSE)
  Command3
  JumpTo (Label1, FALSE)
:Next
  Command4
```

When the script reaches the command `JumpTo (Label1, FALSE)`, it will jump to the “:Label1” label.

At the fifth reading of the command `JumpToIfNumberIs (5, Suite, FALSE)` this command will be “activated”. The script will then execute it and jump to the “:Next” label. `Command3` and `JumpTo (Label1, FALSE)` will not be executed. The script will continue from the “:Next” label, and read and execute `Command4`.

In case `Command4` includes a statement to jump to the “:Loop” label, `Label1`’s counter should be reset to allow the script to execute the commands following `JumpToIfNumberIs` a further 4 times.

For example, if we imagine a setup with two zones on the tablet, if the participant presses the pen in the first zone the script continues, whereas if the participant activates the second zone, the script jumps to “Loop”.

Example of use:

Allows a participant to start a task over again.

Sets the value of a label’s counter

SetLabelCounter (Label, Value)

This command allows you to define the value of a label’s internal counter.

Replace “**Label**” with the name of the label (minus “:”) whose counter value is to be modified.

Replace “**Value**” with the number you wish to assign to this label’s counter.

“*Value*” may also be replaced with another label’s counter value (see “Keywords”, p. 58).

Example

`SetLabelCounter (Label1, 4)` assigns the value 4 to “*Label1*”

Note:

`SetLabelCounter (Label1, 0)` is equivalent to
`ResetLabelCounter (Label1)`

Examples of use

- ❖ When used in conjunction with `JumpToIfNumberIs`, it allows you to change the script blocks that are to be executed, depending on what the participant does;
 - ❖ Allows you to start a task again, but with a different number of trials;
 - ❖ Allows you to define a loop using only part of a list, for example only using items 8 to 15 (see commands and keywords for handling a list of items).
-

Adds to the value of a label’s counter

AddToLabelCounter (Label, Value)

This command allows you to add a value to the value of a label’s internal counter.

Replace “***Label***” with the name of the label (minus “:”) whose counter value is to be modified.

Replace “***Value***” with the number you wish to add to this label’s counter. “*Value*” can also be replaced with another label’s counter value (see “Keywords”, p. 58).

Example

`AddToLabelCounter (Label1, 4)`

Subtracts from the value of a label’s counter

SubtractFromLabelCounter (Label, Value)

This command allows you to subtract a value from the value of a label’s internal counter.

Replace “***Label***” with the name of the label (minus “:”) whose counter value is to be modified.

Replace “***Value***” with the number you wish to subtract from this label’s counter. “*Value*” can also be replaced with another label’s counter value (see “Keywords”, p. 58).

Example

`SubtractFromLabelCounter (Label1, 4)`

List, items and elements

Using a list of items may help either to simplify the process of writing a script or, conversely, to develop a sophisticated experiment.

An Eye and Pen script automatically has a single list.

Each item in the list may be a word, a sentence, a number, a filename, etc. In short, anything that can be written using letters and numbers. Items are numbered from 1.

Adds an item to the list

AddToList(ItemName)

This command allows you to add an item to the list. This new item is appended to the end of the list. The list is not sorted in any specific way: items appear in the order in which they are added.

Replace “**ItemName**” with the item that is to be added to the list: a string of letters and/or numbers (word, sentence, 500, etc.)

Examples

```
AddToList(This is a sentence to read)
AddToList(1200)
AddToList(image3.wmf)
AddToList(sound.wav)
```

Loads an item list from a file

LoadList(FileName)

This command allows you to load a set of items contained in a text file. Its content is then included in the list.

Items are appended to the end of the list. Loading from a file does not delete items previously included in the list.

Replace “**FileName**” with the name of the file to be loaded.

The file must be in a “.TXT” text-only format and should contain one item per line. The file must be located either in the *Stimuli* folder or else at the end of a relative path leading from this folder (cf. p. 8).

Example:

```
LoadList(liste.txt)
```

Clears the list

ResetList

This command empties the list of its contents: all items are deleted.

Shuffles item order in the list

RandomizeList

This command allows you to shuffle the items around in the list.

Shuffles item order in part of the list

RandomizeListRange (Begin, End)

This command allows you to shuffle items in a random order in just part of the list.

Replace “**Begin**” with the number of the first item in the part of the list that is to be shuffled, i.e. at the beginning of the set to be randomized.

Replace “**End**” with the number of the last item in the set to be randomized.

Note: position numbering in the list starts with 1.

Example

The command `RandomizeListRange(3,9)` randomizes all items from positions 3 to 9 (inclusive).

Example of use

To be able to use a list of items within a test session divided in sub-blocks, the items in each one being randomized. For example, while the list’s first items (practice) are not randomized, two successive blocks of items are randomized, all in a single list. This makes it easier to write a script because the experimental session can simply run from item 1 to the end of the list.

ELEMENTS

An item of the list may be a compound of elements such as words, for example. The following commands aim at allowing to manipulate these elements.

Adds an element to a list item

AddElement (ListIndex, Element)

This command allows you to add an element to an item of the list. The element will be appended to the end of the item at position given in the list.

Replace “**ListIndex**” by the order number in the list of the item to modify.

Replace “**Element**” by the alphanumerical string you want to add to the item. If an element contains then # character between “words”, they may be later retrieved as separated items.

Examples

```
AddElement(14, 4 words as one sentence)
```

```
AddElement(46, #pseudoword)
```

```
AddElement(7, #probe#target)
```

Splits a text into elements

GetElements (Text)

This command allows you to split a text into elements, if they are separated by the # character.

Replace “**Text**” by the sentence you want to split.

Once a sentence has been split into elements, keywords allows to individually access these elements.

`%E<index>%` allows to retrieve the element at position `<Index>` in the text. For example `%E32%` retrieves then 32th element of the text processed.

`%ECOUNT%` returns the number of elements that have been split.

Keywords may be mixed as well, using label counter as index, as in `%E:BlocStart%`, for example.

The following example shows how to iterate through a string made of words, using `%ECount%` and `%E<Value>%`, Value being a label counter’s value:

```
GetElements(This is # a # string #of text# made #of# elements)
:MyCounter
  DisplayMessage(%E:MyCounter%,-1,-1,-1,FALSE)
  WaitForKeypress
  LoopIfLabelIsBelow(StringPos, %ECount%)
```

Screen Display

Resolutions and sizes

When using a masking technique, if you intend to use pictures as a background/wallpaper (e.g. using the *DisplayPic* command), we strongly advise that you adjust the screen resolution to match the picture size. For example, if you are using pictures measuring 1024*768, switch the screen resolution to these dimensions (menu ?/System Information, click on “Screen”) and display your pictures at coordinates (0,0).

It should be borne in mind that the pen’s position on the tablet is mapped onto the display (screen) size.

A different frame and coordinate system for the screen may make some of the participant’s writing look distorted.

Thus, a screen configured with a 1280*1024 pixel resolution, using a display/writing zone (*Configuration/Script* menu) of 1024*768 pixels and displaying a centered picture of 800*600 pixels will produce a “shrinking” effect. Obviously, if this what you are aiming for, forget everything that has just been said!

Redefines the coordinates of the display area on the screen

SetDisplayWindowCoord (x1 , y1 , x2 , y2)

This command redefines the coordinates of the display area on the screen (initially defined in the “Script” tab of the acquisition configuration panel).

Replace “**X1**”, “**Y1**”, “**X2**” and “**Y2**” with the coordinates (in pixels) of the new display

area you want to use.

Example

`SetDisplayWindowCoord(0,0,1024,768)` defines a display area on the screen of 1024 x 768 pixels, beginning in the upper left-hand corner.

Hint: If you have two 1024 x 768 resolution screens, the command `SetdisplayWindowCoord(1024,0,2048,768)` will move the display area to the second monitor. For this to work properly, ensure that Windows desktop extends to both monitors. However, note that this configuration can lead to problems with certain LCD tablet drivers.

Redefines the screen's background color

SetBackgroundColor(ColorValue)

This command redefines the screen's background color (initially defined in the "Script" tab of the acquisition configuration panel).

Replace "**ColorValue**" with the number of the color you want.

Color numbers follow the Web standard and range from **#000000 (black)** to **#FFFFFF (white)**.

Example

As a result of the command `SetBackgroundColor(#000000)`, the screen background color will be black.

Redefines the characteristics of messages or texts displayed on the screen

SetFont(FontName, FontSize, FontColor, BkgndColor)

This command redefines the characteristics of the font and the background of the text displayed on screen (initially defined in the "Script" tab of the acquisition configuration panel).

Replace the parameters (described below) with the appropriate values.

LABEL	DESCRIPTION
FontName	Name of the font used. Respect upper/lowercase letters and spaces in the names.
FontSize	Font size (in points, same unit as in word processors).
FontColor	Font color (color number)
BkgndColor	Background color for text display (color number)

Example

After the command `SetFont(Comic sans MS,14,#000000,#FFFFFF)` has been read by the Eye and Pen script interpreter, the text will be displayed:

- in black (*BkgndColor* = "#000000");
- in the Comic sans MS font (*FontName* = "Comic sans MS");

- in 14 points (*FontSize = "14"*);
 - against a white background (*BkgndColor = "#FFFFFF"*).
-

Saves the screen content in an image file

SaveScreenToBMP (PictureFileName)

This command converts the screen into a picture file (BMP format).

Replace "**PictureFileName**" with the name of the picture.

Note: the picture is saved in the *Stimuli* folder, thus allowing you to display it again through a script command.

Examples

1. the command `SaveScreenToBmp (Copy1)` captures the screen content in the file named "*Copy1.bmp*"
 2. for a participant named "*Toto*", the command `SaveScreenToBmp (%S%_Capture)` will save the screen in a file named "*Toto_Capture.bmp*" (the keyword `%S%` is replaced by the participant's Id).
-

Saves a screen area content in an image file

SaveScreenAreaToBMP (PictureFileName, X1, Y1, X2, Y2)

This command converts the screen into a picture file (BMP format).

Replace "**PictureFileName**" with the name of the picture.

Note: the picture is saved in the *Stimuli* folder, thus allowing you to display it again through a script command.

Replace "**X1**", "**Y1**", "**X2**" and "**Y2**" with the coordinates (in pixels) of the part of the display you want to save.

Example:

```
SaveScreenAreaToBMP (Capture1, 100, 100, 800, 600)
```

Captures the screen area from 100 to 800 pixels horizontally, and 100 to 600 pixels vertically, content in the file named "*Capture1.bmp*".

Masking

Masking/unmasking production can take two forms: either the “unmasked area” follows the pen location or it remains at (a) fixed location(s) on the screen that is unmasked when the pen reaches that point. Only one technique can be used at a time. If you define unmasking zones (areas), this mode will always take precedence over the “floating” version.

Designates the file to unmask the display

SetUnmaskFile (MaskFileName)

This command allows you to select a picture file that will be used to unmask an area around the pen location. In other words, this is a “peephole” file that helps writers to “see through” the mask covering the screen display.

An “unmasking” file is a two-color (black and white) BMP format picture, where the white area represents the transparent area. Several samples are provided in the “Cursors” folder (the number included in the filename is the “whole” size, expressed in screen pixels). The file is always centered on the pen tip’s location.



Figure: A file used to unmask the display.

Replace “**MaskFileName**” with the filename you wish to use.

Example

The following command identifies the file named “Oval_asym_right170.bmp” as the unmasking file:

```
SetUnmaskFile(..\cursors\Oval_asym_right170.bmp)
```

Note:

The picture file is located not in the *Stimuli* folder, but in the *Cursors* folder. This means that a relative path starting from the *Stimuli* folder has to be used (for more information about relative paths, see p. 8).

Defines how the screen will be masked

SetMaskingMode (Mode)

This command allows you to determine which kind of mask is to be applied to the screen display, out of a possible 3:

- 1 (COLOR): the display is filled with a uniform color
- 2 (PICTURE): the display is concealed by a picture
- 3 (BLUR): the display content is blurred.

Replace “**Mode**” with the mode number you have selected.

Example: `SetMaskingMode(3)` sets the screen masking mode to “blur”

Note:

If you write your script with the help of the script editor, you will be able to select the mode you require from a scrolling list. Its number will automatically be inserted into the resulting command.

Also see the warning about masking.

Selects the picture that will mask the display**SetMaskingFillPic (PictureFileName)**

This command allows you to select the picture file that will cover the screen display (see User manual for a list of supported picture file formats).

Replace “**PicturFileName**” with the picture filename you want.

Example:

```
SetMaskingFillPic (MaskFile.jpg)
```

Also see the warning about masking, p. **Erreur ! Signet non défini..**

Selects a color to mask the display content**SetMaskingFillColor (ColorNumber)**

This command allows you to select the color that will fill the screen display.

Replace “**ColorNumber**” with the number of the color you have selected.

Note: if you are using the script editor, you will select the color from a color map.

Example

The following command determines the color that is to fill the screen (i.e. orange):

```
SetMaskingFillColor (#FF8000)
```

Sets blur intensity to mask the display**SetMaskingBlurLevel (Intensity)**

This command allows you to adjust the intensity of the blurring that will be applied to the screen content.

Replace “**Intensity**” with the intensity level you have selected (from 1 to 100).

Because the blurring of the trace left by the pen is dynamically calculated, we advise you to choose a value between 3 and 25. Values above 25 considerably increase the calculation time and hence the time needed to refresh the display.

Example

The following command sets the blurring level at 12:

```
SetMaskingBlurLevel (12)
```

Activates/deactivates display masking**ActivateFeedbackMasking (Active)**

This command allows you to turn the screen display masking feature on or off for the following commands: `WaitForTabZoneAt`, `WaitForTabZones` and `RecStandard`.

Replace “**Active**” with TRUE to activate the display masking feature and with FALSE to switch it off.

Example:

```
ActivateFeedbackMasking (TRUE)
```

Defines an unmasking zone

DefineUnmaskZone (x, y)

This command allows you to define an unmasking zone, i.e. a fixed location on the screen that will be unmasked when the pen reaches it.

“**X**” and “**Y**” represent the upper left-hand corner of the area that is to be unmasked (in pixels). The horizontal and vertical sizes of the area are implicit; they are those of the unmasking picture.

Zones have an order: they are processed in their order of creation.

Important

If unmasking zones are created, they take over the “free” unmasking mechanism. Thus, the unmasking will not freely follow the pen position but will only be effective in some areas of the screen, i.e. the unmasking zones.

Example

The following command creates an unmasking zone at screen coordinates x=100 and y=236 : `DefineUnmaskZone (100, 236)`

Example of use

Create a series of unmasking zones, each one corresponding to a place where the participant is expected to write words. A zone only becomes visible (unmasked) for the time it takes for a word to be written in it. As soon as the pen leaves the zone, it is masked again.

The following commands allow you to manage and fine tune the zones’ behavior.

Deletes all unmasking zones

ClearUnmaskZones

This command deletes all the unmasking zones that were previously defined.

Defines neighboring zones that are to be unmasked at the same time

SetUnmaskZonesRange (LowLimit, HighLimit)

This command defines the boundaries of the zones that are activated (unmasked) at the same time as the zone under the pen tip.

Replace “**LowLimit**” with the preceding zone’s number (in zone creation order, see `DefineUnmaskZone` command) and “**HighLimit**” with the number of the next zone,

relative to the active zone.

The active zone (under the pen tip) is always numbered 0.

The zone preceding the active zone (in order of creation) is numbered -1, the one before that is numbered -2, and so on.

Following the same principle, the zone after the active zone is numbered 1.

There is no limit to the number of previous and subsequent zones that can be activated (unmasked) at the same time.

Example

The following command means that each time a zone is unmasked (with the pen), the previous zone (in order of creation) is also unmasked:

```
SetUnmaskZonesRange (-1, 0)
```

Note:

The default range is (0,0): only the activated-by-pen zone is unmasked.

Allows you to go back to a previously unmasked zone

SetUnmaskZonesBackTracking (Allowed)

This command determines whether or not it is possible to go back, i.e. to unmask a zone that has previously been unmasked (in zone creation order).

Replace “**Allowed**” with TRUE to enable you to go back or with FALSE to disallow it.

Example

In the following example, if the participant un.masks first Zone 1, then Zone 2, he or she cannot then go back to Zone 1 and unmask it again:

```
SetUnmaskZonesBackTracking (FALSE)
```

Note:

The default value is TRUE: it is possible to go back.

Determines whether the pen must be pressed in a zone to unmask it

MustWriteToActivateUnmaskZone (MustWrite)

This command allows you to determine whether the participant has to press the pen tip in a zone to unmask it.

This command is specific to zone unmasking.

Replace “**MustWrite**” with TRUE to make the pen pressure mandatory, or with FALSE so that the pen’s presence above a zone is enough to unmask it.

Note:

This value is TRUE by default: the pen must be pressed in a zone for the latter to be unmasked.

Example: MustWriteToActivateUnmaskZone (TRUE)

Stimuli

Displays an animated drawing/writing

PlayModel (*ModelName*, *PauseBefore*, *TimeRatio*, *WithWatermark*,
IsCentered)

This command allows to have a handwriting/drawing sequence replayed from a data file.

This file can be created from Eye and Pen recording and/or modified by hand (see User manual), or even created from a calculation, etc.

This command is **display only**, it is not possible to write at the same time.

LABEL	DESCRIPTION
<i>ModelName</i>	Name of the file (.TXT) that contains the writing:drawing to replay. This file should be found in the <i>Model</i> directory (initially defined in the “Script” tab of the acquisition configuration panel).
<i>PauseBefore</i>	Duration (in milliseconds) of a pause before beginning the replay. Pen cursor is already displayed at the starting point. Duration has two “magic” values: <ul style="list-style-type: none">• 0: make no pause• -2: wait for a key press (instead of time).
<i>TimeRatio</i>	Percentage of the normal speed for replay. 100 means normal speed, less is slow down, more is speed up.
<i>WithWatermark</i>	Should the final product be displayed as a watermark (its color is defined in the configuration panel, <i>Analysis</i> tab) ? If yes, replace <i>WithWatermark</i> by TRUE, else set FALSE.
<i>IsCentered</i>	Should the model be centered on screen ? If yes, replace <i>WithWatermark</i> by TRUE, else set FALSE.

Example

```
PlayModel(3copies.txt,2000,50,TRUE,TRUE)
```

Will play “3copies.txt” after a 2 second pause, at 50% of normal speed. It will be shown as a watermark (so visible before replay starts) and centered on display.

About Model files

A model file is a text file because it allows to build one’s tracing at will, either from an Eye and Pen recording (exported to a Model file after filtering and resampling, for example) or event build from multiple tracing as a “mean” tracing.

A “model” file may be modified, provided it is saved as an ANSI text file (MS-DOS

like). For example, in Microsoft Excel, you may save as Text (separator: tabulation) (*.txt).

The file content is build as follows:

- Sampling rate, in Hz (correct values are between 10 and 1000)
For example: 200.
- Horizontal and vertical resolution of tablet (typically 1000 or 2000 lines / cm). For example: 2000 2000
- Coordinates range of the tablet: origin and horizontal and vertical extend.
For example: 0 0 45630 22480
- Samples, made of X coordinate, Y coordinate, Pressure value.

Model file content example:

```
200
2000      2000
0         0         44500      22300
0         0         0
11667     20524     190
11682     20548     286
11699     20575     294
11718     20606     349
11737     20639     447
11757     20674     537
...

```

Message

Displays a message for a certain length of time

DisplayMsg (Message, Duration, X, Y, Transparent)

This command is used to display a message for a user-defined duration, at the indicated screen coordinates. The script remains “on hold” until the display time has elapsed. The message disappears when the command comes to an end.

To set the content of the message, replace “**Message**” with the text you want to display.

Caution: do not include “,” in your message, as it is a parameter separator.

To set the message display duration, replace “**Duration**” with the desired amount of time, expressed in **milliseconds**.

To set the message's position in the display windows, replace “**X**” with the horizontal coordinate (in pixels) of the message's first character and “**Y**” with its vertical coordinate.

To set the message's horizontal and vertical coordinates, you can:

-
- proceed by trial and error;
 - use a relation including the screen's physical size and resolution.

Hint:

- to center the message in one/both dimension(s), set the corresponding coordinate to “-1”;
- to leave the message on the screen "forever", set duration to “-1”. The script will resume immediately, without erasing the message.

Replace “**Transparent**” with FALSE if you want the message to be displayed on a background stripe colored with the current font background color or with TRUE to have only the characters of the message displayed on the screen.

Note:

When transparency is set to TRUE, the display font is always black, this is a known bug of Windows.

Example

With the command `DisplayMsg(Hello world, 500, 120, 600, TRUE)`, the message “Hello world” will be displayed for 500 milliseconds.

Erases the message left on the screen

HideMessage

This command removes the message previously displayed with `DisplayMsg` (see above).

Text

Displays a text file for a certain length of time

DisplayText (*TextFileName*, *Duration*)

This command is used to display the content of a text file for a certain length of time. The script remains until the display time has elapsed. The text disappears when the command comes to an end.

Replace “**TextFileName**” with the name of the **text file** you placed in the stimuli directory. Do not forget to add the “.txt” extension to the filename.

Replace “**Duration**” with the **duration** (in milliseconds) of the text display.

Hint: if you want the text to remain "forever" on the screen, set the duration to “-1”. The script will immediately resume, without erasing the text.

Example

With the `DisplayText(mytext.txt, 500)` command, the text contained in the text file named “*mytext.txt*” (found in the stimuli directory) will be displayed for 500 milliseconds.

Erases the text on screen

HideText

This command removes the text displayed on the screen as a result of the “DisplayText” command.

Picture

Displays a picture for a certain length of time

DisplayPic (PictureFileName, Duration, X, Y)

This command is used to display a picture (found in the stimuli directory) for a certain length of time. The picture stays on the screen until the display time has elapsed. The picture disappears when the command comes to an end.

Replace “**PictureFileName**” with the name of the **picture** (do not forget the format extension). The picture may have been preloaded into memory with the *PreloadPics* command.

Replace “**Duration**” with **the length of time** (in milliseconds) you want.

To set the picture's position in the display windows, replace “**X**” with the horizontal coordinate (in pixels) and “**Y**” with the vertical coordinate of the picture's upper left-hand corner.

To set the picture's horizontal and vertical coordinates, you can:

- proceed by trial and error;
- use a relation including the screen's physical size and resolution.

Hint:

- to center the picture in one/both dimension(s), set the corresponding coordinate to “-1”.
- to make the picture remain on the screen "forever", set the duration to “-1”. The script will resume immediately, without removing the picture.

Example

With the `DisplayPic(icon.bmp, 500, 120, 120)` command, the picture “*icon.bmp*” will be displayed for 500 milliseconds at the coordinates $X=120$, $Y=120$.

Erases the picture on the screen

HidePicture

This command removes the picture from the screen (previously displayed using the `DisplayPic`, `DisplayPicWithSensor` or `DisplayImageList` command). Background is restored.

Displays a series of pictures one after another

DisplayImageList (**ListFileName**, **X**, **Y**, **DurationPerPicture**,
HideLastPic)

This command displays pictures one after another (at the specified coordinates) for a specified length of time.

Replace “**ListFileName**” with a text filename (TXT format) containing the picture list (see supported pictures formats in the User manual) to be displayed (one picture name per line). Save this file and the pictures in the stimuli directory.

Caution: all pictures must be the same size. The first picture to be displayed will define the size of all the others.

Example:

The file `MyList.txt` contains:

Eye.bmp

And.bmp

Pen.bmp

To set the picture's position in the display windows, replace “X” with the horizontal coordinate (in pixels) of the picture's upper left-hand corner and “Y” with its vertical coordinate.

To set the picture's horizontal and vertical coordinates, you can:

- proceed by trial and error;
- use a relation including the screen's physical size and resolution.

Hint: to center the picture in one/both dimension(s), set the corresponding coordinate to “-1”.

Replace “**DurationPerPicture**” with the amount of time (in milliseconds) you want each picture to be displayed for.

Replace “**HideLastPic**” with TRUE to remove the last picture from the screen or FALSE to leave it.

Hint: if you use the same background color for all the pictures, you will reduce the visual transition effect between the pictures.

Example

With the `DisplayImageList (Mylist.txt, 120, 120, 250, FALSE)` command, the pictures listed in the “`mylist.txt`” file will be displayed one after the other. Each picture will be displayed at the coordinates X=120 and Y=120 for 250 milliseconds and the final picture will remain on the screen.

Loads pictures into memory

PreloadPics (**PictureFileName**, **Feedback**)

This command allows to load an image file (or many image files) into memory.

Replace “**PictureFileName**” by the name of the **picture** (do not forget the format

extension) This file should be found in the *Stimuli* directory (initially defined in the “*Script*” tab of the acquisition configuration panel). The name can a path with jokers (*, ?, etc.), relative to the *Stimuli* folder. For example “.BMP” is used to load all BMP files.

Replace “**Feedback**” by *TRUE* to display the number of pictures while loading into memory. This option is especially useful while creating a script.

Note:

Loading duration and memory consumption can be huge. For example, 312 pictures in a 1024*768 resolution will eat 600Mo of RAM.

Resizing pictures to their useful area, may help. The 24 bits format (16 billion colors) is processed more quickly.

Calling `PreloadPics` does not unload previously loaded pictures, but adds new picture to existing ones. So, one can load pictures from different folders, or from different types (BMP, JPG, etc.).

Erases pictures from memory

UnloadPics

Unloads pictures (previously loaded with `PreloadPics`) from memory.

Defines the location and area of light sensor on screen

SetLightSensorPosition(x1,y1,x2,y2)

This command allows to set the location and area required by a light sensor. This area can be selected with the mouse on the screen, using the script editor Command menu.

Replace “**X1**”, “**Y1**”, “**X2**” and “**Y2**” with the coordinates (in pixels) of the area of the display covered by the light sensor.

Displays a picture on screen

DisplayPicWithSensor(PictureFileName,Duration,X,Y,Color)

This command works the same as `DisplayPic`, except that a color patch is displayed at the light sensor location.

This command is used to display a picture for a certain length of time. The picture stays on the screen until the display time has elapsed. The picture disappears when the command comes to an end.

LABEL	DESCRIPTION
PictureFileName	Replace “ <i>PictureFileName</i> ” with the name of the picture (do not forget the format extension) This file should be found in the <i>Stimuli</i> directory (initially defined in the “ <i>Script</i> ” tab of the acquisition configuration panel).
Duration	Replace “ <i>Duration</i> ” with the length of time (in milliseconds) you want.
X, Y	Sets the picture's position in the display windows. Replace “ <i>X</i> ” with the horizontal coordinate (in pixels) and “ <i>Y</i> ” with the vertical coordinate of the picture's upper left-

	hand corner.
Color	Color of the patch that will be displayed at the light sensor location: 0 for black, 1 for white.

Hint:

- to center the picture in one/both dimension(s), set the corresponding coordinate to “-1”.
- to make the picture remain on the screen "forever", set the duration to “-1”. The script will resume immediately, without removing the picture.

Example

`DisplayPicWithSensor(icon.bmp, 500, 120, 120, 1)` command, the picture “*icon.bmp*” will be displayed for 500 milliseconds at the coordinates $X=120, Y=120$, with a white patch under the light sensor (then activating it).

Video

Displays a video file

DisplayAVI (VideoFileName, X, Y, Wait)

This command displays a video (found in the stimuli directory) at specified coordinates (relative to the display window). The video disappears when it is finished.

Replace “**VideoFileName**” with the video file's full name (including its “.AVI” extension).

Caution:

Eye and Pen can only read “.avi” video format, preferably with MS-RLE compression. To check whether the video file complies:

- In Windows, select the video file and, with a right mouse button click, select the Summary tab in Properties;
- Look in “*Video*” for the “*Compression*” label to check the video's compression mode. It should be either "none" or "MS-RLE".

To set the video's position in the display windows, replace “**X**” with the horizontal coordinate (in pixels) of the video’s upper left-hand corner and “**Y**” with its vertical coordinate.

To set the video's horizontal and vertical coordinates, you can:

- proceed by trial and error;
- use a relation including the screen’s physical size and resolution.

Hint:

To center the video in one/both dimension(s), set the corresponding coordinate to “-1”.

The “**Wait**” parameter determines whether the script should wait for the video to finish before continuing or not. This parameter can be given two values:

- “**FALSE**”: the video starts to play and the script immediately resumes;
- “**TRUE**”: the film starts and the script goes on hold. No script commands are executed while the video is being played.

Example

With the command `DisplayAVI(MyVideo.avi,-1-1,TRUE,)` the “*Myvideo.avi*” video is played in the center of the display window, because “*X*” and “*Y*” both have a “-1” value. The script resumes as soon as the video is finished.

Stops the video playing

StopAVI

This command stops the AVI video currently being played.

Audio

Plays a beep

Beep

This command plays the Windows basic sound defined in the Windows Control Panel (under Windows XP “Sounds and audio devices”, “Sounds” tab), “default beep” entry.

Makes the system generate a beep

SystemBeep (Frequency,Duration)

This command makes the system generate a beep of the frequency and duration you want.

Replace “**Frequency**” with the frequency value of your choice (in Hz) and “**Duration**” with the required duration (in milliseconds).

Caution: this beep is played on through the computer’s internal speaker; it uses neither the sound card nor the audio, headset or speaker output of the sound device (card).

Example

The command `SystemBeep(1000,200)` plays a continuous 1,000 Hz frequency sound for 200 milliseconds.

Plays an audio file

PlaySound (WaveFileName, Wait)

This command plays an audio file (found in the stimuli directory) in “.wav” format. Replace “**WaveFileName**” with the filename (do not forget to add “.wav” to the name).

The “**Wait**” parameter determines whether the sound is to be played as a background task or not.

This parameter can be given two values:

- “**FALSE**”: playing begins and the script resumes immediately;
- “**TRUE**”: playing begins and the script goes on hold. No other script command is executed until the audio sequence has finished.

Caution:

If you use this command twice (with no other command between the two), a "system" malfunction may occur.

This is because “Eye and Pen” uses a media player which takes some time to close, before being able to reopen with another file. It is therefore advisable to create a single sound file containing several sound sequences (e.g. as a series of numbers).

Example

The command `PlaySound(ding.wav, FALSE)` plays the “*ding.wav*” sound file and the script continues (the “*Wait*” parameter is “*FALSE*”). The script's next command is immediately executed.

Example of use: a dictation.

Stops the audio play

StopSound

This command stops the audio file currently being played.

Changes the sound output level

SetVolume (value)

This command allows you to change the sound level for the audio output.

Replace “**value**” with a number between 0 (mute) and 65535 (maximum) to set the sound level.

Wait

Pauses the script for a certain duration

WaitFor (duration)

This command stops the script execution for “**duration**” milliseconds. When the delay is elapsed, the script execution goes on again.

Example

The command `WaitFor(2000)` stops the script for 2000 milliseconds, i.e. 2 seconds. When this delay ends, the script continues.

Stops the script. It starts again when a keyboard key is pressed

WaitForKeyPress

This command puts the script on hold. As soon as a key is pressed, the script execution resumes.

Pauses the script for a certain duration, displaying a countdown

CountDown (Duration)

This command stops the script execution for “**Duration**” seconds. A count down is displayed at the center of the display area.

When the delay is elapsed, the script execution goes on again.

Jumps

The mechanism of **jumping into/through the script** allows you to eliminate the sequential aspect of the script (one command after another, in the right order) and create, for example, loops (repeating the same action) or interactive sequences.

This enables a participant to repeat the same task many times (with a maximum number) or to carry out different tasks, depending on the zone (area) in which the pen is pressed.

Jumps to a label in the script

JumpTo (Label, MustCloseRec)

This command “jumps” to the relevant label (see definition p. 15) and, if specified in the parameters, closes the recording file.


All commands between this command and the relevant label will be ignored and the script will resume after the label.

Replace “**label**” with the name of the label you want the script to jump to (minus “:”).

Replace “**MustCloseRec**” with “TRUE”, if you want to close an open data recording file, or “FALSE” if you do not need to.

Example

```
Command2
JumpTo (Zone1Go, FALSE)
Command3
:Zone1go
command5
```



When the script interpreter reads the command `JumpTo (Zone1Go, FALSE)`, it will jump to “*Zone1go*”. Commands between `JumpTo (Zone1Go, FALSE)` and the destination label will not be executed (`command3`). The script will continue and execute `command4`. If the “*MustCloseRec*” parameter had been given the “*TRUE*” value, the participant's recording file would have been closed.

This command is mostly useful when used in conjunction with conditional jump commands (a jump is made if a condition is met, e.g. when the pen is pressed in a particular area of the tablet).

Jumps to a label in the script

JumpToIfKeyPressIs (Label, Key, MustCloseRec)

This command “jumps” to the relevant label (see definition p. 15) if the last keyboard key pressed was the one specified in the parameters (“*Key*”). If this condition is met, and if specified in the parameters, the recording file is closed.

All commands between this command and the relevant label will be ignored and the script will resume after the label.

Replace “*label*” with the name of the label you want the script to jump to (minus “:”).

Replace “*Key*” with the keyboard key value you want. Please note that this value is case sensitive (“*q*” is different from “*Q*”).

Replace “*MustCloseRec*” with “*TRUE*”, if you want to close an open data recording file, or “*FALSE*” if you do not need to.

Example

```
Command2
JumpToIfKeyPressIs (Zone1Go, q, FALSE)
Command3
:Zone1go
command5
```



When the script interpreter reads `JumpToIfKeyPressIs (Zone1Go, q, FALSE)`, it will check if the last keyboard character typed is “*q*”. If this is the case, it will jump to “*Zone1go*”. Commands between `JumpToIf[...]` and the destination label will not be executed (`command3`). The script will continue and execute `command4`. If the “*MustCloseRec*” parameter had been given the “*TRUE*” value, the participant's recording file would have been closed.

If the last key pressed is not “*q*”, the script will execute the next command (`Command3`).

Jumps to a label in the script if a text matches another text

JumpToIfTextIs (Label, Text1, Text2, MustCloseRec)

This command “jumps” to the relevant label (see definition p. 15) if the texts to compare (*Text1* and *Text2*) specified in the parameters are the same. If this condition is met, and if specified in the parameters, the recording file is closed.

All commands between this command and the relevant label will be ignored and the script will resume after the label.

Replace “**label**” with the name of the label you want the script to jump to (minus “:”).

Replace “**Text1**” with the first “sentence” to compare and “**Text2**” with the second. Please note that the matching is case sensitive: “sentence” and “Sentence” are not the same.

Replace “**MustCloseRec**” with “TRUE”, if you want to close an open data recording file, or “FALSE” if you do not need to.

This command can be powerful, for example when one need to compare the content of a list item, or an element of a list item to a specific value.

Example of use

To present a randomized list of stimuli with a specific task to perform, based on stimulus category.

We will load stimuli in the list (see List, p.18), where each item (line) is made of a stimulus word and an additional word to specify its category. For example:

```
table#object
kindness#concept
[...]
```

Then, when processing the list of items, we will split each item into Elements with the command `GetElements(%L%)` (see Elements, p.19). Then, we will trigger a jump into the script, based on the second element (our “category”), retrieved with the keyword “%E2%” (see Keywords, p.58):

```
→ JumpToIfTextIs (ObjectTask, %E2%, object, FALSE)
→ JumpToIfTextIs (ConceptTask, %E2%, concept, FALSE)
→ JumpTo (NextTask, FALSE)
→ :ObjectTask
   DoOneThing
   JumpTo (NextTask, FALSE)
→ :ConceptTask
   DoAnotherThing
→ :NextTask
```

Let’s read these lines.

In the first line, the content of the second element will be compared with the word `object`. If it is the same, the script interpreter will jump to the label `ObjectTask` (red arrow). If not, the script interpreter will evaluate the next line and compare the second element with the word `concept`. If they match, it will jump to the label `ConceptTask` (blue arrow). If not, it will evaluate the next line, which instructs to jump to the label `NextTask` (black arrow).

Note: we inserted a `JumpTo (NextTask, FALSE)` command after `DoOneThing`, because we do not want the `ConceptTask` and following commands to be executed.

Jumps to a label in the script, based on participant's group name

JumpToIfGroupIs (Label, GroupName, MustCloseRec)

This command “jumps” to the relevant label (see definition p. 15) if the participant's group name (selected in the script panel) and the name specified in the parameters (`GroupName`) are the same. If this condition is met, and if specified in the parameters, the recording file is closed.

All commands between this command and the relevant label will be ignored and the script will resume after the label.

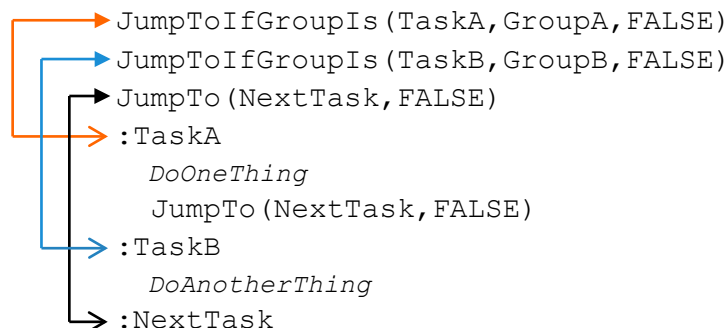
Replace “*label*” with the name of the label you want the script to jump to (minus “:”).

Replace “*GroupName*” with the name of the group to compare to. Please note that the matching is case sensitive: “group a” and “Group A” are not the same.

Replace “*MustCloseRec*” with “TRUE”, if you want to close an open data recording file, or “FALSE” if you do not need to.

Example of use

To have participants perform a specific task, based on their group name.



Let's read these lines.

In the first line, the participant's group name will be compared with the word “GroupA”. If it is the same, the script interpreter will jump to the label `TaskA` (red arrow). If not, the script interpreter will evaluate the next line and compare the participant's group name with the word “GroupB”. If they match, it will jump to the label `TaskB` (blue arrow). If not, it will evaluate the next line, which instructs to jump to the label `NextTask` (black arrow).

Note: we inserted a `JumpTo (NextTask, FALSE)` command after `DoOneThing`, because we do not want the `TaskB` and following commands to be executed.

Jumps to a “target” label when the script has passed the last label (preceding the command) a certain number of times

JumpToIfNumberIs (Iterations, Label, MustCloseRec)

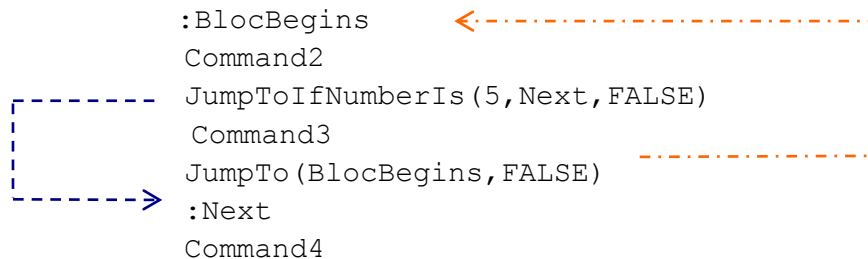
This command is only executed if the script has passed the label in front of (above) the command a certain number of times. If this is the case, the jump will be executed to the relevant label.

Replace “**Iterations**” with the maximum number of times the script can “see” the preceding label before executing the jump.

Replace “**Label**” with the name of the label to jump to (minus “:”).

Replace “**MustCloseRec**” (close an open recording data file or leave it open) with “TRUE” to activate this parameter or “FALSE” to ignore it.

Example



When the script reaches the `JumpTo (BlocBegins, FALSE)` command, it jumps to the “:BlocBegins” label. The `JumpToIfNumberIs (5, Next, FALSE)` command is activated after its fifth “reading”. The command is then executed and the script jumps to the “:Next” label. `Command3` and `JumpTo (BeginningOfBloc, FALSE)` are ignored and not executed. The script resumes from the “:Next” label, reads `command4` and executes it.

Examples of use

- 1 – Making a participant repeat a task several times.
- 2 – Used in conjunction with other commands, allowing a participant to repeat a task a maximum number of times, without any obligation to reach this maximum. By the end, `Command2` will have been executed 5 times, whereas `Command3` will have been executed 4 times.

Jumps to a “target” label when a given label’s counter reaches a given value

JumpToIfLabelIs (DestinationLabel, LabelToCheck, Iterations, MustCloseRec)

This command is only executed if the designated label “to watch” has been read a certain number of times (if its counter reaches a predetermined value). The script execution then “jumps” to the “target” label’s location in the script.

Replace “**DestinationLabel**” with the name of the label to jump to (minus “:”).

Replace “**LabelToCheck**” with the name of the label (minus “:”) whose counter is to be checked against a predetermined value.

Replace “**Iterations**” with the value to “look for”.

Replace “**MustCloseRec**” (close an open recording data file or leave it open) with “TRUE” to activate this parameter or “FALSE” to ignore it.

Note

This command is similar to the `JumpToIfNumberIs` command, except that instead of implicitly using the current label’s (last “seen” label’s) counter value, a label name is explicitly given to serve as a criterion. It is possible to monitor the value of any label defined anywhere in a script.

Jumps to a “target” label if its counter value is below a certain value

LoopIfLabelIsBelow(Label, Iterations, MustCloseRec)

This command is executed if the “watched” label’s counter value is below the value specified in the command parameters.

This command allows you to jump to the “watched” label.

Replace “**Label**” with the name of the label to “watch” (minus “:”) and to jump to when needed.

Replace “**Iterations**” with the threshold value.

Replace “**MustCloseRec**” (close an open recording data file or leave it open) with “TRUE” to activate this parameter or “FALSE” to ignore it.

Use

The basic function of this command is to allow you to perform loops, i.e. a block of actions you want to have performed a certain number of times.

Example

```
→ :BlockStart
   Command1
   Command2
   Command3
   LoopIfLabelIsBelow(BlockStart, 5, FALSE)
   Command4
```

When the script interpreter reads the command

`LoopIfLabelIsBelow(BlockStart, 5, FALSE)`, it checks the value of the internal counter of the “BlockStart” label. If this value is below 5, it jumps to the “BlockStart” label; otherwise it goes on to execute the next command `Command4`.

Example of use

Make a participant perform the same task more than once.

Notes

- When the script interpreter “reads” the “BlockStart” label for the first time, its internal counter is set at 1.
- You can modify a label’s counter value with the help of the `SetLabelCounter` command.

Jumps in the script triggered by a tablet zone

Associates a label with a tablet zone

DefineTabZone (X1, Y1, X2, Y2, Label)

This command allows you to “bind” the indicated label to a tablet area. When the participant presses the pen into the defined tablet area, the script jumps to the label location (in script).

To define the tablet zone, replace “**X1**”, “**Y1**”, “**X2**” and “**Y2**” with the area’s coordinates on the tablet.

Replace “**label**” with a word of your choice.

Reminder: the coordinates are defined in tablet units.

Example

```
DefineTabZone (7327, 5015, 1850, 2415, Zone1Go)
Command3
Command4
:Zone1Go
command5
```

`DefineTabZone (7327, 5015, 1850, 2415, Zone1Go)` defines a tablet zone with coordinates $X1=7327$, $Y1=5015$, $X2=1850$ and $Y2=2415$, bound to the “:Zone1Go” label.

You can use this command several times to create several zones that will be used at the same time.

This way, you create a “multiple choice” situation.

Zones will be checked in their order of creation. This means that you can create one zone inside another, by creating the smaller zone first, then creating a larger one to encompass it. When the pen is pressed in the smaller zone, it will be recognized as an activation of the smaller zone, not the larger one.

You can create multiple zones aiming towards the same label.

Jumps to a label depending on the selected tablet zone

WaitForTabZones (CanDraw, MustLeave, MustCloseRec)

This command stops the script until the participant has pressed the pen in one of the zones previously defined using the `DefineTabZone (X1, Y1, X2, Y2, Label)` command (see previous command description).

The **parameters** can be given two values :

- “TRUE”: activates the parameter;
- “FALSE”: deactivates the parameter.

The table below explains these parameters, assuming a “TRUE” value.

LABEL	DESCRIPTION
-------	-------------


<i>CanDraw</i>	The participant's writing (drawing) is reproduced on the screen until the pen is pressed in one of the zones the script is waiting for.
<i>MustLeave</i>	The pen must "quit" the zone before the script can continue.
<i>MustCloseRec</i>	Closes the current data recording file.

Example

```

Command2
DefineTabZone (7327,5015,1850,2415,Zone1Go)
Command3
WaitForTabZones (TRUE,FALSE,TRUE,FALSE)
Command4
:Zone1go
command5

```



The “:Zone1Go” label *is bound* to a tablet zone by the command

```
DefineTabZone (7327, 5015, 1850, 2415, Zone1Go).
```

The script executes `Command3` and goes on hold, with the command

```
WaitForTabZones (TRUE, FALSE, TRUE, FALSE).
```

As the `CanDraw` parameter is “TRUE”, the participant's writing is shown on the screen until the command comes to an end, i.e. when the participant presses the pen in the defined tablet zone.

As the `MustLeave` parameter is “TRUE”, the command will only come to an end when the pen leaves the tablet zone.

The script will then “jump” to the “:zone1Go” label.

The script will continue and execute `Command5`.

In this example, you can see that `command4` will never be executed.

Waits for the pen to leave the active zone

WaitForPenOut (MaxStimulusDisplay, StimulusType)

This command halts script execution until the participant has removed the pen from the zone it is being pressed in. Command parameters allow you to specify a maximum display duration for a stimulus type.

When used jointly with `WaitForTabZones`, this command allows you to manage the start and end of a stimulus after a given length of time (or earlier, depending on the participant’s behavior).

Example

With the command `WaitForPenOut (3000, 'IMAGE')`, each image-type stimulus will be erased from the screen after 3 seconds (3000 milliseconds), or earlier if the participant removes the pen from the zone before the 3 seconds are up.

Script execution will resume as soon as the participant has removed the pen from the zone.

Stimulus type are:

- IMAGE
- TEXT (text file)
- MESSAGE
- VIDEO
- AUDIO
- ALL: every stimulus type (listed above). If several stimuli are simultaneously played (e.g. a picture and a sound), both will be stopped.

Example of use in a script:

```
; define a tablet zone to trigger the display of a picture
DefineTabZone(1,1,120,120,zone_image_1)

; define a zone to end the script
DefineTabZone(1,200,120,320,zone_end)

; jump into the "loop"
jumpTo(WaitForTabzonesAnchor)

; label for "image 1" zone
:zone_image_1

; display picture image1.bmp for an unlimited duration (-1)
DisplayPic(image1.bmp,-1,0,0)

; wait for the pen to be removed. After 3 seconds, clear the picture.
WaitForPenOut(3000,IMAGE)

; label for the "loop"
:WaitForTabzonesAnchor

; wait for the pen to be pressed in a zone. Parameters: writing on the screen. If
the participant selects a zone intended for script, do not wait for the pen to be
removed.
WaitForTabZones(TRUE,FALSE,FALSE)

; label for "end"
:zone_end

; wait for the pen to be removed. After 2 seconds, stop all stimuli.
WaitForPenOut(2000,ALL)
```

With this script, the participant has to write (the pen's trace will appear on screen) and he or she can do so in two zones, each associated with an action:

- a zone linked to the label "zone_image_1" where the picture image1.bmp appears when the participant presses the pen and disappears after 3 seconds, unless the participant removes the pen beforehand (in which case the picture is immediately cleared);
- a zone linked to the "zone_end" that stops all stimuli after 2 seconds, unless the participant removes the pen beforehand. The script then ends because there are no more commands.

Clears previously defined tablet zones (defined by “DefineTabZones”)

ClearZones

This command erases all tablet zones previously defined using the “DefineTabZone” command.

Eye tracker

Redefines the calibration area coordinates on the screen

SetCalibrationCoord(x1, y1, x2, y2)

This command redefines the calibration area coordinates on the screen (initially defined in the “Eye tracker” tab of the configuration panel).

Replace “X1”, “Y1”, “X2” and “Y2” with the calibration area coordinates (in pixels) you have chosen (see p. 8).

Example

SetCalibrationCoord(0, 0, 1024, 768) defines a calibration area that is 1024 pixels wide (horizontally) and 768 pixels high (vertically), beginning in the screen's upper left-hand corner.

Hint: If you are using two 1024*768 resolution screens, the SetCalibrationCoord(1024, 0, 2048, 768) command will locate the calibration area on the second (screen) monitor.

Launches the calibration procedure

TestCalibration

This command launches the calibration procedure for the selected eye tracker; For Eyelink, the calibration procedure is supplied by the eye tracker interface.

Tests calibration reliability

TestDrift

This command launches a test to assess eye tracker coordinate drift.

For Eyelink, the test is run with a single (central) point, whereas with ASL504, the entire calibration procedure is repeated.

Writing display

Warning

The script commands presented below allow you to manipulate the way in which the pen's trace is displayed on the screen. Bear in mind that using more than one of these "special effects" at the same time may produce some surprising results. Please also remember that the behavior of the masking/unmasking commands (see the Masking section) is linked to the pen tip's position on the screen.

Redefines the color of the trace left by the pen on the screen

SetPenColor (ColorValue)

This command redefines the "ink" color of the pen when tracing is reproduced on the screen (initially defined in the "Display" tab of the configuration panel).

Color numbers follow the web standard and range from #**000000** (**black**) to #**FFFFFF** (**white**).

Example: after the command `SetPenColor (#000000)`, the pen will write in black.

Redefines the size of the trace left by the pen on the screen

SetPenWidth (WidthValue)

This command redefines the thickness of the pen when tracing is reproduced on the screen (initially defined in the "Display" tab of the configuration panel).

Replace "**WidthValue**" with the new line thickness value, expressed in pixels.

Example: `SetPenWidth (2)`.

Time shift writing display

Sets the delay for the pen trace's display on the screen

SetTabTimeShiftDelay (Duration)

This command determines the time it takes for the pen's trace to appear on the screen.

Replace "**Duration**" with the delay value (in milliseconds) you require.

Example

The following command delays the pen trace display on the screen by 300 milliseconds: `SetTabTimeShiftDelay (300)`

Activates/deactivates the delayed display of the pen's trace on the screen

ActivateTabTimeShift (Active)

This command allows you to activate or deactivate the delay in displaying the pen's trace on the screen. This function is associated with the following commands:

WaitForTabZoneAt, WaitForTabZones and RecStandard.

Replace "**Active**" with TRUE to activate the delay and with FALSE to deactivate.

Example

The following command activates the delayed display of the pen's trace on the screen: `ActivateTabTimeShift (TRUE)`

Transform writing display

Shifts writing display on screen

SetTabOffset (x, y)

This command allows you to change the reference location for the display of the pen's trace on the screen. This function is associated with the following commands:

WaitForTabZoneAt, WaitForTabZones and RecStandard.

Replace "**X**" with the horizontal offset value (in pixels) you want. A positive value shifts to the right, whereas a negative value shifts to the left.

Replace "**Y**" with the vertical offset (in pixels) you require. A positive value shifts toward the bottom, whereas a negative value shifts toward the top.

Example

The following command moves the display of writing up by 50 pixels, relative to the location where it would normally be displayed:

```
SetTabOffset (0, -50)
```

To revert to a "normal" display: `SetTabOffset (0, 0)` .

Changes the writing's display proportions on screen

SetTabRatio (Xratio, Yratio)

This command allows you to modify the relationship between the pen movements on the tablet and the way they are displayed on the screen. As such, it allows you to enlarge or shrink the pen trace display on the screen.

"**Xratio**" and "**Yratio**" are expressed as a percentage of the normal ratio. Thus, a value between 1 and 99 will shrink the trace size, a value of 100 will not change anything and a value above 100 will enlarge the writing display.

This function is associated with the following commands: WaitForTabZoneAt, WaitForTabZones and RecStandard.

Replace "**Xratio**" with the horizontal modification factor value you want and

“**Yratio**” with the vertical modification percentage that is needed.

Example

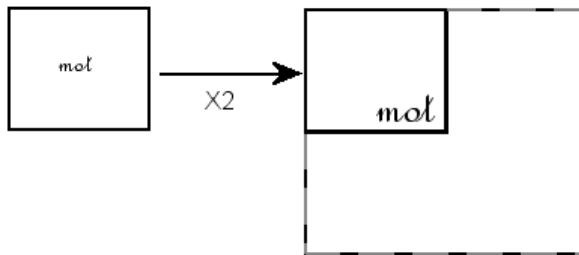
The following command displays the writing trace at 90% of its normal size, both horizontally and vertically:

```
SetTabRatio(90, 90)
```

To revert to the “normal” display: `SetTabRatio(100,100)` .

Note

Since modifying the display proportions affects the display surface area (virtual) but not the screen size (physical), enlarging the writing gives the impression that the writing has shifted to the bottom right-hand corner of the screen, whereas shrinking the writing will “shift” it to the top left-hand corner. For example, if the size of the writing is doubled with `SetTabRatio(200, 200)` , the point that should have been displayed at screen coordinates (20,20) will be displayed at coordinates (40,40). Think of it as though the size of the display had been doubled but only the upper left-hand quarter was still visible on the screen, because the screen’s size cannot be doubled as well.



Write until...

Stops the script and waits for the participant to press the pen in a defined tablet area

WaitForTabZoneAt (x1 , y1 , x2 , y2 , CanDraw , MustLeave)

This command stops the script until the participant presses the pen in the tablet area defined in this command. If the parameter “**CanDraw**” is activated, the participant’s writing is displayed on screen. This command is thus a mean to have a participant write (draw) on the tablet whilst displaying his or her writing’s trace on screen until he or she presses the pen in a defined area (zone).

To define the tablet zone (area), replace “**X1**”, “**Y1**”, “**X2**” and “**Y2**” with the coordinates (in tablet units, i.e. lines) of the area you have chosen.

- With the 4 coordinates set to **0**, the pen may be pressed anywhere on the tablet to "re-start" the script.
- With the 4 coordinates set to **-1**, it is the **start zone** coordinates which are used (defined in the “Script” tab of the acquisition configuration panel)

The other parameters can be given two values :

- “TRUE”: activates the parameter;
- “FALSE”: deactivates the parameter.

The table below explains these parameters, assuming a “TRUE” value.

LABEL	DESCRIPTION
<i>CanDraw</i>	The participant's writing is reproduced on the screen, until the pen is pressed in the zone.
<i>MustLeave</i>	The pen must leave the zone before the script can continue.

Example

`WaitForTabZoneAt(17327, 5015, 19850, 2415, FALSE, TRUE, FALSE)` stops the script. If the participant presses the pen in the defined tablet area (coordinates $X1=17327$, $Y1=5015$, $X2=19850$, $Y2=2415$), the script resumes.

Stops the script and waits for the participant to press a keyboard key
WriteUntilKeypress

This command stops script execution until a keyboard key is pressed, but meanwhile, the participant’s writing is displayed on screen.

Sets a maximum time for production

MaxWritingDuration (MaxDuration)

This command does not suspend script execution, but does set the maximum time (in milliseconds) during which the participant can write (for commands `WaitForTabZoneAt`, `WaitForTabZones`, `WriteUntilKeypress` and `RecStandard`).

Replace `MaxDuration` with a duration expressed in milliseconds.

Example

The command `MaxWritingDuration(360000)` limits the duration to 6 minutes (i.e. $6*60*1,000$). If the participant has not pressed the pen in one of the defined zones (or end zone for `RecStandard`) by the time this period has elapsed, the command terminates “on its own” and script execution continues.

Special value: -1 (unlimited duration). This is the default value when no value has been explicitly defined. You may set this value to cancel a previous time limitation.

Using the “Simple” acquisition mode

The commands described below are intended to give you the ability to use the “Simple” acquisition mode (“Simple” tab of configuration panel), allowing you to change some or all of its parameters.

You will then be able to record more than one task using this paradigm, for instance, with different pictures each time. Then again, you could use this task without any modification, but in the middle of a set of other tasks.

Launches the Simple acquisition paradigm

RecStandard (AddToParticipantName)

The `RecStandard` command allows you to activate the simple acquisition paradigm, with the parameters defined in the configuration panel (Simple tab) of Eye and Pen. Replace “**AddToParticipantName**” with a suffix that will be added to the recording file's name.

Caution: this command manages the opening and closing of the recording data files. This means that you do not have to use `OpenRec` and `CloseRec`.

Example

For the participant “Toto”, the command `RecStandard(_standard)` launches the Simple acquisition (as if you had launched it yourself, using the File/Acquisition/Simple menu of Eye and Pen). The tablet data will be saved in “*toto_standard.tab*” and the eye-tracking data in “*toto_standard.eye*”.

Launches the Simple acquisition, changing background picture and trigger zone use

RecNewUsages (AddToParticipantName, UseBack, UseZone1, ShowOnStart, HideOnPress, UseZone2)

This command allows you to launch the Simple acquisition, redefining the activation of the background picture and trigger zones 1 and 2. These uses were initially defined in the configuration panel (Simple tab).

Caution: this command manages the opening and closing of the recording data files. This means that you do not have to use `OpenRec` and `CloseRec`.

Replace “*AddToParticipantName*” with a suffix that will be added to the recording file's name.

Parameters can be given two values:

- “TRUE”: activates the parameter;
- “FALSE”: deactivates the parameter.

The following table describes the parameters, assuming a “TRUE” value.

PARAMETER	DESCRIPTION
<i>UseBack</i>	Use a background picture.

<i>UseZone1</i>	Use trigger zone 1.
<i>ShowOnStart</i>	Show the picture associated with trigger zone 1 as soon as recording starts.
<i>HideOnPress</i>	Hide the picture associated with trigger zone 1 as soon as the participant presses the pen on the tablet. Caution: this parameter will only be taken into account if “ <i>ShowOnStart</i> ” is activated.
<i>UseZone2</i>	Use trigger zone 2.

Example

With the participant “*Toto*”, the command

```
RecNewUsages (_P3, FALSE, TRUE, FALSE, FALSE, FALSE) :
```

- saves data under the name “*toto_P3*”;
- there will not be any background picture (“*UseBack*” is “*FALSE*”);
- trigger zone 1 will be used (“*UseZone1*”= “*TRUE*”);
- the associated picture will not be displayed (“*ShowOnStart*” is “*FALSE*” and so “*HideOnPress*” will be ignored);
- trigger zone 2 will not be used (“*UseZone2*” is “*FALSE*”).

Launches the Simple acquisition, changing pictures

RecNewPics (AddToParticipantName, BackPic, Pic1, Pic2)

This command allows you to launch the Simple acquisition and to change the background picture and pictures associated with trigger zones 1 and 2 (initially defined in the Simple tab of the configuration panel).

Caution: this command manages the opening and closing of the recording data files. This means that you will not have to use `OpenRec` and `CloseRec`.

Replace “***AddToParticipantName***” with a suffix that will be added to the recording file’s name.

The following table describes the other parameters.

PARAMETER	DESCRIPTION
<i>BackPic</i>	Background picture filename
<i>Pic1</i>	Picture filename associated with trigger zone 1
<i>Pic2</i>	Picture filename associated with trigger zone 2

Example:

For the participant “*Toto*”, the command

```
RecNewPics (_P2, YellowBack.bmp, Capsela.bmp, car.bmp) :
```

- saves the data under the name “*toto_P2*”;
- the background picture is “*YellowBack.bmp*” (“*BackPic*” is

- “YellowBack.bmp”);
- the picture associated with trigger zone 1 is “Capsela.bmp”;
- the picture associated with trigger zone 2 is “car.bmp”.

Launches the Simple acquisition, changing picture names and uses

RecNewPics&Usages (AddToParticipantName, BackPic, UseBack, Pic1, UseZone1, ShowOnStart, HideOnPress, Pic2, UseZone2)

This command is a “compilation” of RecNewPics and RecNewUsages (see above). It launches the Simple acquisition, allowing you to change the pictures and their activation for background and trigger zones at the same time.

Caution: this command manages the opening and closing of the recording data files. This means that you do not have to use OpenRec and CloseRec.

Replace “**AddToParticipantName**” with a suffix that will be added to the recording file’s name.

Parameters for use can be given two values:

- “TRUE”: activates the parameter;
- “FALSE”: deactivates the parameter.

The following table describes the parameters (assuming a “TRUE” value for use parameters):

PARAMETER	DESCRIPTION
BackPic	Name of the background picture file.
UseBack	Use the background picture
Picture1	Name of the picture file associated with trigger zone 1.
UseZone1	Use trigger zone 1
ShowOnStart	Display the picture associated with trigger zone 1 as soon as the protocol recording begins
HideOnPress	Hide the picture associated with trigger zone 1 as soon as the participant presses the pen on the tablet. Caution: this parameter will only be taken into account if “ShowOnStart” is activated.
UseZone2	Use trigger zone 2
Picture2	Name of the picture file associated with trigger zone 2.

Example

For the participant “Toto”, the command

```
RecNewPics&Usages (_P4, Bkgnd.bmp, TRUE, Capsela.bmp, TRUE, TRUE, TRUE, car.bmp, FALSE):
```

- the recorded data will be saved under the name “*toto_P4*”;
- the background picture is “*Bkgnd.bmp*”;
- the background picture will be displayed (*UseBack = TRUE*);
- the picture file named “*Capsela.bmp*” is associated with trigger zone 1;
- trigger zone 1 will be used (*UseZone1 = TRUE*);
- the picture associated with trigger zone 1 will be displayed when the command starts (*ShowOnStart = TRUE*);
- the picture associated with trigger zone 1 will be removed as soon as the participant presses the pen on the tablet to write or draw (*HideOnPress=TRUE*);
- the picture “*car.bmp*” is associated with trigger zone 2;
- trigger zone 2 will not be used (*UseZone2 = FALSE*).

Sets new coordinates for trigger zones 1 and 2

SetPicsZones (*x1Zone1* , *y1Zone1* , *x2Zone1* , *y2Zone1* ,
x1Zone2 , *y1Zone2* , *x2Zone2* , *y2Zone2*)

This command redefines the coordinates of trigger zones 1 and 2 on the tablet. These coordinates will replace the coordinates defined in the configuration panel of Eye and Pen (Simple tab). These new values will remain until the original values are restored by the “RestoreOriginalPicsZones” command (see below).

Replace “*X1Zone1*”, “*Y1Zone1*”, “*X2Zone1*”, “*Y2Zone1*” with the new coordinates you want to use for **trigger zone 1**.

Replace “*X1Zone2*”, “*Y1Zone2*”, “*X2Zone2*”, “*Y2Zone2*” with the new coordinates you want to use for **trigger zone 2**.

Reminder: coordinates are defined in tablet units.

Example

The command

```
SetPicsZones (27094,23480,30480,18203,27094,14626,30480,9434)
```

defines:

- the coordinates for trigger zone 1 :
X1=27094, Y1=23480, X2=30480, Y2=18203 ;
- the coordinates for trigger zone 2 :
X1=27094, Y1=14626, X2=30480, Y2=9434.

Restores original coordinates for trigger zones 1 and 2

RestoreOriginalPicsZones

This command restores the coordinates for trigger zones 1 and 2 that were originally defined in the configuration panel (Simple tab), before being redefined via the `SetPicsZones` command (see above).

Launches the Simple acquisition mode, redefining all parameters

RecNewAll (*AddToParticipantName* , *BkPic* , *UseBack* ,
Picture1 , *x1Zone1* , *y1Zone1* , *x2Zone1* , *y2Zone1* ,
UseZone1 , *ShowOnStart* , *HideOnPress* ,
Picture2 , *x1Zone2* , *y1Zone2* , *x2Zone2* , *y2Zone2* ,

UseZone2 , X1ZoneEnd , Y1ZoneEnd , X2ZoneEnd , Y2ZoneEnd)

This command launches the Simple acquisition paradigm, redefining all the parameters of the acquisition configuration panel (Simple tab).

Caution: this command manages the opening and closing of the recording data files. This means that you do not have to use OpenRec and CloseRec.

Replace “**AddToParticipantName**” with a suffix that will be added to the recording file’s name.

Replace “**x1Zone1**”, “**y1Zone1**”, “**x2Zone1**”, “**y2Zone1**” with new coordinates for trigger zone 1 (in tablet units).

Replace “**x1Zone2**”, “**y1Zone2**”, “**x2Zone2**”, “**y2Zone2**” with new coordinates for trigger zone 2 (in tablet units).

Replace “**x1ZoneEnd**”, “**y1ZoneEnd**”, “**x2ZoneEnd**”, “**y2ZoneEnd**” with new coordinates for the "end" zone (in tablet units)

The other parameters can be given two values:

- “TRUE”: activates the parameter;
- “FALSE”: deactivates the parameter.

The following table describes the other parameters (assuming a “TRUE” value for use parameters):

PARAMETER	DESCRIPTION
UseBack	Use background picture.
BackPic	Name of the background picture file.
Picture1	Name of the picture file associated with trigger zone 1.
UseZone1	Use trigger zone 1.
ShowOnStart	Display the picture associated with trigger zone 1 as soon as the protocol recording begins.
HideOnPress	Hide the picture associated with trigger zone 1 as soon as the participant presses the pen on the tablet. Caution: this parameter will only be taken into account if “ <i>ShowOnStart</i> ” is activated.
UseZone2	Use trigger zone 2.
Picture2	Name of the picture file associated with trigger zone 2.

Example

For the participant “*Toto*”, the command

```
RecNewAll( _P5, YellowBkgnd.bmp, TRUE, Capsela.bmp, 27094, 23480, 30480, 18203, TRUE, TRUE, TRUE, car.bmp, 27094, 14626, 30480, 9434, FALSE, 12456, 1845, 14170, 1235) defines :
```

-
- the data that will be saved under the name “*toto_P5*” (“*toto_P5.tab*”)
 - the background picture is “*YellowBkgnd.bmp*”;
 - the background picture will be displayed (*UseBack = TRUE*);
 - the picture file named “*Capsela.bmp*” is associated with trigger zone 1;
 - the coordinates of trigger zone 1 on the tablet are :
X1=27094, Y1=23480, X2=30480, Y2=1820
 - trigger zone 1 will be used (*UseZone1 = TRUE*);
 - “*Capsela.bmp*” will be displayed when the command starts
(*ShowOnStart = TRUE*);
 - the picture associated with trigger zone 1 will be removed as soon as the participant presses the pen on the tablet to write or draw
(*HideOnPress=TRUE*);
 - the picture “*car.bmp*” is associated with trigger zone 2
 - the coordinates of trigger zone 2 on the tablet are:
X1=27094, Y1=14626, X2=30480, Y2=9434;
 - trigger zone 2 will not be used (*UseZone2 = FALSE*);
 - the coordinates of the "end" zone on the tablet are:
X1=12456, Y1=1845, X2=14170, Y2=1235.
-

Sets if tracing should be displayed on screen

ShowSimpleTrace (Visible)

This command defines if the pen trace should be displayed on screen with the Simple acquisition paradigm (RecStandard).

Replace “**Visible**” with one of the following values:

- “TRUE”: trace will be displayed;
 - “FALSE”: no trace will be shown.
-

Sets the maximum display duration for pictures associated with tablet zones

SetRecStandardMaxDisplay (MaxDurationForPicture1, MaxDurationForPicture2)

This command defines the maximum display duration for the pictures associated with trigger zones 1 and 2 of the Simple acquisition paradigm (RecStandard).

Example

The command `SetRecStandardMaxDisplay(1000,2000)` limits the duration of the picture display to 1 second (1,000 milliseconds) when the participant presses the pen in zone 1 and to 2 seconds when the pen is pressed in zone 2.

Special value: -1 (unlimited duration). This is the default value when no value has been explicitly defined.

NetSync

The NetSync module allows you to drive a recording session on several computers via network commands (see Netsync in the User manual).
The commands shown below should only be used in this context.

Stops the script until a signal is sent by the Master host

WaitForNetSync

This command halts script execution.
Script execution resumes when the “Go” signal is received from the Master host.

Sends a message to the Master host

SendMessageToNetSync (Message)

This command allows you to send a message to the Master host. This message is displayed in the “State” column of the NetSync data grid holding the client hosts.

Examples of use

- Show (on the Master host) the item currently being processed on the client host;
- Follow the stages of the experiment on each client host;

Example

The command `SendMessageToNetSync (item 3)` sends the message “item 3” to the Master host.

The message may include or be based on one or more keywords (see *Keywords* section, p. 58). For example, `SendMessageToNetSync (%L%)` sends the current list item; `SendMessageToNetSync (item %I%/%LCount%)` sends the label counter value and the number of items in the list (e.g. “item 5/12”).

Network messaging

Using the UDP protocol, the following commands allows to send messages to another device on the same network.

Activates/deactivates network messaging feature

UseNetMessageing (Active)

This command enables/disables a network messaging feature (UDP protocol).

Replace “**Active**” with one of the following values:

- “TRUE”: network feature is enabled;
- “FALSE”: network feature is disabled and communication is closed.

Communication parameters are set in the configuration panel, Network tab. The Device test / Network message menu allows to test that the connection is properly

working.

Sends a message to another computer

SendNetMessage (Message)

As its name says, this command sends your message through the network to another computer. For this command to work, network messaging feature should have been activated first (see previous command).

I/O

The following commands allow to use the computer parallel port to send or receive data. A classical usage is to send or receive triggers from/to another device, such as an EEG recorder, for example. The port configuration part is dealt with in the Eye and Pen User manual.

Activates/deactivates parallel port

UseParallelPort (Active)

This command enables/disables a parallel port driver.

Replace “**Active**” with one of the following values:

- “TRUE”: parallel port feature is enabled;
- “FALSE”: parallel port feature is disabled.

Communication parameters are set in the configuration panel, Network tab. The Device test / Network message menu allows to test that the connection is properly working.

Sends a number through parallel port

SendToParallelPort (aValue)

As its name says, this command sends a numeric value through the parallel port.

For this command to work, parallel port driver should have been activated first (see UseParallelPort).

Replace “**aValue**” by a number between 0 and 255.

Sends a number through parallel port

SetParallelPortLines (LinesState)

This command allows to individually set the parallel port lines state, either high (+5volts) or low (0 volt).

For this command to work, the parallel port driver should have been activated first (see UseParallelPort).

Replace “**LinesState**” by a series of values, either 0 (low) or 1 (high), one for each of the 8 lines of the parallel port. Lines are from 1 (left) to 8 (right).

Example

`SetParallelPortLines(10000000)` set high line number 1 and low all the others.

Stops the script until a specific value is recieved on the parallel port

WaitForParallelPortValue(aValue)

This command puts the script in pause until a particular value is read on the parallel port. For this command to work, parallel port driver should have been activated first (see `UseParallelPort`).

Replace “**aValue**” by a number between 0 and 255.

Note: this command can be canceled with a key press on the “Escape” or “F12” keyboard key.

Example of use

This command can be used to synchronize and experiment with an external device or another computer.

Keywords

Keywords are special words that are replaced by a “definite” value during script execution. Keywords can be likened to the notion of *variable* that is used in common programming languages, except that a keyword cannot be modified by hand, being “read-only” (its value can only be “read” by a script).

Keywords can be included in every parameter of every script command.

Retrieves an element

%EValue%

When executing the script, this keyword is replaced with the element in position given by “Value” in the list of elements created after an item has been split with the command `GetElements`.

Example

The following command display the 2nd element:

```
DisplayMsg(%E2%,-1,-1,-1,TRUE)
```

Takes on the number of elements in the item

%ECount%

When executing the script, this keyword is replaced with the number of elements found in the item after it has been split with the command `GetElements`.

Example

The following command may be used to exit a loop when all the elements of an items have been processed:

```
JumpToIfNumberIs (End, %ECount%, TRUE)
```

Group name

%G%

When executing the script, this keyword is replaced with the name of the group set in the script launching dialog box.

Takes the value of the current label's internal counter

%I%

This keyword is replaced with the value of the current label's counter (the last label read in the script).

Example

The following command adds the value of the current label's counter to the name of the protocol recording filename (participant id.): `OpenRec (_%I%)`

Last keyboard key pressed

%K%

When executing the script, this keyword is replaced with the name of the last keyboard key pressed.

Retrieves an item from the list in the position corresponding to the current label's counter value

%L%

During script execution, this keyword is replaced with the content of an item retrieved from the list (the item is the one whose position in the list corresponds to the current label's counter value).

`%L%` may be included in the parameters of all the commands.

Example

The following script adds items to the list, randomizes the list, and has the items displayed on the screen. This example shows how to create a loop to manage the list content. Thanks to `%L%`, the script will remain the same for 100 items.

```
ResetList
```

```
; add two items: picture filenames
```

```
AddToList (image1.bmp)
```

```
AddToList (image2.bmp)
```

```
; shuffle the entire list
```

```
RandomizeList
```

```
:Start
```

```
; %L% retrieves the item at position "n" in the list, "n" being the "Start" label's  
; counter value.
```

```
; The command displays this picture (its name has just been retrieved from the
; list) ; for 2000 milliseconds, centered on the screen.
  displaypic (%L%, -1, -1, 2000)

; if the counter value of the current label ("Start") is 2, then jump to the label
; named "End"
  JumpToIfNumberIs (End, 2, True)

; if not, jump to "Start" for another round
  JumpTo (Start)

:End
```

Takes on the number of items in the list

%LCount%

When executing the script, this keyword is replaced with the number of items found in the list (list size).

Example

The following command may be used to exit a loop when all the items have been processed:

```
JumpToIfNumberIs (End, %LCount%, TRUE)
```

Retrieves a folder path

%PFolder%

When executing the script, this keyword is replaced with the specified folder path value. Values can be:

- %PImages% will be replaced with the path to the Stimuli (pictures, etc.) folder;
 - %PData% will be replaced with the path to the Data folder;
 - %PModels% will be replaced with the path to the Models folder;
 - %PScripts% will be replaced with the path to the Scripts folder.
-

Takes a random number

%RValue%

When executing the script, this keyword is replaced with a random number. Replace *Value* with the upper limit of the range within which the random number must be taken. The range goes from 0 to *Value*.

Example 1

The following command waits for a random duration of between 0 and 499 milliseconds (inclusive): `WaitFor (%R500%)`

Example 2

Create a delay of 1,000 milliseconds: `WaitFor (1000)`

Create a random delay of between 0 and 500 milliseconds:

```
WaitFor (%R501%)
```

Participant's name

%S%

When executing the script, this keyword is replaced with the participant's ID (provided in the script launching dialog box).

Caution: commands that load a file whose name includes this keyword will not be checked (no such files are supposed to exist at the time of script checking).

Example

The command `DisplayPic(%S%_page1.bmp,-1,-1,-1)` can be used to reload a page previously written by a participant and saved with the command `SaveScreenToBMP(%S%_page1)`.

Uses a label's counter value within a keyword

%?:Label%

The label's counter value may be invoked within any keyword (except `%Lcount%`, obviously).

You can therefore "target" a specific label's counter value.

Unlike the syntax used with other commands, here the label should be mentioned with its ":", just as it is shown in the script (alone on a line).

The "?" is replaced with a keyword, either **E**, **I**, **L** or **R**.

Example 1

The following command uses the value of the "Start" label's counter as an index to retrieve an item in the list: `DisplayPic(%L:Start%.jpg,-1,-1,-1)`

Example 2

The value of the "Start" label's counter is used to help "build" the name of a picture to be displayed, e.g. "pic_3.bmp".

`DisplayPic(pic_%I:Debut%.bmp,-1,-1,-1)`

Example 3

The command `WaitFor(%R:label3%)` receives a random value of between 0 and the value of the "label3" label counter.

Example 4

The command `WaitFor(%L:label3%)` uses the item in the list whose position corresponds to the value of the "Label3" label counter. This then allows you to manipulate delays using the list items.

Example 5

The command `SetPenColor(%L%)` selects the ink color from the list.

COMMAND MENU TREE VIEW

Files and Directories	▶	SetPicsDirectory SetDataDirectory SetModelsDirectory		
		Recording	▶	SetSafeRec OpenRec CloseRec
		Log	▶	AddToLog OpenMyLog AddToMyLog CloseMyLog
		Text file	▶	CreateTextFile AddToTextFile CloseTextFile CopyFileTo
Label	▶	Label ResetLabelCounter SetLabelCounter AddToLabelCounter SubTractFromLabelCounter		
List, items and elements	▶	AddToList LoadList ResetList RandomizeList RandomizeListRange AddElement GetElements		
Screen display	▶	SetDisplayWindowCoord SetBackgroundColor SetFont SaveScreenToBMP SaveScreenAreaToBMP		
		Masking	▶	SetUnmaskFile SetMaskingMode SetMaskingFillPic SetMaskingFillColor SetMaskingBlurLevel ActivateFeedbackMasking DefineUnmaskZone ClearUnmaskZones SetUnmaskZonesRange SetUnmaskZonesBackTracking MustWriteToActivateUnmaskZone
Stimuli	▶	PlayModel		
		Message	▶	DisplayMsg HideMessage
		Text	▶	DisplayText HideText
		Picture	▶	DisplayPic HidePicture DisplayImageList PreloadPics unloadPics

		SetLightSensorPosition
		DisplayPicWithSensor
	Video	▶ DisplayAVI
		StopAVI
	Audio	▶ Beep
		SystemBeep
		PlaySound
		StopSound
		SetVolume
Wait	▶	WaitFor
		WaitForKeyPress
		CountDown
Jumps	▶	JumpTo
		JumpToIfKeyPressedIs
		JumpToIfTextIs
		JumpToIfGroupIs
		JumpToIfNumberIs
		JumpToIfLabelIs
		LoopIfLabelIsBelow
Jumps triggered	▶	DefineTabZone
		WaitForTabZones
		WaitForPenOut
		ClearZones
Eye tracker	▶	SetCalibrationCoord
		TestCalibration
		TestDrift
Writing display	▶	SetPenColor
		SetPenWidth
	TimeShift	▶ SetTabTimeShiftDelay
		ActivateTabTimeShift
	Transform	▶ SetTabOffset
		SetTabRatio
Write until...	▶	WaitForTabZoneAt
		WriteUntilKeyPress
		MaxWritingDuration
Simple acquisition	▶	RecStandard
		RecNewUsages
		RecNewPics
		RecNewPics&Usages
		SetPicsZones
		RestoreOriginalPicsZones
		RecNewAll
		ShowSimpleTrace
		SetRecStandardMaxDisplay
NetSync	▶	WaitForNetSync
		SendMessageToNetSync
Network messaging	▶	UseNetMessageing
		SendNetMessage
I/O	▶	UseParallelPort
		SendToParallelPort
		SetParallelPortLines
		WaitForValueFromParallelPort

SCRIPT EXAMPLES

This section of the manual is intended to help you learn/practice script writing based on practical cases.

1. Have a task re-done

This script allows the participant to start task 1 a maximum of four times and task 2 a maximum of two times.

Each task is triggered by a zone on the tablet (we will call the zones' coordinates on the tablet X1, Y1, X2 and Y2, but in a "true" script you must replace these symbols with values expressed in tablet units).

We want to record the participant's activity on each occasion, i.e. in a different acquisition file.

When the maximum is reached for a given task, if the participant presses the relevant zone again, nothing will happen. When he or she presses the pen in the zone associated with "Stop", the script will jump to the label named "Next".

```
; we define three zones on the tablet, each being associated with a label
DefineTabZone(x1, y1, x2, y2, Task1)
DefineTabZone(x1, y1, x2, y2, Task2)
DefineTabZone(x1, y1, x2, y2, Stop)
JumpTo(Go)

:Task1
  JumpToIfNumberIs(Go, 5, FALSE)
  OpenRec(_task1_%I%)
; do something
  JumpTo(Go, TRUE)

:Task2
  JumpToIfNumberIs(Go, 3, FALSE)
  OpenRec(_task2_%I%)
; do something
JumpTo(Go, TRUE)

:Stop
  JumpTo(Next)

:Go
  WaitForTabZones(TRUE, TRUE, TRUE)

:Next
  CloseRec

; Script continues
```

Note: if the acquisition file is already closed, a call to CloseRec will have no "negative" effect.

2. Mask production (Simple)

This script allows the participant to write a page of text, but the display on the screen will be masked by a plain color (green). The participant will only be able to see what he or she writes in a round zone around the pen tip location (this zone keeps track with the pen tip's position).



Figure 149: Masking production.

*; select the unmasking file, the one that will allow you to see “through” the mask covering
; the display on screen*

```
SetUnmaskFile(..\cursors\Mask_circle120.bmp)
```

; masking mode no.1: the display is filled with a plain color

```
SetMaskingMode(1)
```

; filling color: green

```
SetMaskingFillColor(#00FF00)
```

; activate display masking

```
ActivateFeedbackMasking(TRUE)
```

; start recording

```
RecStandard
```

The unmasking file is contained in the *Cursors* folder, not the *Stimuli* folder, which explains why a relative file path is used here: “..\cursors\Mask_circle120.bmp” (for further explanations, see p. 8).

3. Mask production with unmasking locations

The purpose of this script is to have a participant write 5 words in “slots”, whose display on screen will be masked. The participant will only be able to see the word he or she is writing, together with the previous word.

```
; unmasking file selection
SetUnmaskFile(mask_5_words.jpg)

; masking mode n°2: a picture file is displayed on screen and covers the production.
SetMaskingMode(2)

; masking file selection
SetMaskingFillPic(White_Bkgnd_5_words.jpg)

; define unmasking zones (they match with the “slots” of the masking file
; from left to right.
DefineUnmaskZone(35,350)
DefineUnmaskZone(200,350)
DefineUnmaskZone(365,350)
DefineUnmaskZone(530,350)
DefineUnmaskZone(695,350)

; rule: the participant will be able to see the content of the zone he’s in and the left side one
SetUnmaskZonesRange(-1,0)

; going back is forbidden: the participant will not be able to unmask a zone
; previously “visited”
SetUnmaskZonesBackTracking(FALSE)
ActivateFeedbackMasking(TRUE)

;----Trial n°1
WaitForKeyPressPic(cross.bmp,-1,-1)

; Hint: we use the same picture as a background and as a mask
DisplayPic(White_Bkgnd_5_words.jpg,-1,-1,-1)
OpenRec(_zone_test)

; This tablet zone match with the circle in the figure 151.
WaitForTabZoneAt(6072,2981,6248,2805,TRUE,FALSE)
CloseRec
```



Figure 150: “mask_5_words.jpg” picture used for unmasking.

The following figure shows the picture that serve as background and mask, with the representation of an unmasking zone in a dotted line.

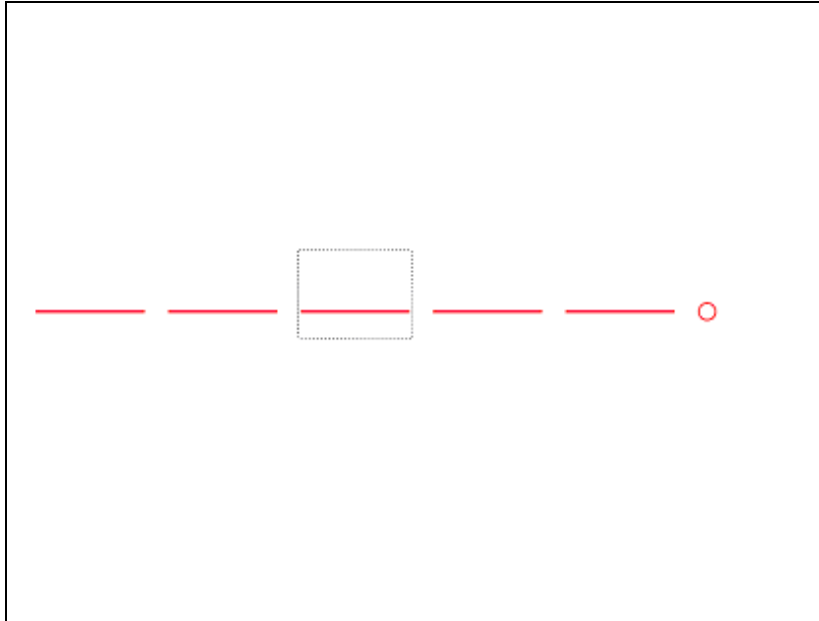


Figure 151: "White_Bkgnd_5words.jpg" (white background with 5 slots for words).

4. Load a list and show its content

The following script loads a list of item from a file (the file named V2_liste.txt, is found in the *Stimuli* folder), then displays it's content, item by item.

```
LoadList(v2_list.txt)
WaitForKeypressMsg(%LCount% items in the list,-1,-1,FALSE)

:Start
  WaitForKeypressMsg(Item %I%:%L%,-1,-1,FALSE)
  JumpToIfNumberIs(StopIt,%LCount%,FALSE)
  JumpTo(Start,FALSE)

:StopIt
```

Using %LCount% (number of items in the list) allows you to have all the items in the whole list displayed, however many there are.

WaitForKeypressMsg(Item %I%:%L%,-1,-1,FALSE) allows you to display the number (%I%) and content (%L%) of each item in the list.

Note: to display the picture whose name is listed in the list (at the current position), use DisplayPic(%L%,-1,-1,-1).

The content of the file named "v2_list.txt" is:

```
Ent1.jpg
Ent2.jpg
Ent3.jpg
pilori.jpg
studio.jpg
defile.jpg
```

detenu.jpg
dragon.jpg

The JPG pictures listed here should be found in the *Stimuli* folder.

5. Randomize sub-blocks of a list

The following script fills a list with 16 numbers (from 1 to 16) and randomizes two independent sub-blocks. Next, the content of the list is displayed, item by item.

; Fill the list with 16 numbers, from 1 to 16

```
:Fill_In
  AddToList(%I%)
  JumpToIfNumberIs(Next,16,FALSE)
  JumpTo(Fill_In,FALSE)
:Next
```

; randomize two independent sub-blocks. Items 1 and 2 are not involved.

```
RandomizeListRange(3,8)
RandomizeListRange(9,16)
```

; Display list's items

```
:Loop
  WaitForKeyPressMsg(Item %I%: %L%,-1,-1,FALSE)
  JumpToIfNumberIs(Stop,16,FALSE)
  JumpTo(Loop,FALSE)
:Stop
```

Note: the sequence “JumpToIfNumberIs(Next,16,FALSE)” then “JumpTo(Fill_In,FALSE)” (lines 3 and 4 of the sample above) has the same function as the command “LoopIfLabelIsBelow(Fill_In,16,FALSE)”: if the value of the “Fill_In” label’s counter is below 16, then jump to this label.

6. Use two label counters with one shifted

The following script shows how it is possible to count, starting with a value greater than 1.

```
SetLabelCounter(Start,2)
WaitForKeyPressMsg(Starting value: %I:Start%,-1,-1,FALSE)

:Start
  :Iteration
  WaitForKeyPressMsg(Iteration %I:Iteration% : %I:Start%,-1,-1,TRUE)
  LoopIfLabelIsBelow(Start,6,FALSE)
```

7. Have a list of words copied, with NetSync

The following script allows you to have a list of items copied by a group of participant. The trials are managed with NetSync (see Netsync in the User manual and, p.55).

```
DisplayMsg(Welcome !,2000,-1,-1,FALSE)
DisplayMsg(Please, wait a moment..., -1,-1,-1,FALSE)
; Wait for the start signal sent by the Master host
WaitForNetSync
HideMessage
ResetList
; Add items to the list, one by one
AddToList(pilori)
AddToList(studio)
AddToList(defile)
AddToList(detenu)
:start
; Display a star and wait for a “pen press” in a zone located on the left side of the tablet
DisplayPic(croix.bmp, -1,-1,-1)
WaitForTabZoneAt(1420,4900,4800,26530,FALSE,FALSE)
; Record the production in a file named <participant>_<item>.TAB
openrec(_%L%)
; Wait for 500 milliseconds
WaitFor(500)
; Send the number of item in the list (ex. “Item5/23”) to the Master host
SendMessageToNetSync(Item%I%/%LCount%)
; Display the item whose number in list match the “start” label’s counter value
; (item n°1 at first “round” - pilori, item n°2 for the second round -
; studio, etc.)
; Hint: the extension “.jpg » is appended to the item name to call a matching picture
DisplayPic(%L%.jpg, -1,-1,-1)
; When the pen is pressed in the writing zone, clear the display
WaitForTabZoneAt(4800,0,54200,31800,FALSE,FALSE)
HidePicture
; Wait for a “pen press” in the “end” zone. Participant’s writing is displayed on screen.
WaitForTabZoneAt(44930,23450,54200,31800,FALSE,TRUE)
CloseRec
; If the current item is the last one, jump to the “end » label
JumpToIfNumberIs(end,%LCount%,FALSE)
JumpTo(start,FALSE)
:end
DisplayMsg(Thanks for coming !,1000,-1,-1,FALSE)
```

Note : to “synchronize” the copy task item by item, i.e. to have the participants to wait before each trial (a collective “Go” is sent par the Master host to “unlock” the trial), the command

WaitForNetSync could have been inserted at the beginning of the “start” block, after a message saying to wait, like in the following example:

```
:start  
  
DisplayMsg(Be ready..., -1, -1, -1, FALSE)  
WaitForNetSync  
HideMessage  
  
; Display a cross...  
DisplayPic(croix.bmp, -1, -1, -1)
```

8. Dictation of a text

The following script allows you to dictate a text, previously recorded as an audio file (Wave format) and save into the *Stimuli* folder.

```
; Display the instructions contained in the text file named “Instructions.txt”.  
DisplayText(Intsructions.txt, -1)  
  
; Wait for the participant to press the pen in the zone labeled “Start dictation”.  
; Parameter: wait until the press is lifted up to go on  
WaitForTabZoneAt(25670, 20780, 29400, 18815, FALSE, TRUE)  
  
; Clear the instructions text  
HideText  
  
; Open the acquisition file <participant>_dictation.TAB  
OpenRec(_dictation)  
  
; Play the audio file and go on with the script without waiting  
PlaySound(dictation.wav, FALSE)  
  
; Wait for the pen to be pressed in the zone labeled « I’m finished » on the tablet.  
; The participant’s writing is displayed until this event.  
WaitForTabZoneAt(25970, 5565, 28900, 3915, TRUE, FALSE)  
  
; Close the acquisition file.  
CloseRec
```

9. Copying along with a mental load

The following script allows you to have a participant to copy a word after having heard a list of numbers. Next, he’ll be to recall (write down) these numbers.

```
WaitForKeyPressPic(wait.bmp, -1, -1)  
  
; Plays the audio recording of the numbers 1-3-8-9-5.  
Playsound(13895.wav, TRUE)  
DisplayPic(croix.bmp, 5000, -1, -1)  
  
; Open the acquisition file named <participant>_5Ejournal.TAB for the copy.  
OpenRec(_5Ejournal)
```

```
; Display the word to copy  
DisplayPic(E-journal.bmp,-1,-1,-1)  
  
; Write until the pen is presses in the tablet "end" zone  
WaitForTabZoneAt(6580,3830,7100,2270,TRUE,TRUE)  
HidePicture  
  
; Close the acquisition file and open a new acquisition file for the numbers transcription  
CloseRec  
OpenRec(_5EjournalCH)  
  
; Display the start signal to write numbers down.  
DisplayPic(B-rappelchif.bmp,-1,-1,-1)  
  
; Write until the pen is pressed in the tablet "end" zone.  
WaitForTabZoneAt(6580,1500,7100,0,TRUE,TRUE)  
CloseRec  
HidePicture
```