The E-Primer

An introduction to creating psychological experiments in E-Prime®

Michiel Spapé
Rinus Verdonschot
Saskia van Dantzig
Henk van Steenbergen

LEIDEN UNIVERSITY PRESS

Cover design: Mulder van Meurs, Amsterdam
Lay-out: V3-Services, Baarn

ISBN 978 90 8728 183 0
E-ISBN 978 94 0060 129 1
NUR 770/980

© Michiel Spapé, Rinus Verdonschot, Saskia van Dantzig, Henk van Steenbergen / Leiden University Press, 2014

All rights reserved. Without limiting the rights under copyright reserved above, no part of this book may be reproduced, stored in or introduced into a retrieval system, or transmitted, in any form or by any means (electronic, mechanical, photocopying, recording or otherwise) without the written permission of both the copyright owner and the authors of the book.

Acknowledgements

This book is the result of intensive collaboration between people who, at one time or another, studied or worked at Leiden University. In 2006, we extended a collection of E-Prime exercises into something resembling a coherent course. In subsequent years, we – initially as a rather dynamic collective of PhD students – have revised the manual in order to help those who want to prepare for creating their own psychological experiments. Later, we expanded the work so it can now also serve as a basic introduction to E-Basic coding in E-Prime.

Along the way, many people helped us in the writing of this book. First, we would like to thank David McFarlane, Michael Richter, Kerstin Brinkmann and the people at Psychology Software Tools for providing helpful comments when reviewing an earlier draft of this book. There were also numerous students who provided useful feedback during the period when this book was being used for courses at Leiden University. In particular, we would like to thank Hans Revers and Erwin Haasnoot for their constructive comments. Many more people have joined our journey, but at different moments in time. Each of us would like to mention a few:

Michiel Spapé: "I would like to thank Elkan Akyürek for introducing me to E-Prime and Jan-Rouke Kuipers for sharing my "pain" as we first started to teach it ourselves. Stephan Verschoor for being my favourite "unhappy victim" for demonstration purposes when no student volunteered – the nightmare of any teacher. Wido la Heij and Gezinus Wolters for providing my mind with many brilliant exemplars in teaching: easily retrieved from memory, yet so hard to successfully imitate, and Zania Sovijärvi-Spapé, for continuing to put up with a coding geek."

Rinus Verdonschot: "I am grateful to the Cognitive Psychology department of Leiden University for giving me the chance to teach E-prime to so many enthusiastic students and also to the people who participate actively in the Google E-Prime group and, thus, help out researchers all over the world. Lastly, a big thanks to my family, friends and colleagues."

Saskia van Dantzig: “I’d like to thank Diane Pecher, who introduced me to E-Prime and encouraged me to develop the E-Prime course for psychology students at the Erasmus University. This course supplemented several chapters of this book. Diane was also one of the reviewers of this book. Thanks to Rolf Zwaan for challenging me to program complicated experiments, which boosted my programming skills and let me explore the endless possibilities of E-Prime. Thanks to my former colleagues at Leiden University and Erasmus University. To conclude, thanks to Alexander for his encouragement and to my kids for enabling me to do the work I love. “

Henk van Steenbergen: “I would like to thank the many colleagues at Leiden University with whom I have been sharing E-Prime problems and solutions on a daily or weekly basis. Thanks to Guido Band, Bernhard Hommel and Sander Nieuwenhuis for encouraging me to publish this book. Thanks to Belle Derks for helping us to set up the revised E-Prime course, and Margot Schel for helping me to thoroughly revise and extend the predecessor of this book, resulting in the current end product. Anne Bolders provided great help with proof-reading. Thanks to Thijs Schrama for providing technical help along the way. Finally, I am grateful to Eveline and my family and friends for encouraging me to get this book published.”

Table of Contents

Introduction 11
What is E-Prime® and what will I learn? 11
Why should I learn E-Prime®? 12
Online support 12

Chapter I: E-Prime® at a glance 15
E-Studio, E-DataAid, E-Recovery and E-Merge 15
Object Oriented Programming 17
Conceptualising an experiment 19
E-Studio's structure 22
E-Objects 24
Procedures, Lists and TextDisplays 25
TextDisplays 26
Tutorial I: A simple RT experiment 36
Exercises 41
Advanced Exercises 43

Chapter II: Attributes, Slides and more on Lists 45
Experimental design in E-Prime® 46
More on lists 47
Showing an image 53
Showing multiple images and layers of texts 56
Feedback 56
Tutorial II: The Simon Task 58
Exercises 62
Tutorial III: Implicit Association Task 62
Exercises 64
Advanced Exercises 65

Chapter III: Sound, movies, hardware, and nested Lists 67
SoundOut objects and digital audio 68
MovieDisplay object 73
Wait object 74
Labels 75
Experiment properties 76
Nested Lists 79
Counterbalancing and between-subject manipulations 80
Tutorial IV: Visual search and distracting sound 81
Exercises 86
Tutorial V: Ego depletion experiment 87
Exercises 88
Advanced excercises 89

Chapter IV: Beginning programming in E-Prime® 91
E-Basic in E-Prime® 92
Different types of variables 93
Math operators 95
Mathematical functions 98
Combining numbers and text 98
String functions 100
Linking variables to input/output windows 101
Linking variables to attributes 103
Inlines everywhere 105
Tutorial VI: A working memory test 108
Exercises 111
Advanced excercises 111

Chapter V: Decision making in E-Basic 113
The If-Then statement 113
Combining conditional expressions 115
Comparing values 118
How to terminate parts of your experiment 119
Tutorial VII: The Ultimatum game 121
Tutorial VIII: The Cyberball game 128
Exercises 134
Advanced exercises 134

Chapter VI: Loops and Arrays in E-Basic 137
Looping with Labels 137
The For-Next Loop 139
Loops with conditional expressions 141
Lists and arrays 143
Arrays of a user-defined data type 149
Tutorial IX: Quasi-random trial selection 150
Exercises 159
Advanced exercises 159

Chapter VII **161**
Interactions between Slide objects and the Mouse 161
The Slide object 162
Accessing the mouse in E-Prime® 165
Programming user interactions 166
Tutorial X: A simple questionnaire 169
Tutorial XI: A mouse tracking task 172
Exercises 175
Advanced exercises 176

Chapter VIII: Various Input/Output devices 179
The Serial-Response box 180
Voice-key experiments 182
Sending signals using the parallel port 185
Reading and writing text files 190
Tutorial XII: Making a Voicekey (VK) test program 192
Exercises 194
Advanced Exercises 194

Reference List 197
Appendix: Overview of available E-Objects 199
About the authors 201
Index 203

Introduction

The E-Primer is written with a reader in mind who is eager to learn, but knows little, *if anything*, about programming, computer science and the actual implementation of all those wonderful scientific experiments that make up the body of the reading list of psychologists and cognitive scientists. This is not to say, however, that more experienced readers will not find it interesting as many chapters also deal with advanced E-Prime® and programming skills.

What is E-Prime® and what will I learn?

E-Prime® is a software package used to design and run psychological experiments, with a focus on psychological and cognitive science, and to acquire and analyse data. E-Prime® consists of a number of programs with different functions. In The E-Primer, we will discuss *E-Studio*, *E-Basic*, *E-Merge*, *E-Recovery* and *E-DataAid*. We will assume you use E-Prime 2, but most of the features we discuss were already introduced in E-Prime 1. When the two versions significantly differ in operation, we discuss each separately.

You will learn how to use each of these programs effectively in order to ultimately implement your very own experiments. First, however, you will learn how to re-create a number of fascinating, famous experiments. We guide you through this process in an easy to follow, step-by-step approach – 'now click on this button over there' – using *Tutorials*. Along the way, we explain why this is being done and try to communicate our insights on general good practice in design. Gradually, we will move beyond the narrow confines of the 'click-here-now-there' steps and ask you to implement simple variations. These form the basis of the *Exercises* at the end of each chapter. Finally, in the *Advanced exercises* you will be asked to explore, with minimal guidance, the horizons of E-Prime® and how you could pursue the effective realisation of your own research interests. Make sure that you save your work, because you may need it in subsequent chapters.

Why should I learn E-Prime®?

There are a number of reasons to learn E-Prime®. First, many students during their studies will become involved in doing a research project. This will eventually involve setting up and programming an experiment, which requires good E-Prime® programming skills.

Second, learning to program is more than just learning a specific programming language. Programming involves mostly logical thinking. Once you have learned to program in E-Prime®, you can easily transfer your knowledge and skills to new programming languages. Moreover, you will have learned to think about experiments in a structured and logical way. This skill is not only useful for setting up an experiment yourself (for example, during your research project), but it also helps you to read and understand empirical papers. Finally, we hope we can communicate some of our own enthusiasm and the idea that programming and realising your imagination, creating something out of nothing, is really a lot of fun.

Online support

E-Prime® has a good support website: *http://www.pstnet.com/support/login.asp*. Here, you will find examples of experiments and answers to frequently asked questions and problems. If you encounter a problem and can't find the answer on their FAQ and knowledge-base pages, you can send them your question via a special form on the website. You will then receive a personal answer, usually within a day or two. In order to get this personal support, you have to register (for free) on the website.

More information can also be found in the E-Prime® manuals. There is a short 'Getting Started Guide' and a more extensive 'User's Guide' and 'Reference Guide' (Schneider, Eschman, & Zuccolotto, 2002).You can also find more information on the STEP (System for Teaching Experimental Psychology) website. STEP is a web-based project designed to maximize the use of E-Prime®: *http://step.psy.cmu.edu/*.

This website includes ample examples of common paradigms in experimental psychology. It should be noted that these experiments are programmed in the previous version of E-Prime® (E-Prime 1). However, you can still open and run these experiments in the current E-Prime® version (E-Prime 2).

Lastly, an independently-run mailing list exists with a sizeable community that may be able to assist with urgent queries that simply can't wait for the E-Prime® support. Or, perhaps you have a more design-related question, or just want to let everyone know how much you love designing experiments in E-Prime®: *http://groups.google.com/group/e-prime*.

Chapter 1

E-Prime® at a glance

In this chapter, you will learn

About:
- E-Studio, E-DataAid, E-Recovery and E-Merge
- Object Oriented Programming
- E-Studio's structure
- Procedures
- Lists
- TextDisplays

How to:
- Create your first experiment
- Pimp your experiment
- Save and analyse your data

This chapter will introduce you to the E-Prime® software package. You will get acquainted with the different programs that enable you to create, run and analyse experiments. You will learn that E-Prime® uses **object oriented programming** to offer you different types of objects. These E-Objects function as building blocks that enable you to create your own experiments in a relatively simple and straightforward way. Before actually starting to program, it is important to visualise what your experiment will look like. Firstly, you will learn how to conceptualise your experiment, which makes the actual programming a lot easier. After reading this chapter, you should be able to program, run and analyse your own first experiment.

E-Studio, E-DataAid, E-Recovery and E-Merge

When we talk about 'working in E-Prime®', 'an experiment written in E-Prime®', or even 'E-Prime® crashed again', we generally mean E-Studio. You may be delighted – or disheartened – to learn that E-Prime® is actually a software package

composed of a number of programs other than E-Studio. We will talk about these other programs throughout this book, but, generally, they are straightforward and don't merit coverage beyond a quick summary.

E-Studio is based on, or perhaps merely inspired by, Visual Studio and can be called an 'IDE: an integrated development environment'. The graphical user interface is a convenient way to write (*develop*) code, simply by dragging and dropping objects onto a timeline. This makes the daunting task of developing experiments at least *look* as simple as using friendly and familiar Windows programs like PowerPoint. However, it is not quite true that no real programming (i.e. *coding* or *scripting*) is required: almost all original experiments at some point require the developer to write at least a few lines of code; and, more importantly, it can save a lot of time to do so. We will see how this works in later chapters. The experiment in E-Studio is stored as an **.es2** file.

E-DataAid is a program that can read E-Prime® output. Whenever an E-Prime® experiment is run, a unique datafile is created (an **.edat2** file). These .edat2 files can't be opened directly by Microsoft Excel or SPSS, but you can use E-DataAid to convert them into such formats. Additionally, E-DataAid comes with many additional features that make it much easier to get your data in proper shape for analysis. For example, you can filter out missing data before exporting, explore outliers and filter them out, generate crosstabs to base your graphs on, etc.

E-Merge does nothing more than merge data. Typically, when you have run *N* subjects, you will end up with *N* .edat2 datafiles. Of course, you can analyse each one in turn, or even import each one into SPSS, but this involves the risk that each action can go wrong, adding a chance of data-corruption due to human- or machine error; furthermore, each action costs time. With E-Merge you can merge the *N* datafiles into one large file. To merge a set of datafiles, take the following steps:

1. find the .edat2 files that your experiment has generated;
2. select them all using your mouse and control- or shift-clicking;
3. click on the *Merge* button.

This generates a **.emrg2** file, which can also be opened and analysed within E-DataAid.

E-Recovery is the smallest and simplest program in the package. If E-Prime® crashes during an experiment, no .edat2 file is generated. However, when the experiment is running (aka. runtime), a **.txt file** is created to which data from each trial are appended. This .txt file contains the same data as the .edat2 file, but it is rather inconvenient to analyse. So, should E-Prime® crash after having gone through several trials, you can open E-Recovery and take the following steps to recover the (partial) data:

1. press *Browse*;
2. look up the specific .txt file that you wish to recover;
3. press *Recover*.

Now you have an .edat2 file that is fully equivalent to the others, except that it lacks a few trials (if you are lucky). Since this is all E-Recovery does, the program will not be mentioned anywhere else in this book.

E-Run allows you to run experiments. When you have created an experiment in E-Studio and press *(control+) F7*, an **.ebs2** (encrypted e-basic script) file is created. The .ebs2 file can be run in E-Run.

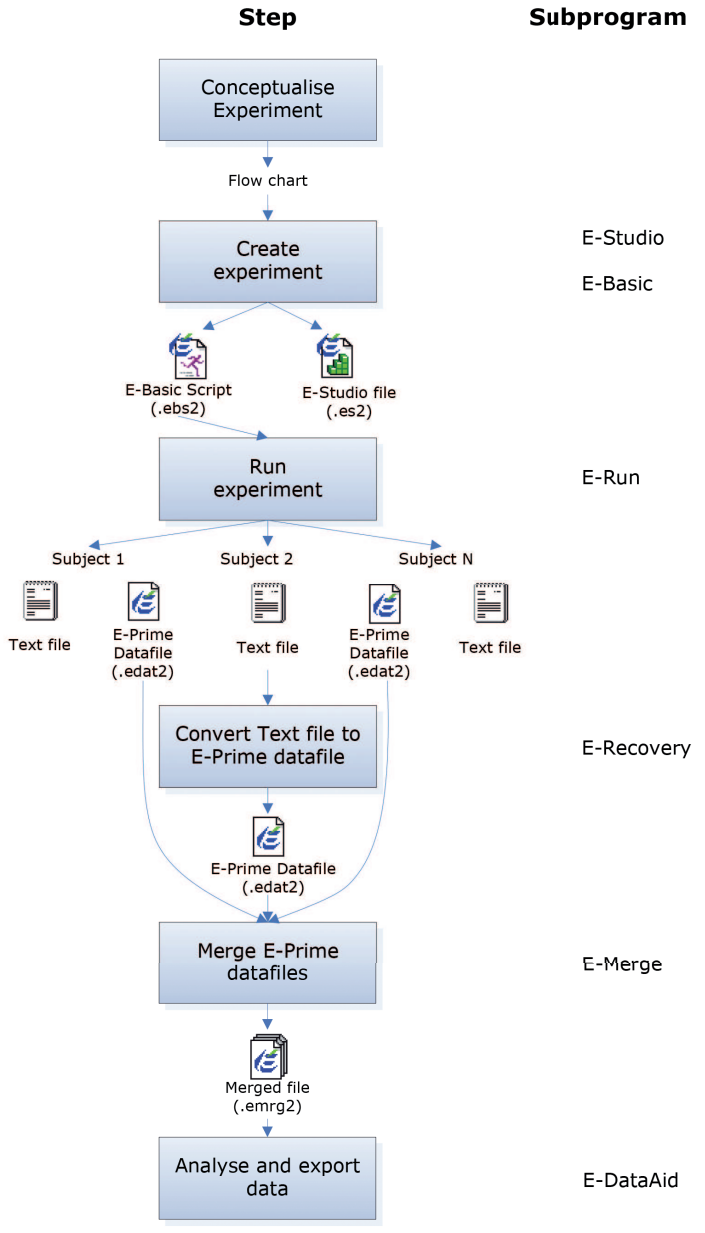
Object Oriented Programming

E-Prime®, like many popular programming languages such as C# and Visual Basic .NET, is based on the concept of **Object Oriented Programming** (OOP).

A good example of an **object** in daily life is 'a car'. One can do certain things with objects, such as *driving*, *steering* and *pursuing horizons*. In programming, we call such abilities **methods**. In programming script, an object's method is indicated in the following way: Object.method(parameters). For example, the code Car.drive(forward) would let the car object drive forward, the *Car* being the object, *drive* being the method, *forward* being a parameter of the *drive*-method. The other important feature of objects is that they usually have **properties**: a car can *be red*, *has a top speed of 200 km/h*, *has four seats* and so on. If we wanted to tell an object oriented programming language that our car is dark blue, we would say that our *Car.colour = dark blue*, *car* being the object, *colour* being the property and *dark blue* being the parameter of the colour property.

From idea to result

Designing, running and analysing an experiment with E-Prime software



Two other concepts of OOP are *instances* and *inheritance*. Our car, for example, is not just any car, it is OUR car! That is to say, *ourCar* is an **instance**;, in other words, a particular or token from the object or type *Car*. This matters for programming, because if we add something to *myCar*, such as a boom-box or mirror dice, this does not alter cars in general, but the same is not true if cars in general come equipped with such devices. **Inheritance** is also about the types and tokens: since our car is a Suzuki, it *inherits* certain properties and methods that are generally true for most cars, such as that it comes with a steering wheel and four wheels. Specifically, our car is a Suzuki Swift, which inherits certain features from the Suzuki object, such as its cheap price and uncomfortable seats.

Don't worry if these concepts strike you as difficult and abstract. Understanding them is not crucial to programming basic experiments, but since they are of such importance to modern-day programming, we hope that their functions will be revealed to you during your work with E-Prime®.

Conceptualising an experiment

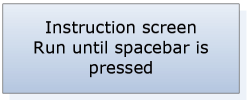
When you are designing an experiment you might be tempted to run to your computer and start programming straight away. However, before you start programming, you should try to visualise what your experiment will look like. This may sound self-evident, but it remains an important step that is easily omitted, with nasty, if not fatal, consequences ensuing. So, try to conceptualise the experiment by asking yourself the following questions:

- What kind of design do you need? A between-subjects design or a within-subjects design?
- Which variables do you manipulate? In other words, what are the independent variables? How many levels do these variables have? How many conditions does the experiment have?
- What are the dependent variables that you will measure? For example, do you aquire data regarding reaction time, error rates, and so on?
- Does your experiment contain blocks of trials? If yes, what is the order of presentation of these blocks?
- Does your experiment have a practice block?
- How do you instruct the participant?
- What happens during a trial? What kind of stimulus is presented? How long is the stimulus presented for? How should the participant respond? What happens if the participant responds too slowly?

- What happens between trials? Does the participant get feedback? How long is the interval between trials (we call this the **inter-trial interval**, **ITI**)?
- In which order are trials presented? In random order? In a fixed order? Or in a semi-random order?

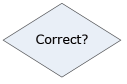
To facilitate programming, it can be useful to draw a flow chart that displays the structure of the experiment. An experiment typically contains a hierarchy of Procedures. The main Procedure (called 'SessionProc' in E-Prime®) determines the global order of events in the experiment. This one is depicted on the left. Sub-Procedures are depicted to the right of the main Procedure. A flow chart contains different elements:

A flow chart contains different elements:



Event: Refers to a specific event during the experiment, for example the presentation of a picture, text or sound. Indicates what happens, the duration of the event, and how the event is terminated (e.g. by pressing the *space bar*).

Sub procedure: Refers to a Procedure at a lower level of the hierarchy. Indicates the name of the Procedure, the number of repetitions of the Procedure and the order of the repetitions (e.g. random).



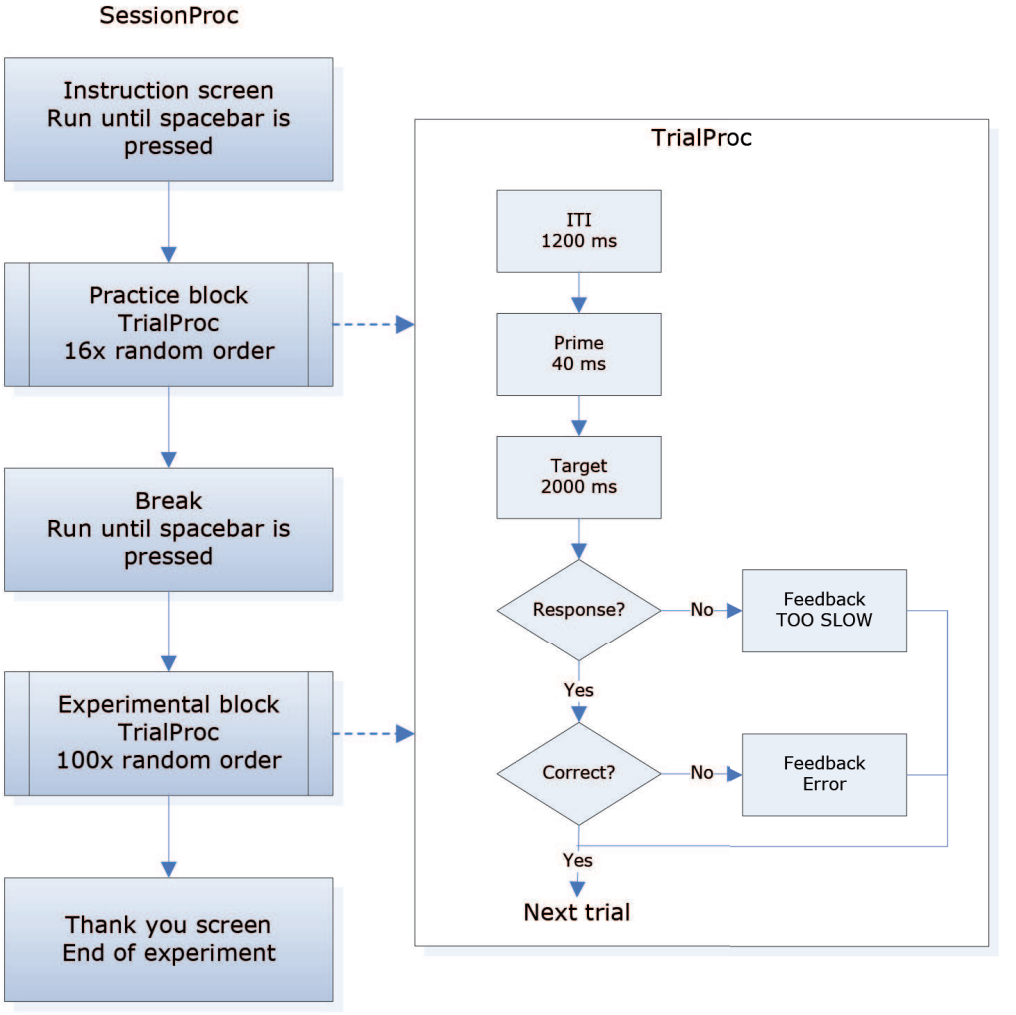
Decision: The Procedure branches into two options. The diamond indicates a criterion (e.g. response = correct, or reaction time < 1000). If the criterion is met, the Yes-branch is followed; if the condition is not met, the No-branch is followed.

Arrow: Indicates the flow of the events.

On the next page is a flow chart of a simple reaction time experiment with one practice block and one experimental block.

Note that if this way of visualising an experiment immediately strikes you as a great way to organise your thoughts into a workable design, then that is wonderful. However, if you feel it is a tedious amount of work that constrains your creativity by needlessly imposing order, then you might do best to save yourself the amount of paper involved. After all, some people prefer to work at a messy office desk. However, even if this is the case, we hope you will try to understand

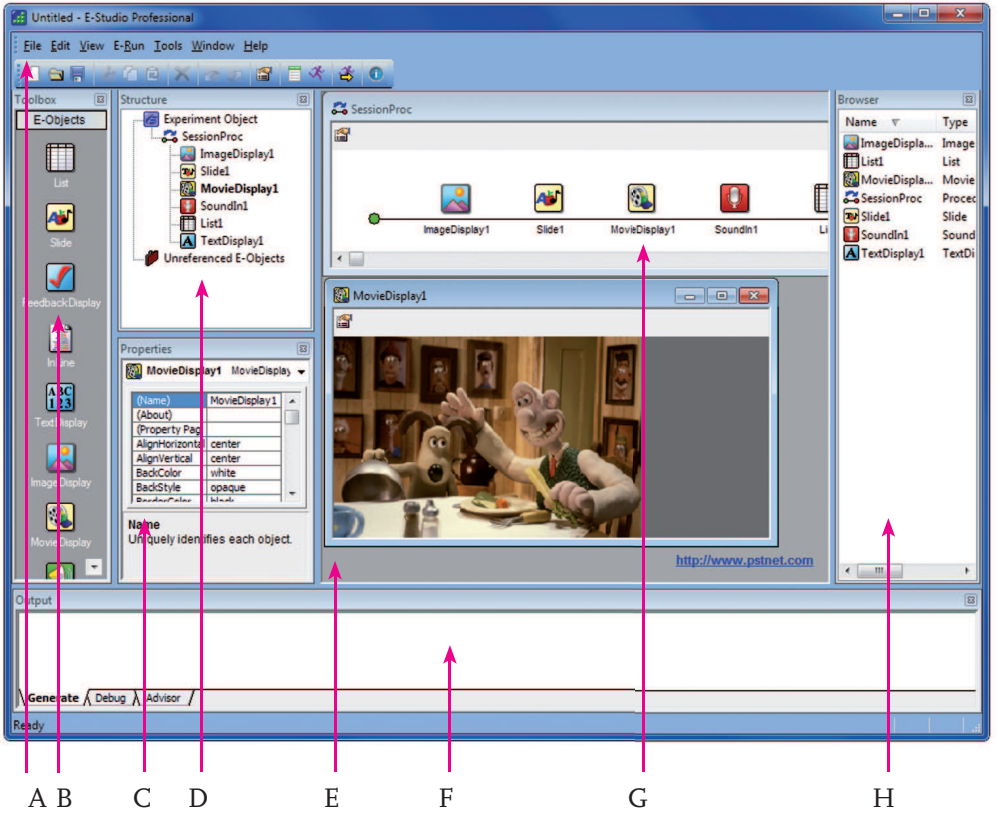
the endeavour as we will be illustrating various aspects of E-Prime® using these flowcharts.



Now, let's get back to the more practical work and have a look at E-Studio!

E-Studio's structure

Here's an example of what an experiment looks like when we open it in E-Studio.



A. In the **Menu** you can perform a number of typical Windows operations, such as opening and saving your experiment. In addition, by clicking on *View*, you can open the other areas (B – G and a few other ones). By clicking the *Run* icon – – or pressing *F7*, you can compile and run the experiment. Use the E-Run Test icon to run a quick test of your experiment (E-Prime 2 only; E-Prime 1 users may consider the **Clock.Scale** code described in Chapter IV).

To abort an experiment early, press **control+alt+shift** to **terminate** the E-Run application.

B. The **Toolbox** area shows all components (E-Objects) available in E-Prime®. To use one of them, drag it onto either the Structure area (D) or a **Procedure object** (G). Here is a favourite trick: right-click on them, unselect *Large icons* and *voilà*: more screen *real estate*!

C. The **Properties window** displays the properties of the currently selected instance. By selecting the TextDisplay, for example (see below), one can quickly change certain properties (such as the background colour) from within the properties area. Typically, you can also use the working area (E) for that, by clicking on the hand (see right), which tends to be easier.

D. The **Structure window** shows the hierarchy of the experiment. Most experiments are organised into **blocks** and **trials**. For example, you may want an experiment to have two blocks: one for training your participants, followed by one for testing them.

E. This area – which covers almost half the screen – is called the **Workspace**. This is where you can edit elements of the experiment in a visual, easy way. When you double click on an object in the Structure window, it appears in the Working Area.

F. The **Output window** appears when you click the *Generate* button in order to compile the experiment. If there is an error in your experiment, the output window will show a message describing that error. In *E-Prime 2 Professional* an Experiment Advisor is available for detecting design and timing errors.

G. The **Procedure object** called **SessionProc** displays a timeline with the main Procedure of the experiment.

H. The **Browser window** shows all objects (instances) that you have created for the experiment. In the Browser window you can copy objects. E-Prime® then creates a new instance with the same properties as the original object. You can also copy objects by dragging the object with your mouse while holding down your *ctrl* key. If you simply want to reuse the same object at a different location of the experiment hold down *ctrl+shift* while dragging.

E-Objects

E-Prime® contains different objects, each with its own characteristic features and purposes. Here is an overview of the objects that are used most often.

A **Procedure** is used to determine the order of events in an experiment.

A **List** contains rows of items with specific properties (attributes).

An **ImageDisplay** displays a picture.

A **TextDisplay** displays one or more lines of text.

A **MovieDisplay** displays a short movie clip.

A **Slide** is a container type of object which can simultaneously present text, images, sound and so on.

A **FeedbackDisplay** gives specific feedback on the participant's response to a stimulus.

A **SoundIn** is used to record sounds.

A **SoundOut** presents a sound file (.wav/.mp3/.wma).

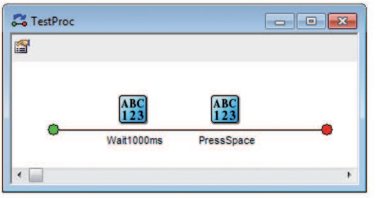
An **InLine** is used to add E-Basic script.

A **Label** indicates a particular location on the timeline. The program can 'jump' backwards or forwards to a Label, in order to repeat or skip a part of the Procedure.

A **PackageCall** contains reusable blocks of E-Basic script written by users of E-Prime 2 (often used in Procedures, which are used repeatedly or, for instance, in connecting equipment such as an eye-tracker to an E-Prime® experiment). As packages are beyond the scope of this book, please see the 'New Features Guide' for E-Prime 2 for a more detailed description.

Procedures, Lists and TextDisplays

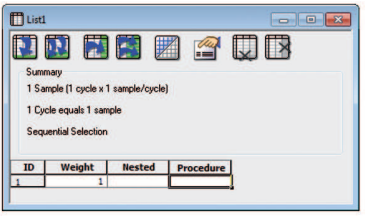
A **Procedure** is the highest unit in the hierarchy of E-Prime®. It is used to specify the sequence of events in the experiment.



A Procedure is depicted as a timeline. The green ball on the left indicates the start of the Procedure and the red ball on the right depicts the end of the Procedure. In this example, the Procedure called 'TestProc' presents two TextDisplays. First, it shows the Wait1000ms TextDisplay, followed by the PressSpace TextDisplay.

When you open a new experiment, it already contains a Procedure, specifying the order of events in an experimental session. This Procedure is called 'SessionProc' by default.

Lists are extremely useful objects. They repeat and reorder Procedures. Consequently, they determine the way in which Procedures are repeated, for instance, randomising certain variables that are contained in the List.

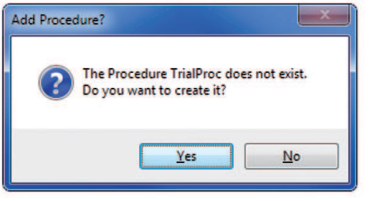


When you create a new List, you will see this window:

The rows contain different items, the columns indicate the properties (called **attributes**) of these items.

By clicking on the *Add Level* icon or the *Add Multple Levels* icon , you can add one or more rows, respectively.

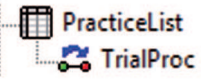
By clicking on the *Add Attribute* icon or the *Add Multiple Attributes* icon , you can add one or more columns, respectively.



Each list has a column named 'Procedure'. By filling in the name of a Procedure in a particular row, you specify which Procedure is used by that row. If the Procedure name doesn't yet exist in the experiment, the following pop-up window appears, telling you that the Procedure doesn't yet exist and asking you whether the Procedure should be created. Click *Yes*.

Subsequently, E-Prime® asks if this Procedure should be the default Procedure for newly created levels. Click *Yes* if you want all rows to use the same Procedure.

When you specify a Procedure, it will appear in the Structure window under the List containing the Procedure, as in the example (left), where *PracticeList* uses a Procedure called 'TrialProc'.



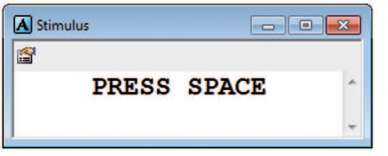
Lists are explained in more detail in Chapter II. For now, it will be enough that **Weight** indicates the number of repetitions of a particular item.

TextDisplays

TextDisplays present text in a singular formatting. This one shows the instruction to press the *space bar*. TextDisplays also offer a feature that may be even more important than showing text on the screen: they can collect responses. This, amongst others, is discussed in detail below.

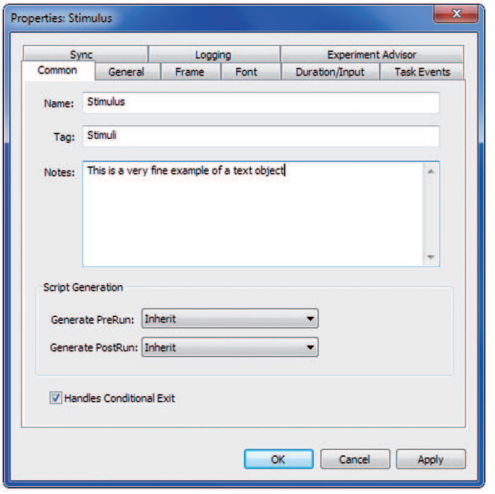
The TextDisplay is the simplest way of showing stimuli and collecting responses and almost the only object required to make a simple Stroop experiment (see Chapter II). Later on, we will see that other objects, such as the Slide and the FeedbackDisplay, can *contain* TextDisplays.

When you drag a TextBox from the Toolbox area onto a Procedure and double click on it, you should see something like the screenshot, with the exception that it is usually named differently and doesn't say 'PRESS SPACE'.



The TextDisplay's name is shown at the top left corner. When you add a new TextDisplay to the experiment, it will be named 'TextDisplay1' (or 'TextDisplay2', if 'TextDisplay1' already exists). It is good practice to rename the objects and give each of them a *unique and descriptive* name without funky characters such as commas, semicolons, spaces, etc. The above example admittedly shows how *not* to name an object: sensory presentations in experiments are stimuli by default, which makes this name not at all descriptive or unique.

Once you have created a TextDisplay, you can click on the *Properties* symbol to open the *Properties window*. This window has different tabs, allowing you to define various aspects of the object.



Common tab

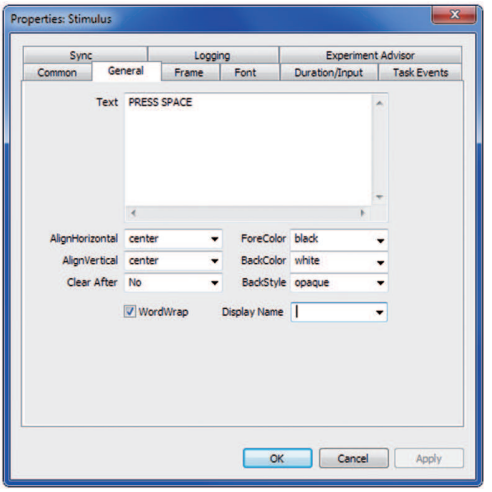
Name: The tag you give the object, which provides a handle for calling its properties. Note that some names are prohibited since they would interfere with generating script: names like IF, THEN, TextDisplay, and so on, should be avoided, as well as spaces and special characters.

Tag: Insert here – although we generally omit this step – an identifier for your TextDisplay.

Notes: Here you can write a description of your TextDisplay. This can be very useful if, for instance, you want someone else to work on your experiment and quickly give them an idea about what the object does, and why. In theory, commenting code is extremely important, as it is very easy to forget what your code does. However, it is probably best to design your experiment in such a way that deep scrutiny on the contents of Notes is redundant.

Generate PreRun/PostRun: These properties affect the moment of the .Load method. If they are set to TopOfProcedure, this TextDisplay's properties will be loaded at the beginning of the Procedure. This can provide a timing benefit when the object is supposed to be shown by displacing the loading requirements in time to a period that is presumably 'non-time critical'. However, note that this is only convenient if the object properties are fully known at the beginning of such a Procedure. If this is not the case, it is best to set the PreRun/PostRun to before/after object run.

Handles Conditional Exit: If enabled (which it is by default), this should provide a method to gracefully exit E-Prime®. It is true that *control+alt+shift* immediately shuts down the currently running experiment, but the immediacy of it can also be problematic: it doesn't close devices and no edat file is generated (but see E-Recovery). E-Prime 2 (but only Professional) users can press **control+alt+backspace** during the experiment and gracefully terminate experiments.



General tab

Text: Here you should enter the text that the TextDisplay will show. Generally, you may find it easier to adjust the Text property by using the more graphical interface shown earlier (the figure showing the 'PRESS SPACE' TextDisplay), but it is important to remember that *.Text* is a property of a TextDisplay, which can be used when you start writing your own script in the later chapters.

AlignHorizontal, AlignVertical: Adjusting these properties adjusts the position of the text relative to the horizontal and vertical dimensions, respectively.

ForeColor: The colour of the text. You can choose a fixed colour name, e.g. *red*, *green*, or *black* from the dropdown menu. Alternatively, you can enter an RGB (Red, Green, Blue) value: three numbers ranging from 0 to 255, representing the relative amount of red, green and blue. Thus, (255,0,0) means red, (0,255,0) means green, (0,0,255) means blue, (255,255,255) means white, and (0,0,0) means black. In this way, you can easily 'mix' your own shades: (12,188,180) is turquoise, for example.

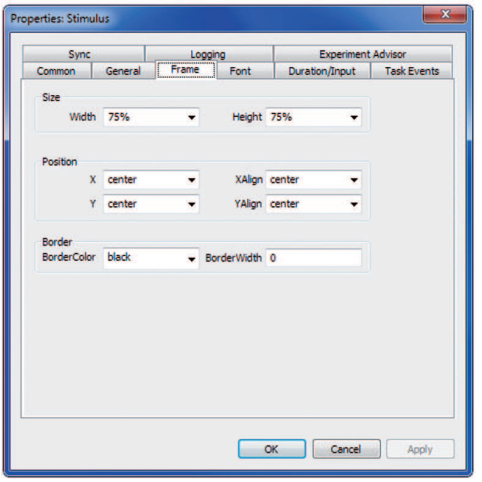
BackColor: The colour of the background. Specified in the same way as ForeColor.

BackStyle: The background colour may also be transparent; i.e. see-through. This is generally not very useful for TextDisplays, but for other objects, such as Slides, it might be practical.

ClearAfter: Specifies whether or not the screen is cleared after the presentation of TextDisplay. Usually, it doesn't matter what you specify here, because the Display is overwritten by the presentation of the next object anyway. It is also *deprecated*, according to Psychology Software Tools, by which software companies generally mean a function is still there to provide backward functionality, but should be avoided for it will be removed in a later version.

WordWrap: Specifies whether E-Prime® should automatically insert 'enters' when the text doesn't fit anymore. Without WordWrap on, E-Prime® will simply cut off the text where the screen ends.

Display Name: *E-Prime 2 Professional* adds the feature to use multiple displays independently. Here, you can select the display to use for this specific Stimulus.

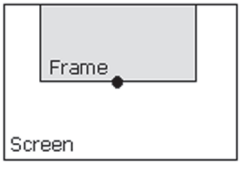


Frame tab

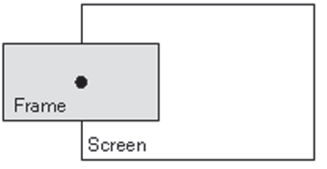
In the **Frame** tab, you can specify a rectangular area of the screen in which the object is presented. This area is called the 'Frame'.

Under **Size** you can specify the **Width** and **Height** of the frame. You can either specify the size relatively (in percentage of the total screen size) or absolutely (in pixels). Note that this is 75% by default as in the *Production Release* version of E-Prime® (2.0.10.242).

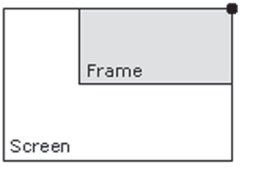
If the Frame is smaller than the screen, you can specify its position on the screen under **Position**. You can set four different parameters. **XAlign** and **YAlign** specify which point of the frame is used as a reference for placing the frame on the screen. **X** and **Y** specify the horizontal and vertical position of the frame's reference point on the screen. You can compare it to putting a piece of paper on a pinboard. The parameters **XAlign** and **YAlign** specify the position of the pin relative to the paper, the parameters **X** and **Y** specify the position of the pin on the pinboard (see examples below).



XAlign: Center
Align: Bottom
X : Center
Y : Center



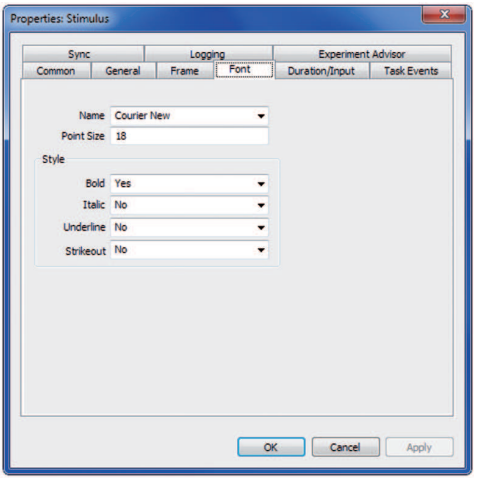
XAlign: Center
YAlign: Center
X : Left
Y : Center



XAlign: Right
YAlign: Top
X : Right
Y : Top

BorderColor: Shows the color of the border, if BorderWidth is greater than 0.

BorderWidth: With this property, you can set the width of the border around the TextDisplay in number of pixels.

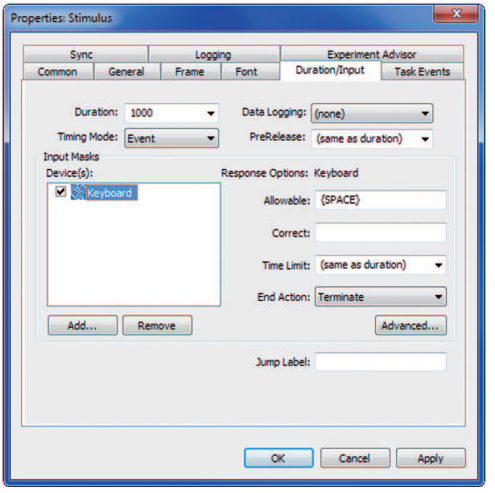


Font tab

Name: The type of font to use in this TextDisplay.

Point Size: The size of the font in points. This is the standard unit of font that is used in all Windows applications, but be careful: most experiments run in a lower resolution than normal, so fonts tend to look bigger when running the experiment.

Bold: Shows the word in a thicker typeface, which is pretty self-evident, but please note that E-Prime's default for Bold is *on*.



Duration tab

This is probably the single-most important tab. Here, you adjust the timing part of the stimulus, stimulus duration adjustments being crucial (some might say: tantamount) to classic cognitive experiments. In addition, it is where you select which input devices (such as keyboard, mouse, Serial Response Box) are used to record responses. The tab also deals with most other aspects of responding: 'what should happen after a response?', 'what was the correct answer?', etc.

Duration: With this property, you can manipulate how long the TextDisplay is presented on the screen. When you set this to -1, it acts the same as when you select the infinite duration.

Timing Mode: E-Prime® is praised for its timing accuracy and its developers claim that E-Prime® can have sub-millisecond accuracy (i.e. have random timing errors with a standard deviation of less than one millisecond). However, this all depends on which other processes are running in the background, which hard-

ware is installed, and whether unrelated software like Norton Antivirus is allowed to run alongside E-Prime®.

More on timing issues later, but for now, here is the basic story. It takes time (mere milliseconds, or even less) between the moment when E-Prime's clock notices that an event should be triggered and when that object is actually presented. Because of this, events may not synchronise with time, which E-Prime® calls 'cumulative drift'. To prevent this, you can change the TimingMode to Cumulative, which changes the duration of this TextDisplay to adjust for this drift. Which TimingMode you should use depends on your experiment. If your experiment consists of relatively long inter-stimulus intervals (say, a few seconds) and timing is not crucial, using the Event Mode may suffice. However, if your experiment depends critically on timing, such as when presenting subliminal stimuli with a duration of 20 ms, you may need to use the Cumulative Mode. However, Procedures using objects in this mode can behave oddly and unpredictably, especially when they include *Terminate* End actions (see description below) or scripts. When timing is critical, you should refer to the chapter on 'Critical Timing' in the 'User's Guide'.

The other way to cope with a certain type of timing error is the **PreRelease**. Suppose you wish to present high-resolution images, perhaps even in a rapid serial visual presentation task. In this case, E-Prime® will have a hard time loading all those large image files. In order to alleviate the stress E-Prime® puts on your processor, you can use TextDisplays prior to the pictures you want to use and set some PreRelease to that TextDisplay. This PreRelease time is used to load the oncoming picture, sound or other 'heavy' object into memory while the current TextDisplay is still shown. Then, when it is show-time, E-Prime® has already loaded the object, thereby decreasing onset errors in the next stimulus. If timing is critical in your experiment, we recommended that you study the 'Critical Timing' chapter in the E-Prime® 'User's Guide'.

Notice that, following the *Production Release* of E-Prime 2, the PreRelease is always set to: *(Same as duration)*. While this will make it much less likely that your experiment will have timing problems, the degree to which subsequent objects will be run *before* the PreReleasing object ends can make your life difficult. In particular, care should be taken when 1) the next object on the timeline is an InLine or a PackageCall; 2) the object with PreRelease is the last object on the timeline; or, 3) the next object is a FeedbackDisplay.

The **Data Logging** property has a few options allowing you to log various timing and response parts of the TextDisplay. We would suggest leaving this untouched and selecting the logging properties in the *Logging tab*, as E-Prime® generally logs far too much if you allow it to. However, we once observed the reaction of someone who had logged everything BUT the critical response times and came up with a convenient rule of thumb: it is better to log too much than too little.

If you want the participant to respond to the TextDisplay, you will have to add an **InputDevice**. To do so, click on *Add* and select *Keyboard* or *Mouse*. More devices (such as the Serial Response Box) can be available, but you will need to add them first by clicking on *Edit* > *Experiment* > *Devices* > *Add*.

After an InputDevice is selected, you can edit which keys are **Allowable**. Normally, you enter a range of characters here, for example: Allowable: abcd. Then, all four keys (a, b, c, d) are seen as valid responses. Pressing *e* or *A* (*shift+a*), for example, will not do anything. If you want to use the space bar or other special keys, you will have to use the round brackets and *capitalised* letters: *Allowable: {SPACE}* for example. The default ({ANY}) is not recommended for a serious experiment, since accidental key-presses should not be counted as 'real' responses.

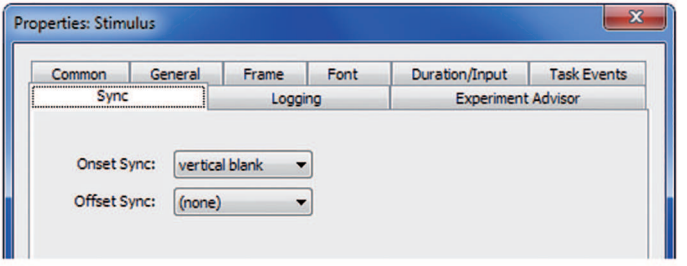
Whereas the Allowable part generally has several options, usually only one of them is **Correct**. It is important to understand the *fundamental difference* between the allowable and the correct response. As a rule of thumb, then: the allowable set of responses covers the range of possible responses and *is generally the same for each trial*; the correct response is typically only the one response that the participant *should* have made and *is generally different for each trial* (although one can, in E-Prime 2 Professional, have multiple correct responses). It is probably best to think of 'correct response' in terms of accuracy, rather than appropriateness. Both in the case of {ANY} response being allowable, and if only one key is allowed, the accuracy of the response becomes pretty meaningless.

The correct response doesn't have to be specified. For example, a welcome screen doesn't have a correct response. It can simply be closed after pressing a specific key. On the other hand, the allowable response should always be specified. If you have failed to do so, and the duration is set to infinite, your experiment will get stuck, since the participant can't press any key to close the TextDisplay.

By adjusting the **Time Limit** property, you can increase or decrease the amount of time, following the onset of the stimulus, in which a response is logged. Often, this will be the same as the **Duration** of the stimulus, which is the selected option by default. That way, if the duration of a stimulus is 2000 ms, a response will still be logged when it follows 1999 ms after the onset of the stimulus. However, it is possible to ignore extremely late responses (outliers), by setting the Time Limit to 1000 ms. A response that follows 1100 ms after the onset will then not be logged. It is also possible to log responses even longer than the duration of the stimulus. If, for example, you use a subliminal priming paradigm, you could set the Duration of the stimulus at 20 ms, but the Time Limit at 1000 ms. Then, responses are logged relative to the onset of the subliminal stimulus, even if it is no longer being shown.

End Action specifies which action to undertake when the participant responds. By setting this to *Terminate* (default), for example, the TextDisplay is immediately wiped off the screen when an allowable response is given. The **Jump** option will be discussed in a later chapter.

Sync tab



The Sync tab enables you to switch on- and offset synchronisation. To understand what synchronisation is about, you must grasp a basic fact about both *cathode ray tube* (CRT – or, the 'old' type) and *liquid crystal display* (LCD – or, the 'flat' type) computer monitors: each dot (pixel) you see on the screen is updated sequentially: that is, from up to down. Although it may look as if the pixels you see on the screen are static (especially with LCD monitors), in fact they are updated at a rate of at least 60 times each second: 60 Hz (or about 100 Hz when using ancient CRT monitors, which are actually better for doing experiments). Crucially for psychologists, especially those working with perception and (subliminal) priming, the presentation of visual stimuli is constrained by the timing characteristics of the monitor.

Consider, for example, a psychologist who wants to show a subliminal prime, say, a smiling face, 10 milliseconds before the onset of a word to which the participant is required to react. When **Onset Sync** and **Offset Sync** are turned off, and the experiment is running on a 60 Hz monitor (still a standard especially in LCD), the

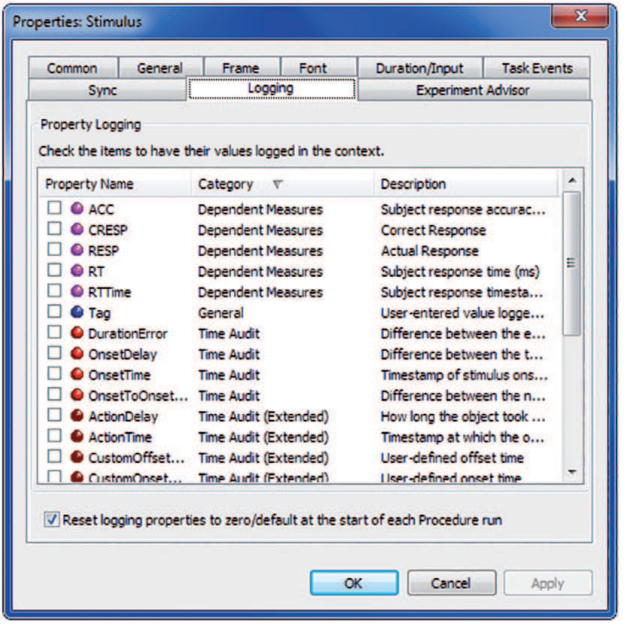
screen is updated every (1000 / 60 =) 17 ms. When E-Prime® is programmed to show the smiley, it sends 'commands to the screen', but there is no way to know exactly how and what it shows: it may show everything 17 ms later, for 17 ms, because it just finished updating the screen; but it may also show only half a smiley because the updating cycle just got round to half the screen on its up to down route. This phenonomenon is called **screen tearing**. The same may then happen with the word to be responded to. However, if we enable Onset Sync for both the prime and the word, E-Prime® will wait until the screen is able to show the word fully. The only problem then is that it must show the prime for at least as long as the refresh cycle (17 ms) lasts, thus creating a timing error here of 7 ms as it is impossible to show anything for less than 17 ms.

For exactly this reason, we recommend the use of (often ridiculously old) monitors that have shorter refresh cycles (100 Hz – 10 ms; which gives nice, round numbers), so we can safely use Onset Sync by default. However, obtaining these monitors is becoming increasingly difficult, *sic transit gloria mundi*.

Note that the Onset Sync is set to *vertical blank* by default. This helps avoid screen tearing, an issue we will explain later when discussing display hardware.

Logging tab

If, like us, you love reaction times and other chronometric measures, E-Prime® is the thing for you. Not only does it let you collect standard outcome measures, such as response, accuracy and reaction time, but it also provides an arsenal of auditing weaponry to bedazzle even the most number-crazed statisticians. For example, if you wish to check whether E-Prime® *really* presents your stimuli for *t* milliseconds, you can **log** the duration error.



You can select as many values to log as you like, but try to be somewhat pragmatic: you may want to log the time it took for a participant to read the introduction-screen, so you can log RT for this display. On the other hand, we have yet to hear from a psychologist who is interested in the timing accuracy of this introduction-screen, so you don't generally log OnsetDelay, for instance.

Often, a psychological experiment requires only one response for each single trial. For example, in a Stroop task, each displayed word requires one reaction. This translates, *e-wise*, in that the text display, which displays the word 'WHITE', collects responses (such as the correct answer: black). For this object, favourite logging properties would then be:

- **CRESP**: Correct response. As stated above, typically depends on the condition and trial.
- **RESP**: The actual response.
- **ACC**: The accuracy, defined as 1 if the RESP and CRESP are equal and otherwise 0.
- **RT**: Reaction, or response time (ms), which is RTTime – OnsetTime.
- **OnsetDelay**: Difference between programmed time the stimulus was to be presented (in the case of visual stimuli: on the screen) and the actual time its presentation started.
- **DurationError**: Difference between the prescribed duration the stimulus was to be shown on screen and the actual time. Or strictly speaking: OffsetTime + PreRelease – OnsetTime – Duration.

Which logging properties you should use, depends on your experiment. Three other valuable logging properties are:

- **RTTime**: Time stamp of the reaction relative to the beginnig of the experiment (ms).
- **OnsetTime**: Time stamp of stimulus onset relative to the beginning of the experiment (ms).
- **OffsetTime**: Time stamp of the end of the presentation of the stimulus (ms) relative to the beginning of the experiment. Notice, however, that this is not necessarily when the stimulus ends: a visual stimulus remains 'on the screen' as long as no other stimulus overwrites it, and an audio file of 4 seconds long may contain 2 seconds of silence.

Tutorial I: A simple RT experiment

Do you, like Michiel's cat (right), have 'lightning reflexes'? Believe it or not, many first-time participants who are unfamiliar with psychological experiments want to know 'how well they did' and considering that your first priority is most likely not pinning someone on a kind of normal (vs abnormal!) distribution, it is always good to tell them they were '*quite fast...*'. Let's find out how to do this!

It can be hard to start programming an experiment from scratch, so you may find it easier to follow a certain process schema. First, and crucially, what is it that you want your participants to see during an experiment – what do you know from your own experiences with psychological research in the lab? Imagine *examples*, rather than defining everything beforehand: instead of trying, for example, to show Stroop-like stimuli, ask yourself: 'so what is a Stroop-like stimulus?' 'Well', you answer, 'something like the word *red* written in *blue*'. Then, define the Procedure of a trial as the sequential presentation of such stimuli.

Common elements of an experiment include:

Trials: Typically, this includes:

- A fixation: This stimulus that is often shaped like a crosshair or addition sign 'warns' the participant that the interesting stimulus is approaching.
- The target: The interesting stimulus itself, to which the participant is to respond.
- Some form of feedback (occasionally).

Blocks: Are defined by the number and variant of trials they contain. For instance,

- A training block may contain some 20 trials and is used to get the participant accustomed to the experiment.
- A testing block contains more trials, depending on the variability of the outcome measures, the number of conditions, etc.

Step 1: Building the basic hierarchy

- Open E-Studio, select *Blank experiment*.

- Save your experiment in a location where you can find it again easily (e.g. a USB stick, your personal drive, etc.). Give the experiment a unique name that doesn't contain weird characters (slashes, dots, etc.).

- Make sure you always save your work. Keep your file structure well-organised! Subsequent chapters may ask you to re-use part of your earlier work. There is an additional advantage: by saving your work you also start a personal collection of experiments that may serve as a source for future reference.

- In the Structure view, double-click on *SessionProc* ; you will see a timeline popping up:

- Drag a List from the Toolbox to the SessionProc to the timeline and call it 'BlockList' (this is a conventional name; you can also use any other name as long as it doesn't contain strange characters or spaces).

- Double-click on the *BlockList* and add one row by clicking on the icon of the arrow pointing down.

- Change the name of the Procedure column of the first row to 'TrainingProc' by editing the text. Please note that it is also possible to click on the down triangle next to the name and change the Procedure to an existing one: SessionProc. *DO NOT DO THIS!* For some reason, this seems the most intuitive action and we have seen many students astonished at how fatally E-Prime® crashes when this seemingly minor mistake is made.

- E-Prime® will ask you whether you really want to create this new Procedure – TrainingProc – and here you select *Yes*. If E-Prime® asks you whether you want this Procedure to be the default one, select *No*.

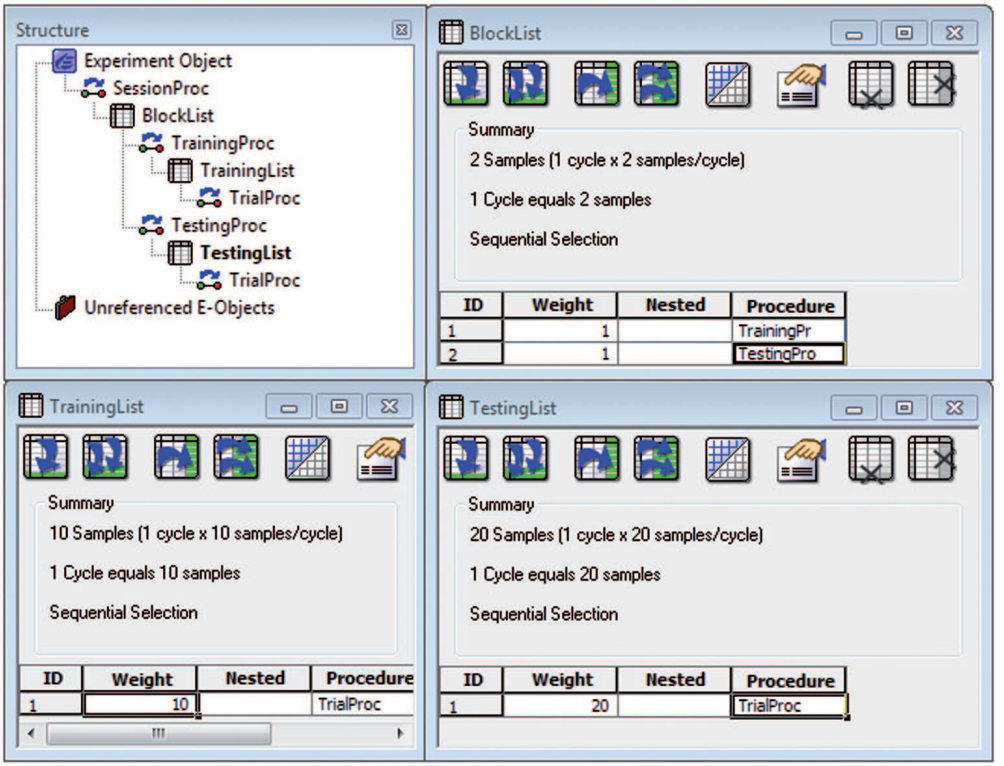
- Change the name of the *Procedure* column of the second row to 'TestingProc' and repeat the previous actions (*Yes, No*). Notice that *not* creating the new procedure will result in a bug.

- Double-click on the *TrainingProc* and add a List there. Rename (select and press *F2*) the List as TrainingList. Double-click on the *TestingProc* and add a List there, rename it as TestingList.

- Edit *TrainingList* and make the weight of the first and only row 10. In the Procedure column, write down the name 'TrialProc'.

- Edit *TestingList* and make the weight of the first and only row 20, then write down the name 'TrialProc' as its Procedure.

- Now you have the basic hierarchy of an experiment: one experiment with two blocks, one for training and one for testing, and the two blocks run the same Procedure; the training 10 times, the testing 20. You can check whether you successfully completed this step by matching your screen with the screenshot below.



You could also consider dropping the *BlockList* and simply running the *TrainingList* and *TestingList* consecutively in your *SessionProc*. However, we think that the BlockList solution has many advantages. For example, while testing whether your experiment runs correctly, you can easily skip parts of it by setting the weights of the respective rows to 0. Moreover, if you want to abort your experiment with *Inline scripts*, you can do so by simply terminating the BlockList (see Chapter V). The BlockList is also a proper place to nest Lists used for counterbalance/between-subject manipulation purposes (see Chapter III). The attributes of these *nested Lists* are then automatically inherited by Lists lower in the experimental hierarchy.

Step 2: Programming the trial

- The trial is perhaps the most important unit in your programming experiments. Here, you will be showing your participant a fixation for 500 ms, and a target for an infinite – or until key-press – amount of time.

- Double-click on the *TrialProc* and drag two TextDisplays to the timeline. Name the first 'Fixation' and the second 'TargetStimulus'.

- Edit the Fixation to show a single '+' sign and to have a duration of 500 ms. This is our 'inter-trial interval' (ITI).

- Edit the TargetStimulus to show the command 'Press space!' and to have an infinite duration. Then, add an input-device by clicking on *Add* in the *Duration/Input tab*, and choose *keyboard*. Set, as the only *Allowable* key, {SPACE}. Mind the capitals, they are important here. Also, set the spacebar as the only *correct* key (this is generally not the case!) and accept the standard type of logging.

- Your experiment should run now, so try this. It is good practice to run your experiment *frequently*, because this makes it easier for you to diagnose, or **debug**, problems. Run your experiment with any subject number but 0, or else nothing is logged. Remember: if you want to abort the experiment quickly, you can always press *control+alt+shift* (or perhaps *control+shift+backspace*). When starting your experiment, the resolution of your screen may change. In Chapter III we will discuss how to change these screen settings.

Step 3: Analyse the data

- When you have finished testing the experiment, start *E-DataAid* and open the data you generated. These can be found in the same folder where your experiment was last saved.

- Scroll through your data and note the various columns. For example, notice how the trial number starts at 1 and goes on to 10, because the first block (the TrainingList) is finished after 10 trials; then starts at 1 again but now goes to 20, because the second block (the TestingList) is finished after 20 trials.

- Since we want to know what your participant's average basic reaction time was, the TargetStimulus.RT is most important to us. Notice how several values will be well below (approaching an unlikely 0) and above (in case you were distracted) the average. Apparently, a bit of filtering needs to happen to get a clear picture of your RT in comparison to your neighbour's.

- Click on *Tools*, select *Analyze* and click on *Filter*. In the dropdown box, select *TargetStimulus.RT* (in alphabetical order here) and click on *Checklist*. Now, click once on the first value that is higher or equal to 100, then scroll down, and *shift+click* on the last value that is lower than 1000. Only then, with all the values you want to include selected, press *spacebar* and click on *OK*. In this way you prevent outlying RT values from distorting your mean RT values.

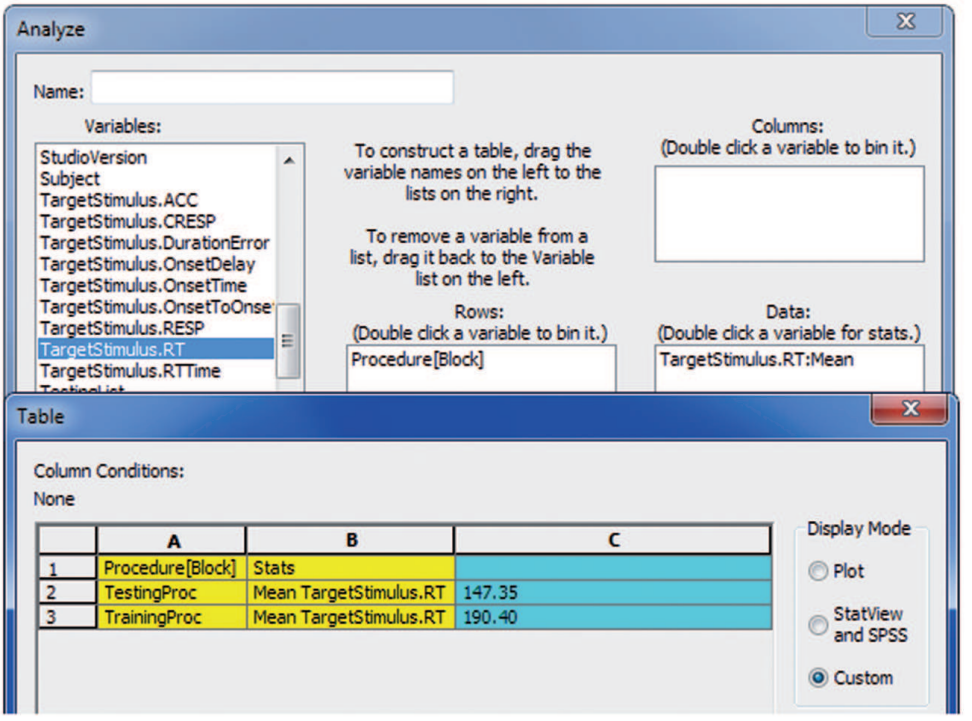
Consider what would happen if you save this analysis and apply it to another dataset later. In this case, it is quite likely that there are new unique RT values in your dataset not yet included in your checklist. In other words, you have to reselect the relevant RTs. In those cases it might be preferable to use the Range alternative. Click *Range...* and set the first range to *Greater than or equal* 100, combined with the second range being *Less than* 1000. Don't forget to select the *AND* operator, since our inclusion criterion is that each single RT needs to meet both conditions.

- So now that we have deleted the outliers from further analysis, close the filter and drag TargetStimulus.RT from the list of variables to *Data*. Click on *Run* and get ready to be astounded by your reaction time! Michiel's was 191 ms.

One of the reasons why many people use E-DataAid in conjunction with E-Prime® is the ease with which you can make crosstabs. Here is how we do it:

- Close the analysis results and, without changing anything else, drag the Procedure[Block] variable from the list to either the row or the columns (try both). Again, click on *Run*.

This is what it should look like:



- So, Michiel was about 43 ms faster after training a bit.

Exercises

- Add an introduction screen to the start of your experiment, with infinite duration, terminating when the participant presses a certain unique key ('press C to continue').

- Add a goodbye screen and a thank you screen to your experiment.

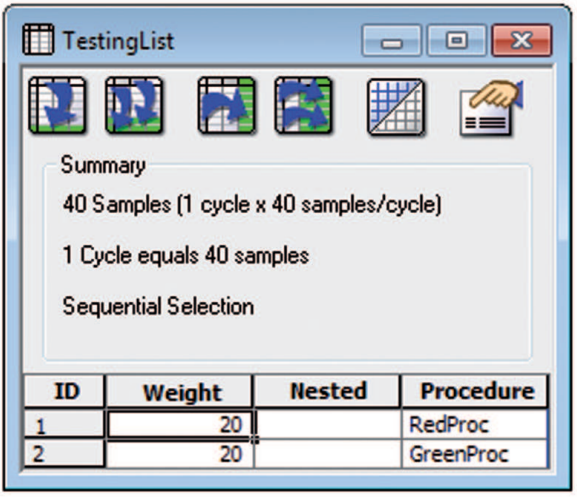
- Use the mouse instead of the keyboard as InputDevice for the TargetStimulus. To do this, you basically do the same as you did with the keyboard as an input device, except that the response keys are defined as 1 (left mouse-button) and

2 (right mouse-button). Therefore, if you enter '12' as allowable both buttons are allowable responses.

- Pimp your experiment: adjust it to your taste or to what you think would be wise; just experiment with all the options.

- Design an experiment to test the following hypothesis: it is easier to respond to green than to red. The idea is clear: typically, we need to stop doing something when a red light appears, so a psychologist could hypothesise that because we internalised this rule and thus suppress all action when a red light appears. It is time to find out whether this is true.

You can base this experiment on the one you made in the tutorial. First, the trial needs to be changed: the fixation should now have a gray background; the target should have no word anymore, but just be a coloured background. The TrainingList should now have *white* targets. The TestingList should now have *green* and *red* targets. Therefore, the TestingList should get one extra row, so that there are two different procedures. Instead of having the TestingList call "TrialProc", let it refer to "RedProc" and "GreenProc":



At this point, you might think: would it not be much easier to *reuse* the same Procedure but only change one aspect – or variable – of the trial? In the next chapter, you will find out that this is indeed the case and you will learn how to do this using *attributes*.

Also, randomise the selection between these two Procedures (see the List properties) so that the participant can't know in advance what colour the target will be. Edit both the RedProc and GreenProc equally so that they are exactly the same in terms of look, duration and response, except that the background of the target differs.

Test your experiment: is it easier to respond to green than to red?

Advanced Exercises

- Instead of analysing all data separately for each participant or cutting and pasting data into Excel from various edat (.edat/.edat2) files, one can simply use E-Merge to merge edat datafiles together into one big file.

- Make sure you have more than one edat datafile (run your experiment with different subject numbers) and put them in a directory. Start E-Merge, go to the directory containing the files using the folder tree and first filter that directory so you can only see the edat files and not anything else (press the *Filter* button) and use *.edat or *.edat2 (the * wildcard means everything).

- Select the *.edat* or *.edat2* files and press the *Merge* button (use the standard merge). Now you will see that an .emrg or .emrg2 file has been created containing all the information from the two separate files.

- Now use E-DataAid to analyse all your participants/sessions at once.

Chapter VII

Interactions between Slide objects and the Mouse

In this chapter, you will learn

About:
- Common properties and methods of the Slide object
- Accessing the mouse in E-Prime®
- Programming user interactions in E-Prime®

How to:
- Program a simple questionnaire
- Program a mouse tracking task

Dear reader, let's assume you have got all the way through six chapters full of useful information and are now well able to code your very own reaction time experiment. However, in order to dazzle your peers even more with something more beautiful, interactive and indeed, 'flashy', in this chapter we will start to go beyond the very basic 80s type of cognitive experiment and will include fancy new hardware, like the mouse, and aspects of more qualitative research; that is, the questionnaire.

It is important to remember that E-Prime's greater strength lies in critical timing and interaction with apparatuses that require such timing, like EEG or eye-tracking. So, when you find yourself mainly involved in questions such as 'how 'betrayed', on a level of 1 to 5, does our participant feel?', or want to test efficacy of immersive user-interfaces combining drag-and-drop behaviour for mobile communication technology, E-Prime® may not be the platform of choice. For indeed, the simple Windows event of 'the user clicked on button A' is not defined within a second of work; whereas web-questionnaires are incredibly easy to make using simple websites.

However, we often find it useful (or sometimes just fun) to add aspects of interactivity, say, one or two clickable objects, to our well-timed experiments. Also, it can be useful to have all the data – including questionnaires – regarding one

participant in one place. It certainly beats the alternative: the ultimate boredom of transposing the pen-and-paper data into SPSS!

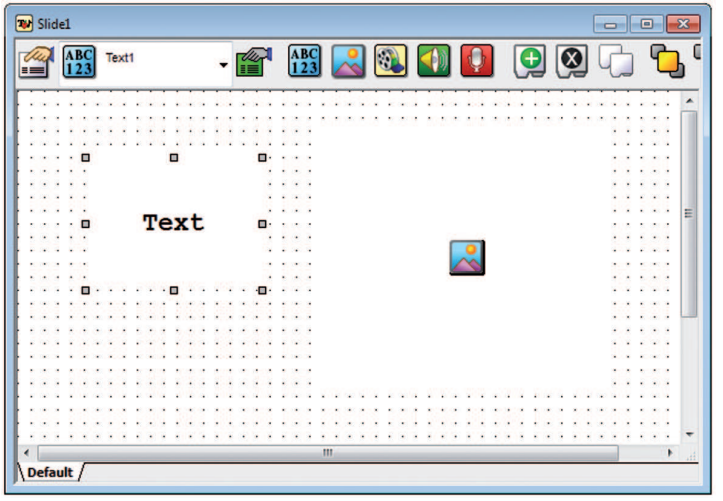
Therefore, in this chapter we will show you how to engage Slide objects to a deeper extent, how to create a simple questionnaire using E-Prime® and, finally, how to do 'fun stuff' with the mouse.

The Slide object

Slide objects are extremely useful because they allow you to combine text, images, sounds and movies in one and the same object. In this chapter we will learn to access the Slide, including its SlideStates and sub-objects using InLine codes.

Remember that E-Basic is an object-oriented programming language, and that you can refer to properties and methods of particular objects using the *dot operator*. Likewise, you can also read or change properties, and call methods of Slides and their sub-objects.

Let's first consider the hierarchical structure of a Slide object. The figure below shows a Slide object ('Slide1') with a particular SlideState ('Default') including two sub-objects: a SlideText ('Text1') and a SlideImage ('Image1').



When a Slide is created, E-Basic automatically declares a hierarchical object consisting of the following elements:

Slide1 As Slide
    Default As SlideState
        Text1 As SlideText
        Image1 As SlideImage

For each level in the hierarchy we can access particular properties and methods in E-Basic. A full Listing of all properties and methods can be found in the E-Basic help file. In the subsequent sections we will describe some common examples for each level in the hierarchy.

Slide object: Properties and methods

To read or change the ActiveState we simply refer to the property:

```
Slide1.ActiveState
```

Similarly, if we want to call up the Procedure that manually draws the Slide1 object on the screen (application to be discussed shortly), we may use the method.

```
Slide1.Draw
```

SlideState object: Properties and methods

At the SlideState level, we can control several settings defining its display frame, colours and border. For example, to change the BackColor in the state "Default" of the Slide object "Slide1" to *red* (using a CColour conversion function), we can use this code:

```
Slide1.States("Default").BackColour = CColour("Red")
```

Note that we refer here to the subordinate SlideState "Default" via the property States of the Slide object. The code demonstrates how to access the nested object in E-Basic.

The method HitTest is typically used in conjunction with mouse input. If you provide some coordinates in pixels, the HitTest method will return the String name of a sub-object of Slide (e.g. a SlideImage or a SlideText) at the specified coordinates. If no SlideImage or SlideText object exists at the specified coordinates, an empty String "" is returned.

```
Debug.print Slide1.States("Default").HitTest(400, 300)
```

So, if we run this line of code in an experiment with a screen resolution of 800 x 600 pixels and an object called Image2 is presented in Slide1 at the centre of the screen, the String "Image2" would appear in the debugging window.

Sub-objects within a SlideState: Properties and methods

In order to access SlideText and SlideImage sub-objects in a particular SlideState, you can refer to them via the objects property of a given SlideState. However, before you can read or change properties of a sub-object, you need a temporary variable which references the sub-object.

```
Dim theSlideText As SlideText
Set theSlideText = CSlideText(Slide1.States("Default").Objects("Text1"))
```

So, in the first line we declare our temporary variable of the type SlideText. Then, in the second line we use **Set** to make a reference to the particular SlideText and use the CSlideText casting function to interpret the Text1 object as having the data type SlideText (if you want to learn more about casting functions, check the E-Basic help file).

Now, we can change, for instance, the BorderWidth of the just referenced SlideText (so in this example SlideText "Text1" in the "Default" state of "Slide1") by adding the following line of code:

```
theSlideText.BorderWidth = 3
```

Similarly, you can also cast a SlideImage, by simply adapting the variable declaration and casting part. Here is an example:

```
Dim theSlideImage As SlideImage
Set theSlideImage = CSlideImage(Slide1.States("Default").Objects("Image1"))
```

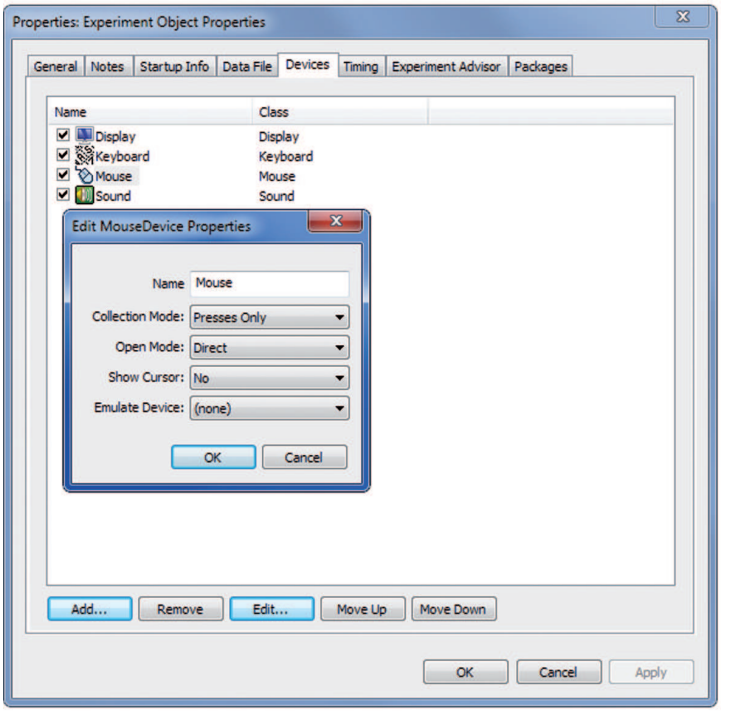
Then, you may change, for instance, the filename associated with the image (so in this example SlideImage "Image1" in the "Default" state of "Slide1"), using this line of code:

```
theSlideImage.FileName = "newfile.bmp"
```

As you can imagine, a similar Procedure is available for other sub-objects such as Sounds and Movies. For more information about all properties and methods available for Slide sub-objects, please refer to the E-Basic Help file.

Accessing the mouse in E-Prime®

The **mouse** is an interesting device as soon as you would like to add complicated hand movements or include questionnaires in E-Prime®. In the Properties of your Experiment, the mouse is activated by default, but its cursor is usually not shown. To change this, simply set the **Show Cursor** property to *Yes*, as indicated in the figure below:



However, showing a mouse cursor is often annoying when you run an experiment that depends on manual responses. So might it not be better to toggle the cursor on and off, depending on the particular time point in the experiment? Well, that is a good idea and pretty easy to do.

To show the mouse cursor on the screen, use the **ShowCursor** method, like this:

```
Mouse.ShowCursor True
```

And to hide it, use this code:

```
Mouse.ShowCursor False
```

Another useful property you may often need is to read the mouse coordinates in pixels. Simply use its CursorX and CursorY properties, as is shown in the example below:

```
Debug.Print Mouse.CursorX
Debug.Print Mouse.CursorY
```

Programming user interactions

There are a number of situations where we may feel the need to combine mouse input and Slide objects. These basically boil down to two major possibilities:

1. You would like to show feedback immediately after a button or mouse click. Unless an exit criterion is met, feedback should be continuously presented on the screen.

2. You would like to have *instant* (not only following a button or mouse click) and continuous online control over your Slide.

The sections below describe both basic setups and their principles. Use them as templates; they can easily be adapted to suit your own needs.

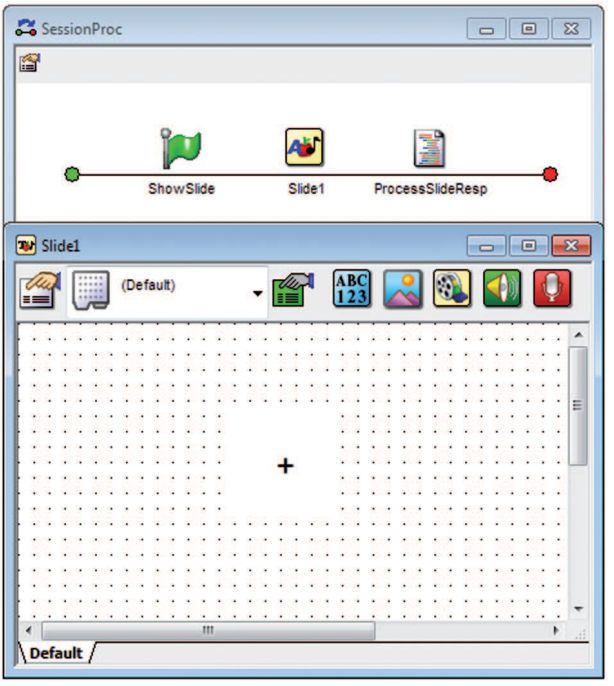
Continuous feedback after mouse clicks

Refreshing a Slide object after a mouse click is something you may want to do when programming questionnaires, visual analogue scales, or other types of mouse pointing-and-clicking tasks.

Imagine you would like to show some visual feedback immediately after a user makes a left-button mouse click in a Slide object, and this Procedure should be repeated again and again until the user clicks with the other (right) mouse button. As feedback, the SlideText position should move to the position of the mouse cursor after each mouse click.

To implement this, we need three objects: a Label, a Slide object, and an InLine object. In addition, add a SlideText sub-object to the default Slide state. See the examples below.

Make sure to set the Slide Duration to *Infinite* and Terminate after the user made a mouse click. Also set ShowCursor to Yes in the experimental properties.



This is the code we need in the ProcessSlideResp Inline object:

```
If Slide1.InputMasks.Responses.Count > 0 Then
   'Get the mouse response
   Dim theMouseResponseData As MouseResponseData
   Set theMouseResponseData = _
     CMouseResponseData(Slide1.InputMasks.Responses(1))
   If theMouseResponseData.RESP = "1" Then
      Dim theSlideText As SlideText
      Set theSlideText = _
        CSlideText(Slide1.States("Default").Objects("Text1"))
      theSlideText.X = theMouseResponseData.CursorX
      theSlideText.Y = theMouseResponseData.CursorY
   Else
      'Exit criterion, stop the experiment
      End
   End If
End If
GoTo ShowSlide
```

In this case, the Goto command at the bottom of the code creates a loop, which triggers a continuous refreshing of the content of the Slide object each time the user makes a mouse click.

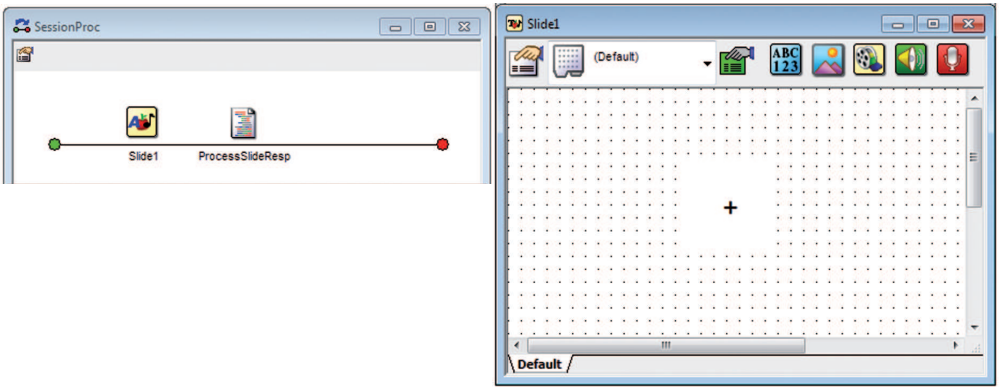
The script introduces a MouseResponseData object, which can be used to read the mouse position and responses. The MouseResponseData refers to the object InputMasks.Responses(1), which does the actual response logging of the Slide object. If you would like to learn more about the InputMask object, please study its complicated details in the E-Basic Help file.

Continous online control

Some situations require you to have online access to a device and instantly refresh the Slide object. For example, if you would like to program an approach / avoidance task where you can pull towards (enlarge) or push away (shrink) images with a joystick, you need to sample mouse or joystick data as fast as possible and instantly refresh the screen according to the new cursor position. Other examples are mouse tracking tasks and implicit learning tasks that read complicated hand movements using the mouse or other devices.

The code below shows the basic programming flow you need when you would like to update a SlideText's position instantly. In this case, the position is matched to the position of the mouse cursor instantly (i.e. whenever the mouse is moved, the display is updated immediately). This Procedure described below runs in a Slide object for 10 seconds.

To have continuous online control, we use the InLine object again. To show the display we use the Slide1 object, which is constantly redrawn by our script. See the example below:



Make sure you set the Slide Duration to 0! Note that there is still the possibility to log responses: simply set the Time Limit to the value you prefer (i.e. a value >0).

Also set ShowCursor to *Yes* in the experimental properties.

This is the content of the ProcessSlideResp InLine object you need:

```
Dim theSlideText As SlideText
Do While (Clock.Read - Slide1.OnsetTime) < 10000
   Set theSlideText = CSlideText(Slide1.States("Default").Objects("Text1"))
   theSlideText.X = Mouse.CursorX
   theSlideText.Y = Mouse.CursorY
   Slide1.Draw
   Display.WaitForVerticalBlank
   Sleep 10
Loop
```

The Do-While Loop structure here repeats drawing Slide1 again and again, until 10,000 ms (10 seconds) since the Slide1.OnsetTime has passed. However, given that it is useless having faster loops than the screen refresh rate, we decided to add a Display.WaitForVerticalBlank statement and a sleep command to create a loop that is simply as fast as (but not faster than) the refresh rate of the monitor.

In case you also want to terminate the loop whenever the user makes a button press, simply changing the device's End Action to Terminate will not work. Instead, you have to add an additional criterion to the loop checking whether StimSlide.InputMasks.IsPending() is *True*. As soon as IsPending() gets the value *False*, you know that a response was made or the Time Limit was exceeded.

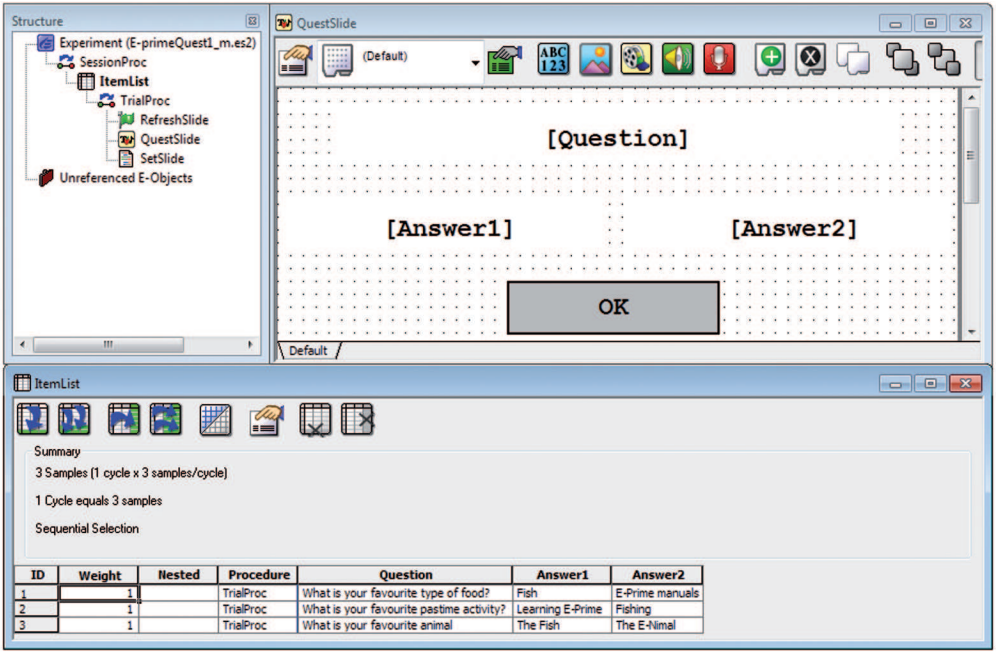
Tutorial X: A simple questionnaire

Let's program a simple Questionnaire in E-Prime®! The ItemList determines which questions and answers should be presented. The participant is allowed to make multiple selections. Selected items need to be indicated by a black border.

Note that we basically can adapt the 'continuous feedback after mouse clicks' design proposed in this chapter.

Step 1: The basic design

Program the design as depicted in the figures below.



Make sure you set the Slide Duration to *infinite* and terminate after the user makes a mouse click. Also set ShowCursor to *Yes* in the experimental properties.

Name the relevant SlideText objects 'Option1', 'Option2' and 'OKbtn'.

Step 2: Declare variables

Add the following lines of codes to the SetSlide Inline object:

```
Dim Opt1 As SlideText
Dim Opt2 As SlideText
Set Opt1 = CSlideText(QuestSlide.States("Default").Objects("Option1"))
Set Opt2 = CSlideText(QuestSlide.States("Default").Objects("Option2"))
```

In these lines, we create SlideText variables that refer to Option1 and Option2 in the QuestSlide. These variables allow us to later set their property *BorderWidth* to 1 or 0 (indicating whether the particular item is selected).

Step 3: Check the mouse response and do the hit test

The next step is to load the mouse-data response that triggered the termination of the QuestSlide. We first make sure that a response is really made (Responses.Count > 0). In addition, we use the *HitTest* method to know whether the user clicked any object at the specified mouse cursor coordinates. If so, the variable strHit will contain the name of this object.

```
If QuestSlide.InputMasks.Responses.Count > 0 Then
   Dim theMouseResponseData As MouseResponseData
   Set theMouseResponseData = _
     CMouseResponseData(QuestSlide.InputMasks.Responses(1))
   Dim strHit As String
   strHit = QuestSlide.States("Default").HitTest _
     (theMouseResponseData.CursorX, theMouseResponseData.CursorY)
   'process strHit
   'TODO in Step 4
End If
```

Please add these lines of codes to the Inline object.

Step 4: Process the hit test data

Now, consider how to process the hit test data. If the user clicked on Item1, we would like to show some selection/deselection feedback, by adjusting the BorderWidth of the respective SlideText object.

To toggle between selection and deselection add these lines of code and remove the comment related to Step 4:

```
   If strHit = "Option1" Then
     If Opt1.BorderWidth = 0 Then
       Opt1.BorderWidth = 1
     Else
       Opt1.BorderWidth = 0
     End If
   End If
```

Repeat these lines of code for Item2:

```
   If strHit = "Option2" Then
     If Opt2.BorderWidth = 0 Then
       Opt2.BorderWidth = 1
     Else
       Opt2.BorderWidth = 0
     End If
   End If
```

Step 5: Decide when to refresh the Slide

In the final step we have to consider our exit criterion. When do we want to refresh the Slide object and when do we want to exit the script? Well, the Slide object should always be refreshed unless the hit test reveals that the user clicked the OK-btn object. So, we would like to jump back to the RefreshSlide if strHit <> "OKbtn":

```
If strHit <> "OKbtn" Then
   GoTo RefreshSlide
End If
```

What if the user clicked *OK*? Well, then the program proceeds to the end of the TrialProc and will finish. But, wait a moment! In that case, we may first want to store the selections in the *edat2 file*. To do so, *replace* the If-Then statement mentioned above and add these lines of code to the end of our InLine script:

```
If strHit <> "OKbtn" Then
   GoTo RefreshSlide
Else
   C.SetAttrib "Option1Selected", CStr(Opt1.BorderWidth)
   C.SetAttrib "Option2Selected", CStr(Opt2.BorderWidth)
End If
```

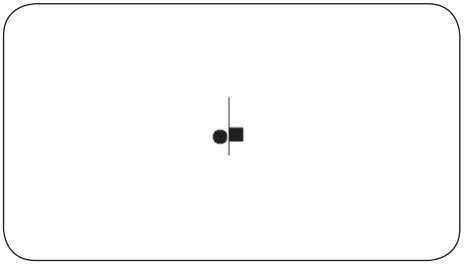
That's it! Test whether your first questionnaire in E-Prime® works appropriately.

Tutorial XI: A mouse tracking task

As a young student you are likely to have efficient motor control over your mouse, but what happens when you become older? Well, it is likely that your motor performance will become impaired. In this tutorial we are going to program a mouse tracking task that may be useful for calculating indices of motor (dis)ability.

This mouse tracking task is an adapted version of the one-dimensional task described by Riviere & Thakor (1996). As shown in the figure on the next page,

a stationary vertical line segment (100 pixels) is displayed on the computer screen (1024 x 768 pixels). A small square target oscillates along the right side of the line segment in sinusoidal fashion. The participant tracks the target's motion with a small round mouse cursor, which moves along the left side of the line. The experiment should be programmed such that the X component of the mouse signal is ignored and only the Y component is sampled. In this way, the mouse cursor is constrained to move vertically and is not sensitive to sideways mouse movement. The trial should stop after a mouse click, or in case of omission after 10 seconds.



Note that we basically need to adapt the 'continuous online control' design proposed in this chapter.

Step 1: The basic design

Create an E-Prime® experiment including a TrialList and a TrialProc (no attributes defined). Add a Slide object and name it 'TrackingSlide'. Add an InLine object and name it 'ProcStim'.

Set the duration of the TrackingSlide to 0. Add a Mouse Device and set the *TimeLimit* to 10,000 ms.

Add two SlideImages and name them 'Dot' and 'Square', respectively (see the figure above). Create a Dot.bmp and a Square.bmp file in MS Paint (20 x 20 pixels) and load them into the SlideImages.

The line segment can be created using a TextDisplay having a Width of 1 and a Borderwidth of 1.

Make sure the line is presented in the centre of the screen. Align the *X positions* of the Dot and the Square images so that they are presented along the left and right side of the line, respectively.

Step 2: Set the online Slide control

In the ProcStim Inline script, add the following lines of code.

```
Dim CurrTarget As SlideImage
Dim CurrCursor As SlideImage
Set CurrTarget = _
  CSlideImage(TrackingSlide.States("Default").Objects("Square"))
Set CurrCursor = _
  CSlideImage(TrackingSlide.States("Default").Objects("Dot"))
```

These references allow us to later change the *Y position* of the respective SlideImages.

The only part of the code that should be added is shown below:

```
Dim ClockNow As Long
Dim PeriodDur As Single
Dim y As Single
Dim Amplitude As Integer
Amplitude = 20
PeriodDur = 2000 'ms, duration of one period
Do While (TrackingSlide.InputMasks.IsPending())
   ClockNow = Clock.Read
   y = Sin((ClockNow - TrackingSlide.OnsetTime)/PeriodDur*2*pi)
   CurrTarget.Y = (Display.YRes/2) - (Amplitude * y)
   'move cursor
   CurrCursor.Y = Mouse.CursorY
   Display.WaitForVerticalBlank
   TrackingSlide.Draw
   Sleep 10
Loop
```

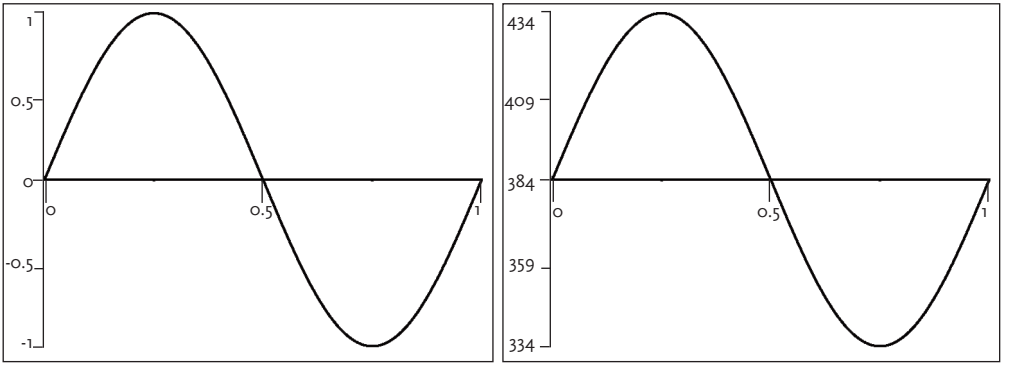
Let's study these lines of code thoroughly!

The most important structure is the **Do While – Loop** structure, which repeatedly calls the TrackingSlide.Draw method while the mouse response is pending (i.e. no mouse click given and time limit not yet exceeded).

The other important part relates to the lines starting with 'CurrTarget.Y =', and 'CurrCursor.Y ='. Here we change the position of the Square and Dot images in the TrackingSlide object.

Because the target should move in a sinusoidal fashion, we first calculate *y*, which uses the **Sin()** function in combination with the time passed (= ClockNow - TrackingSlide.OnsetTime) and the duration of the period (here 2000 ms) in radials (i.e. * 2 * pi) to produce a value between -1 and +1.

The left panel of the figure below shows a standard sinusoidal function for 1 cycle.



However, given that we would like to calculate the value in pixels measured from the top of the screen, we use the formula (Display.YRes/2) - (Amplitude * y). So, when using a 1024 x 768 resolution we used (768/2) - (50 * y) to produce the sinusoidal movement around the vertical midline on the screen (as shown in the right panel of the figure).

Run the experiment and check whether it works correctly!

Note: the variable Duration in the example above has a fixed value throughout the experiment. In such cases you might consider to declare a constant instead, using the **Const** statement. See the E-Basic help file for more information about the **Const** statement.

Exercises

- Add extra trials to the List of the questionnaire in Tutorial X. Add an InLine script to the beginning of the TrialProc to make sure that each question always starts showing all options deselected.

- Disable the possibility of selecting option1 and option2 at the same time, so that the participant is forced to choose one out of two. Draw a flowchart before programming.

- Now adapt the questionnaire in such a way that the participant has to choose one out of eight options. Draw a flowchart before programming.

- The questionnaire you created can be useful as a manipulation check. Add the questionnaire to the end of the Ego depletion experiment (Tutorial V, Chapter III). Ask participants to rate on an 8-points scale how tired, thoughtful, excited, happy, worn out, sad/depressed, angry, and calm they feel at the end of the film viewing task. Make sure the response is stored in the edat file.

- Download the file *E-primeQuestionnaire.es2* from www.e-primer.com.This program shows how you can present one or more sheets of ten questionnaire items on a screen in E-Prime®. Perfect for long questionnaires! The answers given are automatically stored in the edat file (see the attributes *ChosenOption* and *ChosenOptDes*). To ensure a convenient structure in the edat file, the questionnaire was programmed in a roundabout way, needing many Lists, Procedures and InLine (too complicated to explain here). However, the only thing you need to know for now is: Whenever the Procedure *RunOneSheetOf10Items* is executed, it presents the content of 11 consecutive rows (one header + ten items with their options) stored in the *AllQuestionnaires* List.

- Adapt the content of the questionnaire to suit your own needs (e.g. copy-paste a bunch of your favourite personality questionnaires). In order to change the text for the items, just change the content of the *AllQuestionnaires* List (note that row numbers 1-11, 12-22, 23-33, etc. will be presented in separate sheets). ItemIDs fields that are left empty will not be used, neither are option fields that are left empty. Optionally, you can set the PreSelection field, when a particular item number should be preselected in advance.

- Call the Procedure *RunOneSheetOf10Items* at the proper location (in a particular List) in your experiment and repeat this for the number of sheets you would like to present.

- Experiment with changing the content of the *AllQuestionnaires* List and run the questionnaires!

Advanced exercises

- Add additional frequency conditions (1 Hz, 2Hz and 4Hz) to the TrialList of the experiment in Tutorial XI and adjust the code so that the value in PeriodDur depends on the given attribute in the List.

- Store the CurrTarget.Y and CurrCursor.Y values in two Arrays. The size should equal the maximum number of cycles through the loop. Each cycle is stored in a separate element of the Array.

- Super advanced level (!): Give feedback about performance at the end of each trial using the Accuracy Index (AI). Riviere & Thakor (1996, p. 9) describe how to calculate this index: (Hint: read 'Arrays' when they refer to 'vectors'; RMS = the root of the mean of the squares of all elements in the Array / vector):

  > The position vector of the target on the computer screen was the input to the human sensorimotor system in these tests, referred to as the "target" and represented as t. The motion that the human subject made in response to the target was indicated by the mouse cursor location. This was considered the output, o, of the system. The error vector e was obtained by the equation.
  >
  > $e = t - o.$
  >
  > (...)For the 1-D tests, these three are only one-dimensional vectors :
  >
  > $e = [e_1], t = [t_1], o = [o_1].$
  >
  > For each test, E and A, the RMS values of the error vector e and the target signal t, respectively, were calculated for each test. The subjects' overall accuracy in tracking was represented by a measure called the "Accuracy Index (AI)". (...)
  >
  > In the special case of the 1-D tests, there was no x-component, and the AI could be simplified as
  >
  > $AI = 1 - E / A$
  >
  > For all tests, perfectly accurate motion generation resulted in zero error and, therefore, in an AI of unity. Leaving the cursor unmoved in the centre of the screen resulted in a value of zero for AI.

- Check whether your feedback script works correctly!

Reference List

Baumeister, R.F., Bratslavsky, E., Muraven, M. & Tice, D.M. (1998). Ego depletion: Is the active self a limited resource? *Journal of Personality and Social Psychology, 74*, 1252.

Blais, C. (2008). Random without replacement is not random: caveat emptor. *Behavior Research Methods, 40*, 961-968.

Duncan, J. & Humphreys, G.W. (1989). Visual search and stimulus similarity. *Psychological Review, 96*, 433.

Greenwald, A.G., McGhee, D.E. & Schwartz, J.L. (1998). Measuring individual differences in implicit cognition: The implicit association test. *Journal of Personality and Social Psychology, 74*, 1464.

Güth, W., Schmittberger, R. & Schwarze, B. (1982). An experimental analysis of ultimatum bargaining. *Journal of Economic Behavior & Organization, 3*, 367-388.

Luck, S.J. & Vogel, E.K. (1997). The capacity of visual working memory for features and conjunctions. *Nature, 390*, 279-280.

Riviere, C.N. & Thakor, N.V. (1996). Effects of age and disability on tracking tasks with a computer mouse: accuracy and linearity. *Journal of Rehabilitation Research and Development, 33*, 6-15.

Schneider, W., Eschman, A., & Zuccolotto, A. (2002). E-Prime User's Guide. *Psychology Software Tools*: Pittsburgh, PA.

Simon, J.R. & Rudell, A.P. (1967). Auditory SR compatibility: The effect of an irrelevant cue on information processing. *Journal of Applied Psychology, 51*, 300.

Sternberg, S. (1969). Memory-scanning: Mental processes revealed by reaction-time experiments. *American Scientist*, 421-457.

Stroop, J.R. (1935). Studies of interference in serial verbal reactions. *Journal of Experimental Psychology*, *18*, 643.

Williams, K.D. & Jarvis, B. (2006). Cyberball: A program for use in research on interpersonal ostracism and acceptance. *Behavior Research Methods*, *38*, 174-180.

Appendix: Overview of available E-Objects

Here we provide an overview of all E-Objects, listing their main function and the chapter in which the object is introduced. For a complete overview of all properties of each E-Object, we refer you to the E-Prime 2.0 documentation, which can be downloaded from the E-Prime® support page.

Icon	Name	Description	Chapter
	Procedure	Used to determine the order of events in an experiment.	I
	List	Contains rows of items with specific properties (attributes). Lists usually call Procedures.	I,II,III
	TextDisplay	Displays one or more lines of text.	I
	ImageDisplay	Displays pictures.	II
	Slide	Presents a combination of text, images, movies and sound.	II
	FeedbackDisplay	Provides specific feedback based on the participant's response to objects presented earlier in the experiment flow.	II
	MovieDisplay	Displays a movie clip.	III
	SoundOut	Presents a sound file (.wav/.mp3/.wma).	III
	SoundIn	Records sounds.	Not in this book
	Wait	Waits for a specified time without changing the visual output.	III

Icon	Name	Description	Chapter
	InLine	Used to add E-Basic script at a specific location of the experiment flow.	IV
	Label	Indicates a particular location on the timeline (Procedure). The program can 'jump' backward or forward to a Label, in order to repeat or skip a part of the Procedure.	III
	PackageCall	Contains reusable blocks of E-Basic script written by users of E-Prime 2 (often used in Procedures which are used repeatedly or for instance in connecting equipment such as an eye-tracker to an E-Prime® experiment). As packages are beyond the scope of this book, please see the 'E-Prime 2' documentation for a more detailed description.	Not in this book

About the authors

Michiel M. Spapé is a cognitive psychologist at the Helsinki Institute for Information Technology in Finland. Since obtaining his PhD at Leiden University in 2009, he has focused on interrelating the classical functions of the human mind: showing the perception of action, the memory of control, the emotions of problem-solving and the social aspect of individual cognition. Methodologically, he likes to tinker with machines and biology, working with big computers, trackpads, eye trackers, tactors, Kinects, EEG, EMG, MEG and so on. His favourite type of study reheats a venerable psychological effect, spicing it with new tech and adding neuro sauce, before serving it to the academic table.

Rinus G. Verdonschot is trained as a psycholinguist specialized in applied linguistics and cognitive neuroscience. His main work is in in the fields of psycho- and neurolinguistics focusing on language production, reading and bilingualism. He has considerable practical experience working at diverse experimental labs (including EEG/fMRI) in different countries. He enjoys testing theory-driven hypotheses, writing scientific articles about them and presenting the results at international conferences. He also enjoys teaching students at undergraduate and graduate levels and to collaborate with and learn from other researchers.

Saskia van Dantzig was trained as a cognitive psychologist and obtained her PhD on the topic of embodied cognition. She currently works as a senior scientist at Philips Research (Eindhoven, the Netherlands), developing products and services that support people to live a healthy life. She loves to work at the crossroads of psychology and technology, and believes that this combination leads to useful and much needed innovations.

Henk van Steenbergen was trained as an experimental psychologist at Leiden University in the Netherlands where he currently is assistent professor. He likes to combine behavioural, physiological and neuroscientific methods to study human behaviour, in particular the role of emotion and motivation in driving goal-

directed action. He was originally educated as an electronic engineer, and so likes to apply cool hardware and software in psychological research. Having been a programmer since his teens, he is convinced that good programming skills enable more efficient and more enjoyable research, a message he tries to convey when teaching E-Prime® programming to graduate students.

Index

Abs 98
ACC 35
ActiveState 56, 62, 124, 134
Addition (+) 36, 96
Address book 149
And 82, 117, 131, 165
Array 137, 143, 146, 154, 155, 177
Arrays of a user-defined data type 149
Assignment 94, 126
Attribute 25, 39, 42, 45, 47, 50, 52, 61, 62, 63, 64, 82, 83, 88, 103, 125, 134, 144, 145, 176, 199

Basic 75
Beep 105
Between-subject manipulations 80
Bit-depth 67, 73
Bitmap 53
Bits and bytes 186
Block 19, 23, 36, 38, 40, 63, 64, 159, 200
Boolean 93, 117
BorderColor 29
BorderWidth 29
Browser window 23
Buffer Mode 70
Buffer Size 70
c.GetAttrib 104
CInt 99
Clock.Read 106
Clock.Scale 22, 107
Code substitution 99, 115
Collection Mode 181
Combine conditional expressions 117
Combining Arrays and Lists 148
Comment 95, 157
Comment and uncomment blocks of code 95
Comparing values 118
Comparison operators 118
  Equal to (=) 40, 59, 118
  Greater than (>) 118
  Greater than or equal to (>=) 118
  Less than (<) 30, 118
  Less than or equal to (<=) 118
  Not equal to (<>) 118
Conceptualising an experiment 19
Conditional expression 114, 115, 117, 141, 143
Continous online control 168
Continuous feedback after mouse clicks 166
control+alt+shift 22, 27, 39
Conversion functions
  CInt 99
  Convert a String to a Single, 99
  Convert a String to an Integer 99
  Convert numerical variables to a String 99
  CSng 99
  CStr 99

Convert a String to a Single, 99
Convert a String to an Integer 99
Convert numerical variables to a String 99
Counterbalancing 39, 48, 80
CRESP 35
c.SetAttrib 144
CSng 99
CStr 99
Cyberball game 128
Cycle 48, 177

Debug.Print 94
Decision making 134
ElseIf 131
If-Then-Else-End If statement 113, 114
If-Then statement 113, 114
Nested If-Then structures 116
Select...Case 131
Declaring and using Arrays 146
Devices 27, 32, 68, 69, 76, 77, 180
Digital audio 71
Dim 93
Display hardware 77
Display.Refreshrate 104
Distracting sound 81
Division (/) 96
Do – Loop Until 142
Do – Loop While 143
Double 37, 38, 39, 93
Do Until – Loop 142
Do While – Loop 143, 174
Duration 30, 33, 39, 52, 75, 85, 132, 166, 169, 182
DurationError 35
Duration tab 30

E-Basic 11, 77, 92, 115, 137, 200
Echo 109
E-DataAid 11, 15, 16, 40, 43, 88
Ego depletion experiment 87, 88, 176
ElseIf 131
E-Merge 11, 15, 16, 43, 88
End 31, 33, 71, 74, 75, 87, 114, 120, 121, 132, 135, 159, 169
End Movie Action 74, 132
End Sound Action 71
E-Object
FeedbackDisplay 24, 31, 56, 58, 62, 65, 134, 199
ImageDisplay 24, 53, 199
InLine 24, 31, 92, 124, 134, 175, 176, 195, 200
Label 24, 75, 200
List 24, 37, 38, 52, 59, 61, 62, 64, 83, 86, 88, 104, 120, 134, 144, 175, 176, 199
MovieDisplay 24, 73, 87, 199
PackageCall 24, 31, 200
Procedure 23, 24, 25, 27, 37, 38, 41, 42, 59, 60, 65, 76, 83, 94, 120, 133, 159, 176, 199, 200
Slide 24, 56, 61, 62, 64, 134, 163, 169, 172, 174, 199
SoundIn 24, 185, 199
SoundOut 24, 68, 199
TextDisplay 24, 27, 28, 30, 31, 32, 33, 88, 134, 199
Wait 74, 199
EOF 191
Equal to (=) 40, 59, 118
E-Recovery 11, 15, 17, 27
E-Run 17, 22, 108
E-Studio 11, 15, 16, 22, 37
Event 20, 31
Experimental design 46
Experiment properties 76
Exponentiation ^ 96

Feedback 56
FeedbackDisplay 24, 31, 56, 58, 62, 65, 134, 199
Field 149, 176
FileExists 190, 191
Filename 53, 70, 73
File types
.ebs2 file 17
.edat2 file 16, 27, 43, 79, 172, 176
.emrg2 file 16, 43
.es2 file 16, 176
.txt file 17
FirstFrameTime 74
Font tab 30
Format 99
For-Next loop 139, 159
FrameRate 74
FramesDisplayed 74
FramesDropped 74
Frame tab 29
Full tab 92
Functions 98

General tab 28
Global level 97
Global variables 92
Goto 75
Greater than (>) 118
Greater than or equal to (>=) 118

If-Then-Else-End If statement 113, 114
If-Then statement 113, 114
ImageDisplay 24, 53, 199
Implicit Association Task 62
Inheritance 19, 39
InLine 24, 31, 91, 92, 124, 134, 175, 176, 195, 200
InputBox 102, 103
Instance 19, 27, 36, 59, 83, 159, 200
InStr 100, 109
Integer 93, 99
Integer division (\) 96

Jitter 67, 79, 80, 107
Jump 131
Jump Label 75

Label 24, 75, 131, 135, 200
Latent 46, 50, 59, 82
LCase 100
Left 29, 101
Len 100
Less than (<) 30, 118
Less than or equal to (<=) 118
Line Input 191
List 24, 37, 38, 52, 59, 61, 62, 64, 83, 86, 88, 104, 120, 134, 144, 154, 155, 175, 176, 199
List methods
c.GetAttrib 104
c.SetAttrib 144
<ListName>.GetAttrib 145, 154
<ListName>.SetAttrib 155
<ListName>.Terminate 120, 134
Log 32, 33, 34, 39, 79, 169
Logging tab 32, 34
Logging timing
Logical operator
And 82, 117, 131, 165
Not 117, 199, 200, 135
Or 20, 50, 117
Truth tables 117
Xor 117
Long 93, 98
Loop 71, 139, 142, 143
Looping with Labels 137
Loops 138, 158, 159, 169, 177
Do – Loop Until 142
Do – Loop While 143
Do Until – Loop 142

Do While – Loop 143, 174
For-Next loop 139, 159
Loops with conditional expressions 141
LPT port 185, 187, 188, 194

Manifest 46, 59, 82
Manipulate the content of a String 100
Match Desktop Resolution At Runtime 67
Mathematical functions
Abs 98
Random 98
Sin 98, 174
Math manipulations 96
Math operators 96
Addition (+) 36, 96
Division (/) 96
Exponentiation ^ 96
Integer division (\) 96
Modulus division (mod) 96
Multiplication (*) 96
Negation (-) 96
Subtraction (-) 96
Menu 22
Method 17, 27, 163, 164
Mid 101
Mirror 54
Modulus division (mod) 96
Mouse 32, 41, 77, 161, 165, 166, 170, 171, 172, 177
Mouse tracking task 172
MovieDisplay 24, 73, 87, 199
MsgBox 102, 103
Multiplication (*) 96

Negation (-) 96
Nested If-Then structures 116
Nested Lists 79
Not 117, 135, 199, 200
Not equal to (<>) 118
Object 15, 17, 23, 27, 28, 31, 57, 58, 62, 73, 74, 83, 92, 124, 134, 163
Object Oriented Programming 17
Offset 33, 49
OffsetTime 35
OnsetDelay 35
Onset Sync 33, 78, 79
OnsetTime 35
Open 37, 60, 134, 190, 191
Operator 40, 95, 98, 126, 150, 162
Or 20, 50, 117
Output window 23, 91

PackageCall 24, 31, 200
Pan Control 71
Parallel port 185, 187, 190, 194
Permutation 49
Port 181
Position Time Format 70
Print 192
Procedure 23, 24, 25, 27, 37, 38, 41, 42, 59, 60, 65, 76, 83, 94, 120, 133, 159, 176, 199, 200
Properties 17, 23, 26, 27, 28, 30, 32, 33, 52, 53, 54, 61, 62, 77, 87, 124, 163, 164, 199
ACC 35
ActiveState 56, 62, 124, 134
Buffer Mode 70
Buffer Size 70
Collection Mode 181
CRESP 35
Duration 30, 33, 39, 52, 75, 85, 132, 169, 182
DurationError 35
End Movie Action 74, 132
End Sound Action 71
Filename 53, 70, 73
FirstFrameTime 74
FrameRate 74

FramesDisplayed 74
FramesDropped 74
Jump Label 75
Mirror 54
OffsetTime 35
OnsetDelay 35
OnsetTime 35
Pan Control 71
Port 181
Position Time Format 70
RESP 35, 124
RT 35, 40, 134
RTTime 35
ShowCursor 165
Start Offset 70
Start Position 73
Stop After 71, 74
Stop Offset 70, 71
Stop Position 73
Stretch 55, 74
Volume 71
Weight 26
Properties window 23, 26

Quasi-random trial selection 150, 151, 159
Questionnaire 175, 176

Random 98
RandomizeArray 156
Reading a text file 190
Reset / Exit tab 49
RESP 35, 124
Right 29, 101
RT 35, 36, 40, 134
RTTime 35

Sample rate 67, 73
Scope 97, 190, 200
Screen tearing 34
Script 174
Script window 92
Select...Case 131
Selection tab 48
Sending signals using the parallel port 185
Serial-Response box 180
SessionProc 23, 37, 39, 87, 88, 131
Set 29, 39, 64, 87, 134, 164, 166, 174
ShowCursor 165, 166
Showing an image 53
Showing multiple images and layers of texts 56
Simon Effect 58
Sin 98, 174
Single 93
Sleep 105, 188
Slide 24, 56, 61, 62, 64, 134, 161, 162, 163, 166, 169, 172, 174, 199
SlideState 163
SoundDevice 68, 77
SoundIn 24, 185, 199
SoundOut 24, 68, 199
SRBOX 180, 182
Start Offset 70
Start Position 73
Startup Info 76, 81
Stop After 71, 74
Stop Offset 70, 71
Stop Position 73
Stretch 55, 74
Stretching 54, 55, 195
String 83, 91, 93, 99
String functions
Format 99
InStr 100, 109
LCase 100
Left 101
Len 100
Manipulate the content of a String 100

Mid 101
Right 101
Trim 100
UCase 100
Stroop task 77, 88, 134
Structure window 23
Subject 80, 103
Sub-objects within a SlideState 164
Subtraction (-) 96
Sync tab 33
Syntax 93, 115

Task Events 190
Terminate 22, 27, 31, 33, 71, 119, 120, 121, 132, 134, 169
Text 93
TextDisplay 24, 27, 28, 29, 30, 31, 32, 33, 88, 134, 199
Toolbox 23, 37, 94
Transparency 55
Trials 19, 20, 23, 36, 40, 50, 62, 82, 134, 159, 175
Trim 100
Truth tables 117
Types 30, 31, 39, 50, 60, 91, 93, 150
Type statement 149

UCase 100
Ultimatum game 121
User-defined 149
User interactions 166
User Script 92, 97, 130, 149
User tab 92

Variable 41, 42, 59, 82, 93, 99, 104, 125
Variable types
Array 137, 143, 146, 177
Boolean 93, 117
Double 37, 38, 39, 93
Integer 93
Long 93, 98
Single 93
String 83, 93
Variant 93
Variant 93
Visual search 81
Voice key 182
Voice-key experiments 182
Volume 71

Wait 74, 199
Wave files 199
Weight 26
Width 29
Workspace 23
Writing to a text file 191

X 29, 169
XAlign 29
Xor 117

Y 29
YAlign 29