This preview only contains Chapters 1 and 7.

For information about where to buy this book, please visit:

www.e-primer.com
The E-Primer

An introduction to creating psychological experiments in E-Prime

Michiel Spapé
Rinus Verdonschot
Henk van Steenbergen

Leiden University Press
# Table of Contents

Preface to the Second Edition 9  
Preface to the First Edition 11

**Introduction** 13  
What is E-Prime and what will I learn? 13  
Why should I learn E-Prime? 14  
Online support 14

**Chapter I: E-Prime at a glance** 17  
E-Studio, E-DataAid, E-Recovery and E-Merge 17  
Object Oriented Programming 19  
Conceptualising an experiment 21  
The structure of E-Studio 24  
E-Objects 26  
Procedures, Lists and TextDisplays 27  
The TextDisplay Object 28  
Tutorial I: A simple RT experiment 38  
Exercises 44  
Additional Exercises 47

**Chapter II: List Attributes and Slides** 49  
Experimental design in E-Prime 50  
More on Lists 51  
The ImageDisplay object: Showing an image 58  
The Slide Object: Showing multiple images and layers of texts 62  
The FeedbackDisplay Object 63  
Tutorial II: The Simon Task 65
InLines everywhere 126
Tutorial VIII: A working memory test 129
Exercises 132
Additional Exercises 133

Chapter V: Decision making in E-Basic 135
The If-Then statement 135
Combining conditional expressions 138
Comparing values 141
How to terminate parts of your experiment 141
Tutorial IX: The Ultimatum game 144
Tutorial X: The Cyberball game 151
Exercises 157
Additional Exercises 158

Chapter VI: Loops and Arrays in E-Basic 161
Looping with Labels 161
The For-Next Loop 163
Loops with conditional expressions 165
Lists and Arrays 167
Arrays of a user-defined data type 173
Tutorial XI: Quasi-random trial selection 175
Exercises 183
Additional Exercises 184

Chapter VII: Interactions between Slide objects and the Mouse 185
The Slide object 186
Accessing the mouse in E-Prime 189
Programming user interactions 190
Tutorial XII: A simple questionnaire 193
Tutorial XIII: A mouse tracking task 196
Tutorial XIV: Embodied cognition, political leaning and Stroop featuring buttons 199
Exercises 204
Additional Exercises 205
Chapter VIII: Various Input/Output devices 207
The Serial Response Box 208
The Chronos 210
Voice key experiments 211
Sending signals using the parallel port 215
How to send signals 218
Reading and writing text files 225
Tutorial XV: Making a Voice Key (VK) test program 227
Exercises 229
Additional Exercises 230

References 231
Appendix: Overview of available E-Objects 233
About the authors 235
Index 237
Preface to the Second Edition

It has been five years since the first edition of The E-Primer and we remain proud of our work. Soon after, however, Psychology Software Tools updated their E-Prime to version 3 with tons of new features. As it quickly became apparent that the demand for The E-Primer far outstripped our expectation, we decided The E-Primer could do with a fresh and shiny coat of paint as well. We are immensely pleased to announce this second edition achieves that, and much more. The most important new features of E-Prime 3 are now all covered, complete with new exercises and tutorials. We also made many adjustments to the original text in response to suggestions from students and colleagues. Hopefully, they will agree this new edition is much improved, without the humorous style being sacrificed: We remain convinced that programming experiments in E-Prime is great fun and should never become a chore.

We would like to thank Saskia van Dantzig for her contributions to the first edition of the book. This second edition has benefited much from expert feedback from David McFarlane and the people at Psychology Software Tools (David Nicholson, Anthony P. Zuccolotto, and Gretchen Brauch): thank you all for all your time and effort! Finally, we would each like to mention a few people personally:

Michiel Spapé: “I would like to express my appreciation for colleagues who supported and inspired my teaching: Thanks, Manuel Eugster (Aalto University) and Liverpool Hope University gang: Letizia Palumbo, Belen Lopez-Perez, Yue Yue, & Antonio Zuffianò. I wish I could imitate you better!”

Rinus Verdonschot: “I would like to thank my colleagues and friends for providing me with inspiration and my participants for taking part in my (occasionally mind-numbing) experiments.”

Henk van Steenbergen: “I would like to thank the technicians at the SOLO department at Leiden University for their help and suggestions for improvements and the students and colleagues in my yearly E-Prime classes for their feedback.”
Preface to the First Edition

This book is the result of intensive collaboration between people who, at one time or another, all studied or worked at Leiden University in the Netherlands. In 2006, we expanded 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 – revised the manual in order to help those who want to learn how to create their own psychological experiments. Later, we expanded the manual even further, so that 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: “Thanks, Elkan Akyürek, for introducing me to E-Prime, and Jan-Rouke Kuipers for sharing the burden when we took over and redesigned the course. I am also grateful to many inspiring teachers and colleagues at Leiden University (Wido La Heij, Gezinus Wolters, Stephan Verschoor). Finally, thank you, Zania Sovijärvi-Spapé, for being my favourite test-subject and person.”

Rinus Verdonschot: “I am grateful to the Cognitive Psychology department of Leiden University for having given me the chance to teach E-Prime to so many enthusiastic students; 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 thank you 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. That 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 to Margot Schel for helping me to thoroughly revise and extend a precursor 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 their support.”
Introduction

The E-Primer is written with a reader in mind who is eager to learn, but knows lit-
tle, if anything, about programming, computer science and the actual implement-
tation of all those wonderful scientific experiments that populate the reading lists
of psychologists and cognitive scientists. This is not to say, however, that more
experienced readers will not find it instructive, 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 developed and distributed by Psychology Software
Tools, Inc. It is specially designed to facilitate development of custom psychologi-
cal experiments, and to acquire and analyse data. E-Prime consists of several pro-
grams 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 are using E-Prime 2 or 3,
but most of the features we discuss were already introduced in E-Prime 1. When
versions differ significantly in operation, we discuss each version separately.

You will learn how to use each of these programs effectively, ultimately in order
to implement your very own experiments. First, however, you will learn how to re-
create a number of fascinating, famous experiments. We will guide you through
this process in an easy-to-follow, step-by-step approach – ‘now click on this button
over here’ – using Tutorials. Along the way, we’ll explain the reasoning behind
each step and give you tips on good practice in experimental design. Gradually,
we will move beyond the narrow confines of ‘click-here-now-click-there’ and ask
you to implement simple variations on the experiments you’ve met so far. These
form the Exercises at the end of each chapter. Finally, in the Additional Exercises,
you will be invited to explore, with minimal guidance, the horizons of E-Prime
and how you could use what you’ve learned to implement your own research
questions. Always 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 will become involved in a research project during their studies. 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 essentially involves 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 a research project), but it also helps you to read and understand empirical papers. Finally, we hope we can communicate some of our own enthusiasm: in the end, programming and realising your ideas – creating something from scratch – is really a lot of fun.

**Online support**

E-Prime has a useful support website: [https://support.pstnet.com/hc/en-us](https://support.pstnet.com/hc/en-us). Here, you will find examples of experiments, answers to frequently asked questions and solutions for problems. If you encounter a problem and can’t find the answer on their FAQ and Knowledge Base pages, you can simply send them your question via a special form on the website. You’ll then receive a personal answer, usually within a day or two. All you have to do in order to get this personal support is to register (for free) on the website.

More information can 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 webbased project designed to maximise the use of E-Prime: [https://step.talkbank.org/](https://step.talkbank.org/) This website includes a multitude of examples of common paradigms in experimental psychology. Some of these experiments are programmed in an earlier version of E-Prime (E-Prime 1). However, you can still open and run these experiments in more recent versions of E-Prime. PST (Psychology Software Tools, the developer of E-Prime) has also set up their own Experiment Library, see [http://www.pstnet.com](http://www.pstnet.com).
Lastly, you could think about joining the E-Prime mailing list. It’s independently run and has a sizeable community who may be able to help you out with urgent queries when you haven’t got time to wait for a response from E-Prime support. Or perhaps you have a question that’s more about experimental design. Or maybe you just want to let everyone know how much you love designing experiments in E-Prime! Visit \textit{http://groups.google.com/group/e-prime}.
Chapter I

E-Prime at a glance

In this chapter, you will learn

About:
• E-Studio, E-DataAid, E-Recovery and E-Merge
• Object Oriented Programming
• The structure of E-Studio
• Procedures
• Lists
• TextDisplays

How to:
• Create your first reaction-time experiment
• Polish 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 the different elements that make up an experiment. 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. Therefore, you will first learn how to conceptualise your experiment, which makes the actual programming part of it 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 say we are ‘working in E-Prime’, talk about ‘an experiment written in E-Prime’, or even complain that ‘E-Prime has crashed again’, we generally mean E-Studio. You may be delighted – or disheartened – to learn that E-Prime is actually a suite of software, composed of a number of programs besides just E-Studio.
We will talk about these other programs throughout this book, but, generally, they are pretty straightforward and a quick summary should be enough to get you started.

**E-Studio** is based on (or perhaps merely inspired by) Visual Studio and can therefore be called an IDE: an integrated development environment. The graphical user interface is a convenient way to code (develop) the experiment, which you can do by simply dragging and dropping objects onto a timeline. This makes the daunting task of developing experiments as simple and user-friendly as familiar Windows programs like PowerPoint – at least, on the face of it. However, it’s not quite true that no real programming (i.e. coding or scripting) is required: almost all original experiments will at some point require you to write at least a few lines of code; and, more importantly, doing so can save you a lot of time. We will see how this works in later chapters. The experiment in E-Studio is stored as an .es3 file (.es or .es2 file in E-Prime 1 and 2).

**E-DataAid** is a program that can read E-Prime output. Whenever an E-Prime experiment is run, a unique data file is created (an .edat file). These .edat files can’t be opened directly by Microsoft Excel or SPSS, but you can use E-DataAid to convert them into those 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, and much more.

**E-Merge** does nothing more than merge data. Typically, when you have run a certain number of subjects, you will end up with that number of .edat data files. Of course, you can analyse each one in turn, or even import each one separately into SPSS, but then you run the risk of making mistakes or encountering errors, all of which add up to a higher chance of data corruption (or just a mess). Not to mention how time consuming it is to process data files one by one. Luckily, with E-Merge you can merge all those datafiles into one large file. To merge a set of datafiles, take the following steps:

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

This generates an .emrg 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 .edat file is generated. However, when the experiment is running (also called runtime), a .txt file is created and updated with data from each trial as it happens. This .txt file contains the same data as the .edat file, but it is rather inconvenient to analyse. However, if E-Prime crashes part-way through the experiment, you can still open E-Recovery and recover the (partial) data, as follows:

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

Now you have an .edat file that is fully equivalent to all the others, except that it lacks some trials (if you are lucky, only a few). Since this is all E-Recovery does, we won’t be covering the program again in this book.

E-Run allows you to run experiments. When you have created an experiment in E-Studio, you will then compile it to create an .ebs (encrypted E-basic script) file that you will actually use to collect data. This .ebs file is 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. With a car object, you can do a lot of things, such as driving, steering and disappearing into the sunset. In programming, we call such possible actions methods. If you have a look at some code, you’ll see that the method associated with an object is indicated like this: Object. method(parameters). For example, the code Car.drive(forward) would make the car object drive forward: Car is the object, drive is the method and forward is a parameter of the method drive.

The other important feature of objects is that they usually have properties. A car object can have a range of properties: is 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. Here, Car is the object, colour is the property, and dark blue is the parameter of the colour property.
Two other concepts that are important for understanding OOP are **instances** and **inheritance**. Our car, for example, is not just any car: it’s OUR car! In other words, `ourCar` is an instance (or a ‘token’ of) the object or type `Car`. This matters for programming, because if we add something to `ourCar`, such as heated seats or fluffy dice, this does not alter every instance of `Car`. Of course, there are some things that all cars come equipped with. Inheritance is also about the types and tokens: our car inherits certain properties and methods that are generally true for most cars, such as that it comes with a steering wheel and four wheels. But 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 for being able to program basic experiments. However, since they are so important for modern-day programming, we hope that they will become clear to you as you get to grips with E-Prime.

**Conceptualising an experiment**

When you are designing an experiment, you might be tempted to turn on your computer and start programming straightaway. However, before you start programming, you should try to visualise what your experiment will look like. This may sound obvious, but it remains an important step that is easily skipped, with nasty, if not fatal, consequences. So, try to conceptualise the experiment first by asking yourself the following questions:

- **What kind of design do you need?** A between-subjects design or a within-subjects design?
- **Which variables will 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, will you acquire data regarding reaction time, error rates, or something else entirely?
- **Does your experiment contain blocks of trials?** If so, what order should these blocks be presented in?
- **Does your experiment have a practice block?** How do you want to 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 (also called the ‘inter-trial interval’ or ITI)?

Which order are trials presented in? In a random order? In a fixed order? Or in a semi-random order?

To make the task of programming easier, it can be useful to first draw a flow chart that shows the structure of the experiment. An experiment typically contains a hierarchy of Procedures. The global order of events in the experiment is determined by the main Procedure (called ‘SessionProc’ in E-Prime). This one is depicted on the left. Sub-procedures are depicted to the right of the main Procedure.

A flow chart should contain a number of different elements:

- **Event**: Refers to a specific event during the experiment, for example, the presentation of a picture, text or sound. Here you should indicate what happens in the event, 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. Here you should indicate the name of the Procedure, the number of repetitions of the Procedure and the order of the repetitions (e.g. random).

- **Decision**: This is when the Procedure branches into two options. The diamond indicates a criterion (e.g. the response is correct, or reaction time is less than 1000ms). 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.

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 your time and paper. After all, some people are apparently perfectly happy working in a messy environment! However, even if that’s you, we hope you will try to see the benefit of these flow
charts, as we will be using them to illustrate various aspects of E-Prime as we go along.

Now, let's get back to it and have a look at E-Studio!
The structure of E-Studio

Here’s an example of what an experiment looks like when you 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 (available only from E-Prime 2 on; E-Prime 1 users may want to consider the Clock.Scale code described in Chapter IV). E-Prime 3 users can click the Test icon, to quick-start the experiment from a particular List in the experiment.

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’s a quick trick: right-click on them, de-select Large icons and voilà: more screen real-estate!

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

D. The **Experiment Explorer window** allows you to switch between different views of your experiment: the Structure view, the Browser view, and the Attributes view. These are shown in separate tabs. The **Structure view** (default) shows the hierarchy of the experiment and sequence of events within it. 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. Clicking on the tab next to Structure shows the **Browser view**, which shows all the objects you’ve created for the experiment. This view also allows you to copy-paste objects using the clipboard. When you copy-paste an object, E-Prime creates a new instance with the same properties as the original object. Finally, the tab next to the Browser view can provide a list of all the **Attributes** created within Lists in the experiment.

You can also copy objects by dragging the object with your mouse while holding down the `ctrl` key. If you simply want to reuse the same object at a different location in the experiment, hold down `ctrl+shift` while dragging.

E. This area – which covers most of the screen – we call the **Workspace**. This is where you can edit elements of the experiment in an easy, visual, way. When you double-click on an object in the Structure window, it appears in the Workspace. In the figure, two windows are floating in the Workspace: a **Procedure object** called **SessionProc** and a **Slide object** called **ASlide**. The procedure displays a timeline showing the sequence that objects will be run in. Note that this sequence includes the other object currently displayed in the Workspace, i.e. ASlide.

F. The **Output window** has two tabs: **Generate** and **Debug**. As soon as you run the experiment, E-Prime will tell you if there is an error in the output window. We mainly use the Debug tab for our own information. For example, if you
notice that your experiment keeps on crashing halfway through, the code here should give you more information about *where exactly* the problem lies.

G. The **Advisor** window shows the Experiment Advisor, in which E-Prime displays useful tips as you work (only in E-Prime 2 Professional and E-Prime 3). These tips are meant to help you design experiments to ensure timing accuracy and *portability*, i.e. the extent to which your experiment looks and works similarly across different systems.

**E-Objects**

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

- A **List** organises the hierarchy of the experiment, enabling randomisation, balancing and so on.
- A **Slide** is a container type of object which can simultaneously present text, images, sound and more.
- A **FeedbackDisplay** shows information (feedback) to the participant about their response to a stimulus.
- An **InLine** is used to add custom E-Basic code.
- A **TextDisplay** displays one or more lines of text.
- An **ImageDisplay** shows pictures.
- A **MovieDisplay** displays video clips.
- A **SoundOut** presents an audio file.
- A **SoundIn** is used to record sound.
- A **Wait** object delays the running of the subsequent object.
- A **Label** indicates a special location in the procedure. The program can ‘jump’ backwards or forwards in order to repeat or skip part of the procedure.
A **PackageCall** is a reusable block of E-Basic script. PackageCalls are often used in procedures to run custom code for connecting equipment such as eye-trackers, EEG amplifiers, and so on. Such equipment tends to be very technical and, of course, specialised (eye-trackers and amplifiers are quite different things!), so we won’t be covering them in this book. However, you can find information in the online documentation (Help > Documentation).

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

### Procedures, Lists and TextDisplays

A **Procedure** is the highest unit in the hierarchy of an E-Prime experiment. 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, blank 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 allow us to repeat and reorder Procedures. They determine the way in which Procedures are repeated, for example, by 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, and the columns indicate the properties (called **attributes**) of these items.
By clicking on the Add Level icon or the Add Multiple Levels icon, you can add one or more rows, respectively.

Likewise, clicking on Add Attribute or Add Multiple Attributes adds one or more columns.

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-added 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. You can see this in the example (right), where PracticeList uses a Procedure called ‘TrialProc’.

Lists are explained in more detail in Chapter II. For now, just remember that Weight indicates the number of repetitions of a particular item.

**The TextDisplay Object**

TextDisplays present text in a uniform format. That is to say, the ‘PRESS SPACE’ in the below figure is either completely in bold or not, aligned to the left, or the right, and so on. It cannot present just a few letters in bold, blue, at the top right of the screen while showing the rest just regular in the middle. Yet, they are surprisingly powerful objects, as they can both show simple stimuli, and collect responses, all with only a few clicks. Indeed, they are pretty much the only object that is needed to make a simple Stroop experiment (see Chapter II). Furthermore, you will find out later that other objects, such as the Slide and the FeedbackDisplay, can actually contain TextDisplays. Now let us have a look at the TextDisplay in practice.
When you drag a TextBox from the Toolbox area onto a Procedure and double-click on it, you should see something like the screenshot, except that it is usually named differently and doesn’t say ‘PRESS SPACE’.

The name of the TextDisplay is shown in 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 one a unique and descriptive name, without funky characters such as commas, semicolons, spaces, etc. The example above is actually a good example of how not to name an object: all sensory presentations in experiments are ‘stimuli’, so this name is 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 name you give the object, which provides a handle for calling its properties. Note that some names are prohibited because they would interfere with generating the script: names like ‘If’, ‘Then’, ‘TextDisplay’ and so on should be avoided, as well as spaces and special characters.

**Tag:** Here you can fill in an identifier for your TextDisplay (although we generally skip this).

**Notes:** Here you can write some comments about your TextDisplay. This can be very useful if, for instance, you want someone else to work on your experiment – adding a description here will let them know exactly what the object does, and why. In general, with any code you write, commenting is extremely important: it is very easy to forget what your own code does. However, it is even more important that your design is clear enough by itself, rather than your colleagues (or future you) needing to dig around in the various Notes that are buried within the dozens of objects you inevitably end up with.
Generate PreRun/PostRun: These properties affect the timing of the .Load method. If they are set to TopOfProcedure, the properties of this TextDisplay will be loaded at the beginning of the Procedure. This can provide a timing benefit by having the loading take place at a moment when accurate timing is not absolutely critical. This means that, when the object is shown, there is less likely to be a timing delay or error. However, note that this is only convenient if the properties of the object are fully known at the beginning of the 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 an experiment that’s running, but it can be little too immediate: exiting that way doesn’t close devices and no .edat file is generated (but see E-Recovery). E-Prime 2 Professional and E-Prime 3 users can press control+alt+backspace during the experiment to terminate experiments more gracefully.

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 graphical interface shown earlier (the figure showing the ‘PRESS SPACE’ TextDisplay). However, it’s important to remember that what the Text field refers to is actually a property of a TextDisplay, i.e. .Text. You can use this when you start writing your own script in later chapters.

AlignHorizontal, AlignVertical: Adjusting these properties allows you to specify the position of the text horizontally and vertically.

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. In RGB values, (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. Based on this, you can play around with mixing 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 can be a useful feature.

**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. According to Psychology Software Tools, this function is *deprecated*. This is a software term meaning a function that is no longer in use and kept there only to provide backward compatibility – it will eventually be removed and so should be avoided.

**WordWrap:** Specifies whether E-Prime should automatically start a new line when the text reaches the edge of the space it’s in. Without WordWrap on, E-Prime will simply cut off the text where the screen ends.

**Display Name:** E-Prime 2 Professional and E-Prime 3 add the feature to use multiple displays (screens) independently. Here, you can select the display that you want 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 in E-Prime 2, this is 75% by default (but not in older or newer versions of E-Prime). It might seem odd to have a frame set to 75% by default (hence the decision to change this back to 100% in E-Prime 3). However, there are certain advantages to showing a smaller TextDisplay. It improves timing (since only part of the screen needs to be changed) and enables you to sequentially add more layers to a display. But before we get into that, let’s first see how a Frame is positioned on the screen.
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 a reference point in the frame for defining how the frame is going to be placed on the screen. **X** and **Y** then specify the horizontal and vertical position of this 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 on the piece of paper, while the parameters **X** and **Y** specify the position of the pin on the pinboard (see examples below).

![Frame positioning examples](image)

**BorderColor**: Shows the colour 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 size 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 note that in E-Prime 2 (but not 3!) the default setting for Bold is on.
Duration/Input tab

This is probably the single most important tab. Here is where you adjust the timing of the stimulus. Stimulus duration adjustments are crucial, whether you’re doing some old-school behavioural experiments or a fancy new neuroscience study. It is also where you define which input devices (such as keyboard, mouse, Serial Response Box) are used to record responses. Generally speaking, this is the tab that deals with all things related to responding: ‘what should happen after a response?’, ‘what was the correct answer?’, and so on.

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 set the duration to infinite.

Timing Mode: E-Prime is praised for its timing accuracy and the developers claim that E-Prime can have sub-millisecond accuracy: in other words, that any random timing errors that creep in will altogether have a standard deviation of less than one millisecond. However, this all depends on which other processes are running in the background, which hardware is installed, and whether unrelated software like Norton Antivirus is allowed to run alongside E-Prime.

More on timing issues later: 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’. To validate the timing of your experiment you will likely need additional hardware and technical expertise.

The other way to cope with a certain type of timing error is the PreRelease. Suppose you want 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 reduce the stress E-Prime puts on your processor, you can have a TextDisplay prior to the picture you want to use and set some PreRelease for that TextDisplay. This PreRelease time is used to load the upcoming picture, sound or other heavy object into memory while the current TextDisplay is still being shown. Then, when it’s showtime, E-Prime has already loaded the object, which reduces the chance of onset errors in the next stimulus.

Notice that, since the Production Release of E-Prime 2 (i.e. also in E-Prime 3), 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 possibility that subsequent objects will be run before the PreReleasing object ends can throw up issues. 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. If timing is not critical, the easiest solution is to set PreRelease manually to 0 (ms).

The Data Logging property has a few options allowing you to log various timing and response information from the TextDisplay. We suggest you leave this untouched and select the logging properties in the Logging tab, as E-Prime generally logs far too much if you allow it to. However, after witnessing a colleague who managed to log everything but the variable she was interested in, we came to the conclusion: 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 made 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, if you enter ‘abcd’, all four keys (a, b, c, d) are seen by E-Prime as valid responses. Pressing e or A (shift+a), for example, will not do anything. To use the space bar or other special keys, use curly brackets and capital letters: {SPACE} for example. The default {ANY}
allows any character: this is not recommended for a serious experiment, since accidental key-presses should not be counted as real responses. Our E-Primer website (http://www.e-primer.com/) has a post on how to use special keys (e.g. {RIGHTARROW}) in your experiment.

Whereas there are generally several keys that are Allowable, usually only one of them is the correct response. It is important to understand the fundamental difference between Allowable and Correct. The allowable set of responses covers the range of possible responses; the correct response is the response that the participant should have made. As a rule of thumb, the set of Allowable responses is generally the same for each trial. Meanwhile, the correct response is generally different for each trial, and there is typically only one correct response (although one can, in E-Prime 2 Professional and E-Prime 3, have multiple correct responses). Note that it is probably best to think of the notion ‘correct response’ in terms of accuracy, rather than appropriateness.

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 fail 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 period 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 option that is selected by default. That way, if the duration of a stimulus is 2000 ms, a response will still be logged if it follows 1999 ms after the onset of the stimulus. However, it is possible to ignore extremely slow responses (i.e. 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 to 20 ms, but the Time Limit to 1000 ms. Then, responses will still be 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 erased from 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 onset and offset synchronisation. To grasp what synchronisation is, it’s first necessary to understand something about the way computer monitors work. Computer monitors come in different types, with the two main ones being cathode ray tube (CRT – or, the ‘old’ type) and liquid crystal display (LCD – or, the ‘flat’ type).

Each pixel you see on the screen is updated sequentially: that is, from top to bottom. 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 per second. In other words, they have a refresh rate of 60 Hz. Some CRT or newer LCD/LED monitors may even have a refresh rate of 100 Hz, which is actually better for doing experiments. Users of LCD screens may also need to take into account the ‘response time’, which is how quickly a monitor can make a pixel active and inactive again (lower is better) to avoid seeing an after-image (called ‘ghosting’). Crucially for psychologists, especially those investigating perception and (subliminal) priming, the presentation of visual stimuli is constrained by the timing and other characteristics of the monitor.

Consider, for example, an experiment where we want 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 (which is still standard, especially for LCD), the screen is updated every 17 ms (i.e. 1000/60). At the moment that E-Prime is programmed to show the smiling face, it sends a command to the screen to do so, but there is no way to know exactly what the result is: if the computer only just finished updating the screen, there may be an extra wait of 17 ms (after which the smiling face is shown for the next 17 ms); alternatively, it may show only half a smiling face, because the updating cycle had just reached halfway down the screen on its route from top to bottom. This phenomenon is called screen tearing. The same may then happen with the word that the participant is required to react 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 lasts (17 ms). We therefore end up with a timing error of 7 ms, as it is impossible to show anything for less than 17 ms.
For this reason, we recommend the use of (often ridiculously old) CRT monitors that have higher refresh rates and therefore shorter refresh cycles. For example, a refresh rate of 100 Hz gives a refresh cycle of 10 ms (this also has the advantage of giving nice, round numbers). Then we can safely use Onset Sync by default. However, obtaining these monitors is becoming increasingly difficult (though you could try to get hold of a newer, high refresh-rate, low response-time LCD/LED screen).

Note that the Onset Sync is set to ‘vertical blank’ by default. This helps avoid the screen tearing issue mentioned above (and which we will look into again 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 of statisticians. For example, if you wish to check whether E-Prime really presents your stimuli for a certain number of milliseconds, you can log the duration error.

You can select as many values to log as you like, but try to be practical in your choices. You may want to log the time it took for a participant to read the introduction screen, in which case, you can log the RT for this display. However, we have yet to hear from a psychologist who is interested in the timing accuracy of their ‘Welcome to the experiment!’ screen, so you probably don’t need to log the OnsetDelay. There is one caveat though: it is easier to ignore data you have recorded than to retrieve data you never saved in the first place!

Often, a psychological experiment requires only one response for each individual trial. For example, in a Stroop task, each word displayed (e.g. ‘WHITE’ written in
black letters) requires one reaction (e.g. ‘black’). In E-Prime terms, this translates to the TextDisplay (which displays the word ‘WHITE’) collecting responses. For this TextDisplay object, our favourite logging properties would then be:

- **CRESP**: Correct response. As stated above, this 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 the time the stimulus was programmed to be presented and the actual time its presentation started.
- **DurationError**: Difference between the duration the stimulus was programmed to be shown on screen and the actual time it was shown for. Strictly speaking, this is calculated as: 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 response relative to the beginning 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 execution of the stimulus relative to the beginning of the experiment (ms). Note, however, that this is not necessarily when the stimulus ends. For example, a visual stimulus remains on screen as long as no other stimulus overwrites it, and an audio file that’s 4 seconds long may contain 2 seconds of silence.

**Tutorial I: A simple RT experiment**

Do you, like Athena (right), have lightning reflexes? Believe it or not, many first-time participants who are unfamiliar with psychological experiments are eager to know ‘how well they did’. In the interests of diplomacy, 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 if you take some time to think it through first. Firstly, and most importantly, what is it that you want your participants to see during an experiment? Think about what you know from your own experiences with psychological research in the lab? Try to imagine examples, rather than just coming up with a general definition. So, instead of just thinking that you’ll show ‘Stroop-like stimuli’, ask yourself: what is a Stroop-like stimulus? ‘Well’, you answer, ‘something like the word RED written in blue’. Great! The next step is to define the Procedure of a trial as the sequential presentation of stimuli like this.

Common elements of an experiment include:

**Trials:** Typically, this includes:
- A fixation: This is a stimulus that is often shaped like a crosshair or plus sign. Its purpose is to ‘warn’ the participant that the interesting stimulus is imminent.
- The target: The interesting stimulus itself, which the participant should respond to.
- Some form of feedback (if your experimental design calls for it).

**Blocks:** These are defined by the number and type of trials they contain. For instance,
- A training block may contain, say, 20 trials; these are intended to get the participant used to the experiment.
- A testing block contains more trials, depending on the range of outcome measures, the number of conditions and so forth.

**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.). Best practice is to save your experiment on the computer’s fastest hard drive. 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 library of experiments that might come in handy in the future.
• In the Structure view, double-click on SessionProc; you will see a timeline pop up:

• Drag a List from the Toolbox to this timeline and call it ‘BlockList’ (this is a conventional name; you can use any 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. You’ll see that it is also possible to click on the downward-pointing triangle next to the name and change the Procedure to the existing one: SessionProc. **DO NOT DO THIS!** For some reason (perhaps simply because it’s there!) you may feel the urge to do this, but don’t. We have seen many students astonished to find that this causes E-Prime to crash. On the face of it, there seems to be no reason. But if you think about it, what you are doing is asking E-Prime to run something that is *already running*. Compare it to a recipe. Imagine you want to bake a cake (some of our colleagues are avid Great British Bake-Off watchers!). Let’s say you’ve found a gloriously tasty cake recipe online (one that also happens to be deliciously millisecond accurate). You start reading the instructions. ‘Step 1’, it reads, ‘get a cake recipe’. No doubt you would be as confused as the cat above. E-Prime feels the same way, and deals with the unbearable confusion by simply (and understandably) crashing.

• 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 (selecting Yes then No). Notice that *not* creating the new procedure will result in a bug. Again, this is pretty understandable. You tell E-Prime you want it to do something. E-Prime replies: ‘What do you want
me to do?’, and you say ‘I’m not going to tell you’. Pro-tip for life: don’t do that in your relationship. Pro-tip for E-Prime: do create that procedure.


- Edit TrainingList and set the weight of the first (and only) row to 10. In the Procedure column, enter the name ‘TrialProc’.

- Edit TestingList and set the weight of the first (and only) row to 20, then write 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. The two blocks run the same Procedure: the training block runs it 10 times, the testing block runs it 20 times. You can check whether you have successfully completed this step by checking that your screen looks the same as the screenshot below.
Another possibility would be to drop the BlockList and simply run the TrainingList and TestingList consecutively in your SessionProc. However, we think that the BlockList solution has many advantages. For example, when you’re testing whether your experiment runs correctly, you can easily skip parts of it by setting the weights of the respective rows to 0. Plus, 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 an ideal place to nest Lists used for counterbalancing or 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 element in programming your experiment. Here, you will be showing your participant a fixation for 500 ms, and a target for an infinite amount of time (or, until a key is pressed).

- Double-click on the TrialProc and drag two TextDisplays to the timeline. Name the first one ‘Fixation’ and the second one ‘TargetStimulus’.

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

- Edit the TargetStimulus to display 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 {SPACE} as the only Allowable key. Mind the capital letters: case is important here. Now, set the space bar as the only correct key (although this is normally not the case!). Accept the standard type of logging.

- Your experiment should run now, so try it out. It is good practice to run your experiment frequently, because this makes it easier for you to diagnose problems, or debug the program. Make sure you run your experiment with a subject number that is not 0, or else nothing will be logged. Remember: if you want to abort the experiment early, you can always press ctrl+alt+shift (or ctrl+shift+backspace). You may notice that when you start your experiment, the resolution of your screen changes. In Chapter III we will discuss how to adjust 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 have a look through the various columns. For example, notice that the trial number starts at 1 and goes up to 10, because the first block (the TrainingList) is finished after 10 trials; then it starts at 1 again but now goes to 20, because the second block (the TestingList) is finished after 20 trials.

- We want to know what our participant’s average reaction time was, so that they can go home happy. For this, TargetStimulus.RT is the most important. You’ll probably find that several values are well below the average (approaching an improbable 0) and some are well above the average. It seems that a bit of filtering needs to happen in order to get a clear picture of your participant’s RT.

- 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. Then, when all the values you want to include are selected, press the space bar and click on OK. In this way you prevent outlying RT values from distorting your mean RT values.

> Consider what would happen if you saved this analysis and wanted to apply it to another dataset later. There would probably be new unique RT values in your dataset that weren’t included in your checklist. In other words, you would have to re-select the relevant RTs. In that case, it might be preferable to use the Range alternative. Click Range... and set the first range to Greater than or equal to 100, combined with the second range set to 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 (one of the authors of this book) was 191 ms.
One of the reasons why many people use E-DataAid in conjunction with E-Prime is that it’s very easy to 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:

![Image of crosstab interface]

- So, Michiel was almost twice as fast (79.5 ms faster on average) after training a bit. Not bad.

**Exercises**

- Add an introduction screen to the start of your experiment, with infinite duration, that terminates when the participant presses a certain unique key (your instructions might contain something along the lines of ‘press C to continue’).
• Add a ‘thank you and goodbye’ screen to your experiment.

• Use the mouse instead of the keyboard as InputDevice for the TargetStimulus. To do this, you essentially follow the same process as you did to set the keyboard as an input device. For the mouse, the response keys are defined as 1 (left mouse button) and 2 (right mouse button). Therefore, if you enter ‘12’ as Allowable, both mouse buttons are allowable responses.

• Polish your experiment. E-Prime’s default appearance is not all that aesthetically pleasing. Using the information above regarding the Frame, adjust the Fixation and TargetStimulus such that participants see the following screen:

![Press Space](image)

• Are you wondering how we did that? To get this effect, we changed the following: 1) set the Fixation background to black; 2) set the TargetStimulus background to navy and foreground to white; 3) set the width and height of the TargetStimulus to 50%; 4) change the Font to Freestyle Script; and 5) set its BorderColor to ‘gray’, and BorderWidth to 4.

• Now that you’ve got everything working and seen what you can do to adjust the appearance, it’s time to polish the look of the rest of your experiment too. Try to make things to your liking.

• Design an experiment to test the following hypothesis: ‘it is easier to respond to green than to red’. Here’s the idea: a lot of the time, a red light is a sign that we need to stop doing something. So, a psychologist could hypothesise that we’ve internalised this rule to the extent that we automatically suppress all action when a red light appears. Seems plausible, but is it true? Let’s find out.
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 grey background; the target should now not have any word at all, but just display 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, make it refer to ‘RedProc’ and ‘GreenProc’:

At this point, you may be thinking: wouldn’t it 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 you’re right (always nice) and you will learn how to implement this method using attributes.

Lastly, randomise the selection between these two Procedures (see the List properties) so that it’s not possible for the participant to guess in advance what colour the target will be. Edit the RedProc and GreenProc so that they are identical in appearance, duration and response, except that the background of the target differs.

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

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

- Make sure you have more than one .edat data file (to generate them, run your experiment with different subject numbers) and put them in a directory. Now start E-Merge and browse to the directory containing the files. First, filter that directory so you can only see the .edat files and nothing else (click the Filter button) by using *.eda* (the * sign acts as a wildcard, so here we specify that is can begin and end with any character).

- Select the .edat (E-Prime 1), .edat2 (E-Prime 2), or .edat3 (E-Prime 3) files and click the Merge button (use the standard merge). Now you will see that an .emrg* file has been created, containing all the information from the two separate files.

- Open the .emrg file in E-DataAid and analyse the data from all your participants (or sessions) in one go.
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 from scratch  
• Program a mouse tracking task  
• Program an embodied cognition task using Slide Buttons

Dear reader, if you have been with us all the way from the start, it’s great to see you here. You have made your way through six chapters full of useful information and are now well beyond just coding your own reaction time experiments. As a reward for all your hard work, we are going to teach you how dazzle your friends with something beautiful, interactive and, some may say, exhilarating. In this chapter we will break out of the decidedly ‘80s realm of reaction times and button presses and will introduce you to a fancy new piece of hardware: the mouse. As a bonus (if you can cope) you will also learn how to program your own questionnaire using E-Basic code.

It is important to remember that E-Prime’s greatest strength lies in critical timing and interaction with equipment that requires such accurate timing, like EEG or eye-tracking apparatus. So, when you find yourself desperate to know how ‘betrayed’ your participant feels (on a scale of 1 to 5, naturally), or want to get your feet wet with immersive user-interfaces combining drag-and-drop behaviour for mobile communication technology, E-Prime may not be the platform of choice. After all, the seemingly simple Windows event of ‘the user clicked on button A’ is not a piece of cake in E-Prime land, and questionnaires are nowadays incredibly easy to make using tools available online.
However, we often find it useful (or sometimes just fun) to add aspects of interactivity, say, one or two clickable objects, to our accurately-timed experiments. In addition, it can be useful to have all the data on one participant – including questionnaire responses – in one place. It certainly beats the old-school approach: no more the ultimate boredom of transcribing pen-and-paper data into SPSS!

Therefore, in this chapter we will show you how to engage with Slide objects to a deeper extent, how to create a simple questionnaire using E-Basic code 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 code in InLine objects.

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 take a look at 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:

\[
\begin{align*}
\text{Slide1 As Slide} \quad & \text{Default As SlideState} \\
& \text{Text1 As SlideText} \\
& \text{Image1 As SlideImage}
\end{align*}
\]

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:

\[
\text{Slide1.ActiveState}
\]

Similarly, if we want to call up the Procedure that manually draws the Slide1 object on the screen (you’ll see in a minute why we would want to do this), we may use the method:

\[
\text{Slide1.Draw}
\]

**SlideState object: Properties and methods**

At the level of the SlideState, 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 the CColor conversion function), we can use this code:

\[
\text{Slide1.States("Default").BackColor = CColor("Red")}
\]

Note that we refer here to the subordinate SlideState Default via the States property of the Slide object. This 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 coordi-
nates. If no SlideImage or SlideText object exists at the specified coordinates, an empty String “” is returned.

```eba
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.

```eba
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 reference 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 referenced SlideText (so, in this example, SlideText ‘Text1’ in the Default state of Slide1) by adding the following line of code:

```eba
theSlideText.BorderWidth = 3
```

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

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

You can then also change, for instance, the filename associated with the image (so in this example, SlideImage ‘Image1’ in the Default state of Slide1), using the following line of code:
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** can be used to add complicated hand movements to your experiment or include questionnaires in E-Prime. In the Properties of your Experiment, the mouse is activated by default, but its cursor is not necessarily shown. To change this, simply set the **Show Cursor** property to Yes (this is set as the default in E-Prime 3). This is illustrated in the figure below:

![Figure showing how to access the mouse in E-Prime](image)

Showing a mouse cursor might be annoying if you’re running an experiment that depends on manual responses. Wouldn’t it be great if you could toggle the cursor on and off, depending on the particular time point in the experiment? Well, that is a good idea and it’s actually 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:

```vbs
Mouse.ShowCursor False
```

Another property you may find useful is the possibility to read the mouse coordinates in pixels. Simply refer to its `CursorX` and `CursorY` properties, as shown in the example below:

```vbs
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 main possibilities:

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

2. You would like to have *real-time* online control over your Slide (i.e. not only following a button or mouse click).

The sections below describe basic setups and the principles behind them. Use them as templates: they can easily be adapted to suit your own needs.

**Instant 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 point-and-click tasks.

Imagine you want to show some visual feedback immediately after a user left-clicks in a Slide object, and that you want this Procedure to be repeated again and again until the user clicks with the right mouse button instead. As feedback, you want the SlideText position to move to the position of the mouse cursor following each mouse click.

To implement this, we need three objects: a Label, a Slide object, and an InLine object. In addition, we need to add a SlideText sub-object to the default Slide state. See the examples below.
Make sure to set the Slide Duration to Infinite, and to Terminate once the user has clicked the mouse. In addition, set ShowCursor to Yes in the experiment properties. Because duration is set to infinite, we do not need to worry about premature execution of the subsequent InLine: the default value of PreRelease is already 0 in this case.

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

```vbnet
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 the Slide object to keep on refreshing its content each time the user clicks the mouse.

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’d like to learn more about the InputMask object, feel free to study its complicated details in the E-Basic help file.

Real-time online control

Some situations require you to have real-time access to a device and instantly refresh the Slide object. For example, you might want to program an approach / avoidance task where you can pull towards (enlarge) or push away (shrink) images with a joystick. For this you need to be able to sample mouse or joystick data as fast as possible and instantly update the screen to match the new cursor position. Other examples of real-time online control applications include mouse tracking tasks, and implicit learning tasks that read complicated hand movements using the mouse or other devices.

To achieve continuous online control, we’ll be using 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 of your choice (i.e. some value >0).

In addition, set ShowCursor to Yes in the experimental properties.
This is the content that the ProcessSlideResp InLine object should contain:

```vbs
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
    Display.WaitForVerticalBlank
    Slide1.Draw
    Sleep 10
Loop
```

The Do While Loop structure here draws Slide1 again and again, until 10,000 ms (i.e. 10 seconds) has elapsed since Slide1.OnsetTime. However, it’s useless to have faster loops than the screen refresh rate. For this reason, we decided to add a Display.WaitForVerticalBlank statement and a sleep command: this creates a loop that is as fast as (but not faster than) the refresh rate of the monitor.

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

**Tutorial XII: A simple questionnaire**

As you found out in Tutorial IV (Chapter 3), E-Prime 3 allows you to create questionnaires without a single line of code. Programming a simple questionnaire using E-Basic is much more challenging. On the other hand, programming it yourself will certainly be an educational experience. It will challenge you to integrate all the E-Basic knowledge you have learned so far. Are you up for it? Let’s get started!

Firstly, we need an ItemList that determines which questions and answers should be presented. The participant is allowed to make multiple selections. Selected items should be indicated by a black border.

Fortunately, to help us along the way, we can adapt the design that was used earlier in this chapter to implement continuous feedback after mouse clicks.
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 clicks the mouse. Set ShowCursor to Yes in the experiment properties.

Name the relevant SlideText objects ‘Option1’, ‘Option2’ and ‘OKbtn’.

Step 2: Declare variables

Add the following lines of code to the SetSlide InLine object:

```vba
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. Using these variables we can later set the BorderWidth property of the items 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 has definitely been made (Responses.Count > 0). In addition, we use the HitTest method to find out whether the user has clicked any object at the cursor coordinates specified. If so, the variable strHit will contain the name of this object.

```vbnet
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
```

Add these lines of code to the InLine object. Don’t be a copy-paste monkey: try to make sure you understand the code you’ve just entered!

Step 4: Process the hit test data

Now let’s think about how to process the HitTest data. If the user clicks on Item1, we want to show some feedback, that is, indicating which item has been selected – or, indeed, deselected – 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):

```vbnet
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 need to figure out 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 OKbtn object. In other words, if strHit <> “OKbtn”, we want to jump back to the RefreshSlide. The following code implements this:

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

So, what if the user did click on the OK button? Well, then the program proceeds to the end of the TrialProc and finishes. But, wait a moment! When that happens, we first want to store the selections that the user made, in the .edat file. To do this, we need to replace the If-Then statement given above by adding 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! All that’s left is to test whether your first questionnaire in E-Prime works the way it should.

Tutorial XIII: A mouse tracking task

As a fresh young student, you are likely to have efficient motor control over your mouse, but what happens when you get old? Well, it’s quite probable that your motor performance will decline with age. In this tutorial we are going to program
a mouse tracking task that you could use to investigate motor ability across different age groups.

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 no click is registered, after 10 seconds.

Here, we basically need to adapt the continuous online control design described earlier 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 Time-Limit 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 with 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. Make sure both the Frame Height and Frame Width of the SlideState are set to 100%.

**Step 2: Set the online Slide control**

In the ProcStim InLine script, add the following lines of code:

```vba
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 needs to be added is provided below:

```vba
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 take a closer look at these lines of code.

The most important structure is the **Do While – Loop** structure, which calls the TrackingSlide.Draw method repeatedly as long as the mouse response is pending and the time limit has not been exceeded.

The other important part to note are the lines starting with ‘CurrTarget.Y =’, and ‘CurrCursor.Y =’. Here, we are changing the position of the Square and Dot images in the TrackingSlide object.
Because we want the target to move in a sinusoidal fashion, we first calculate $y$, which uses the `Sin()` function in combination with the time passed (i.e. `ClockNow - TrackingSlide.OnsetTime`) and the duration of the period (here, 2000 ms) in radians (i.e. $2 \times \pi$) to produce a value between -1 and +1.

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

However, we want to calculate the value in pixels measured from the top of the screen. Therefore, we use the formula $(\text{Display.YRes}/2) - (\text{Amplitude} \times y)$. So, when using a 1024 x 768 resolution, we can use $(768/2) - (50 \times y)$ 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 variables PeriodDur and Amplitude in the example above have fixed values throughout the experiment. In such cases, you might consider declaring constants instead, using the `Const` statement. See the E-Basic help file for more information about the `Const` statement.

Tutorial XIV: Embodied cognition, political leaning and Stroop featuring buttons

Embodied cognition is, of course, hot. Stroop is, sadly, not hot. The same goes for social science versus cognitive psychology. In the following exercise, we will try to somehow blend all these ingredients into an elixir of grant-awardable bliss.
In public discourse, the political spectrum is often represented on a linear spectrum from ‘left’ to ‘right’. Embodied fans may argue that people use physical symbols in order to represent abstract concepts. If this is the case, we might expect the political space to be represented in terms of a physical, horizontal space. Therefore, when we think of the political left and right, we might be primed towards the physical left and right, and therefore inclined to act in these directions! Great hypothesis, don’t you think? Unfortunately, it’s a case of “sounds good; doesn’t work”.

Not to be deterred by the possibility of low replicability, we will adapt a dusty old-fashioned Stroop paradigm to become a sparkly new embodied experiment. Along the way, you will learn how to use the new Button sub-object available in E-Prime 3. So, even if you object that this study has already been published (it should be, it’s brilliant), we are just going to carry on regardless – in any case, we can always come up with a better version, as long as we don’t let science stand in the way of creativity...

**Step 1: The basics**

The experiment should have two parts:

1. A questionnaire that measures the participant’s political leaning on a left-to-right scale. Conveniently, you already know how to do that: see Tutorial IV. If you do not use E-Prime 3, you can adapt the questionnaire from Tutorial XII to show the items from Tutorial IV. Make sure this questionnaire is provided at the beginning of the experiment.

2. A Stroop task (see Chapter I Tutorial I and the exercises from Chapter II). If you’re still having trouble creating a Stroop task in under 30 minutes, have a go at the exercises in Chapter II. For easy reference, here is the basic structure and TrialList:
The task (given in the instructions) is simply this:

“If the colour is red, click left. If the colour is green, click right. The word itself is irrelevant.”

**Step 2: Adding the button**

The TrialProc has a simple fixation (+) of 1000 ms, a StroopSlide, and a blank inter-trial interval (ITI) of 1000 ms. You should have no trouble implementing the fixation and inter-trial interval, but the StroopSlide should look like this:

Notice that it has a normal TextDisplay (width and height 20%, silver background, BorderWidth 2), flanked by two buttons. Buttons are new in E-Prime 3, and provide interesting new ways of interacting with E-Prime, whether that be by mouse (clicking) or touch (tapping). You’ll find that the text can’t be altered by double-clicking the button. You will have to change this in the sub-object properties of the button under the Theme tab:
In order to get the button input to work, we do not use the normal keyboard response input, or even the mouse device (which might seem intuitive), but instead add the Button as a Device item:

![Properties: StroopSlide](image)

Unlike the other Devices, the Button gives a response that is defined in the General tab of its sub-object properties, indicated by curly braces (to distinguish it from the letters that make up the name, such that ‘red’ is not the same as ‘r’, ‘e’, ‘d’). For example, the response of the red button (redbtn) above is identical to its text, i.e. ‘red’. Therefore, clicking this button will give the response {red}, as is also helpfully shown by E-Prime. Notice that if there is no response explicitly defined, the response will default to its name (here, {redbtn}).

![Properties: redbtn](image)

Now the only thing we need to do is get our List to agree with this. If the {red} and {green} responses are properly defined, you can simply add the CorrectResponse
{green} in the List for each green StimColour, and the CorrectResponse {red} for each ‘red’ StimColour.

One of the convenient aspects of the Button is that with minimal changes, we can easily adapt it to different circumstances. Do you want to use a touch screen instead of your mouse? It will work immediately. Do you want to define regions of interest using an eye-tracker? Again, the Button can easily do what used to require a lot of coding.

**Step 3: Bodily politics**

Now, of course it is possible that a political leaning right or left will facilitate physical responses in the same direction (or at least give rise to some kind of mental default). But what exactly does that predict? Will there be a smaller incongruence effect if political leaning is right but response location is left, because the left response is *shifted to the right*? Or maybe any response in the political direction is facilitated, congruent or not?

Any outcome of the experiment sounds clickbait worthy, and the worst way to approach openness in an experimental paradigm is to just embrace all the possible ways that the prediction could conceivably work. The one that ultimately works will suddenly feel intuitively reasonable, and the urge to just report this one effect will become quite strong. This is an example of the Texas Sharpshooter fallacy: the cowboy who shoots blindly around himself ends up drawing a nice circle when a few bullets inevitably cluster together (unlike the rest of the examples in this book, this is one we don’t want you to try at home).

A long day in the lab, though, can leave you feeling a bit ‘Once Upon E-Prime in the West’. So maybe you feel like this is a perfectly reasonable strategy, and it’ll be even better if you have more bullets– uh, hypotheses! So now you think: what if reaction times aren’t affected, but people simply respond further to the left or right depending on their political leaning? Let’s test this hypothesis.

Simply add the following InLine code just after the StroopSlide:

```vbscript
Dim theButtonResponseData As ButtonResponseData
Set theButtonResponseData = _
  CButtonResponseData(StroopSlide.InputMasks.Responses(1))
c.SetAttrib "ClickPos", theButtonResponseData.CursorX
```
Now, E-Prime saves the X position of the click – we can now formulate the hypothesis that the more right-wing someone is, the higher the X. This will enable us to plot every participant’s physical response pattern and correlate it with their political one, which is so unlikely to work that even we haven’t tried it yet.

But if you do try it, don’t forget to cite the E-Primer!

**Exercises**

- Add extra trials to the List of the questionnaire in Tutorial XII. Add an InLine script to the beginning of the TrialProc to make sure that each question always starts by showing all options as 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 flow chart to think through how this will work before you start programming.

- Now adapt the questionnaire in such a way that the participant has to choose one out of eight options. Again, draw a flow chart before programming.

- The questionnaire you created can be useful as a manipulation check. Add the questionnaire to the end of the ego depletion experiment from Chapter III Tutorial VI. Ask participants to rate on an 8-point scale how tired, thoughtful, excited, happy, worn out, 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 has been programmed in a roundabout way, needing many Lists, Procedures and InLines. We won’t explain all of it here; the only thing you need to know for now is that 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 selection 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). Item ID fields that are left empty will not be used, nor will option fields that are left empty. If you want, you can set the PreSelection field, which allows a particular item number to be selected in advance.

- Now let’s call the Procedure RunOneSheetOf10Items at the proper location (e.g. the final row in your BlockList) in your own experiment. If you would like to present more than one sheet of questions, make sure you add more rows that call this procedure.

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

**Additional Exercises**

- Add additional frequency conditions (1 Hz, 2Hz and 4Hz) to the TrialList of the experiment in Tutorial XIII. Adjust the code so that the value in Period-Dur 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.

- Now for an Advanced Additional Exercise (we know you’re ready!). 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. Here’s a hint: read ‘Arrays’ when they refer to ‘vectors’; also, RMS refers to the root of the mean of the squares of all elements in the Array (or 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

$$\text{AI} = 1 - \frac{E}{A}$$

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.

Once you've implemented it, check whether your feedback script works correctly!

- Let’s add counterbalancing to the experiment in Tutorial XIV. Assume we want half of our participants to see red on the left and the other half to see green on the left. Clone the SlideState, name it redright, and swap the left for the right button to allow both possibilities. Enable counterbalancing of the response locations by using the ActiveState of the StroopSlide and a nested List (see examples in Chapter 3), but without using any code.
References


Appendix: Overview of available E-Objects

Here we provide an overview of all E-Objects, listing their main functions 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 documentation, which can be downloaded from the E-Prime support page.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Name</th>
<th>Description</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Procedure</td>
<td>Used to determine the order of events in an experiment.</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>List</td>
<td>Contains rows of items with specific properties (attributes). Lists usually call Procedures.</td>
<td>I,II,III</td>
</tr>
<tr>
<td></td>
<td>TextDisplay</td>
<td>Displays one or more lines of text.</td>
<td>I</td>
</tr>
<tr>
<td></td>
<td>ImageDisplay</td>
<td>Displays pictures.</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>Slide</td>
<td>Presents a combination of text, images, movies and sound.</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>FeedbackDisplay</td>
<td>Provides specific feedback based on the participant’s response to objects presented earlier in the experiment flow.</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>MovieDisplay</td>
<td>Displays a movie clip.</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>SoundOut</td>
<td>Presents a sound file (.wav/.mp3/.wma).</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>SoundIn</td>
<td>Records sounds.</td>
<td>Not in this book</td>
</tr>
<tr>
<td></td>
<td>Wait</td>
<td>Waits for a specified time without changing the visual output.</td>
<td>III</td>
</tr>
<tr>
<td>Icon</td>
<td>Name</td>
<td>Description</td>
<td>Chapter</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td><img src="image" alt="InLine Icon" /></td>
<td>InLine</td>
<td>Used to add E-Basic script at a specific location in the experiment flow.</td>
<td>IV</td>
</tr>
<tr>
<td><img src="image" alt="Label Icon" /></td>
<td>Label</td>
<td>Indicates a particular location on the timeline (Procedure). The program can ‘jump’ backwards or forwards to a Label, in order to repeat or skip a part of the Procedure.</td>
<td>III</td>
</tr>
<tr>
<td><img src="image" alt="PackageCall Icon" /></td>
<td>PackageCall</td>
<td>Contains reusable blocks of E-Basic script written by users of E-Prime 2 (often used in Procedures which are used repeatedly, or for connecting equipment such as an eye-tracker to an E-Prime experiment). As packages are not covered in this book, please see the E-Prime 3 documentation for a more detailed description.</td>
<td>Not in this book</td>
</tr>
</tbody>
</table>
About the authors

Michiel M. Spapé obtained his PhD in cognitive psychology at Leiden University on the topic of cognitive control (Stroop!). He has since worked as a postdoctoral researcher at the University of Nottingham and Aalto University, as assistant professor at Liverpool Hope University, and is now senior researcher at the University of Helsinki. His topics of interest include emotion, decision making, social neuroscience, haptics and pretty much anything else. Most of all, he likes combining these with machines and biosignals, working with big computers, trackpads, eye-trackers, tactors, Kinects, EEG, EMG, MEG, ECG devices and all that jazz. His favourite type of study is one that reheats a venerable psychological effect, spices it up with new tech and adds a generous slosh of neuro-sauce, before being served at the academic table. He has four cats.

Rinus G. Verdonschot trained as a psycholinguist, specialising in applied linguistics and cognitive neuroscience. His main work is in the fields of psycho- and neurolinguistics, focusing on language production, reading and bilingualism. He has considerable practical experience working in diverse experimental labs (including EEG and 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 both undergraduate and graduate level and collaborating with and learning from other researchers.

Henk van Steenbergen trained as an experimental psychologist at Leiden University in the Netherlands where he is currently assistant professor. He likes to combine behavioural, physiological and neuroscientific methods to study human behaviour, in particular emotion-cognition interactions. With a background in electronic engineering, he likes to implement cool hardware and software in his psychological research. Having been a programmer since his teens, he is convinced that good programming skills make for more efficient and more enjoyable research, a message he tries to convey to graduate students as he teaches them how to program in E-Prime.
Index

& operator 119
= operator 149

Abs 118
ACC 38
ActiveState 63, 70, 146, 147, 149, 157, 187, 206
Addition (+) 116
Allowable 34, 35, 42, 45, 57, 211, 228
Array 161, 167, 170-175, 177-181, 183, 205
Arrays of a user-defined data type 173
Assignment 114, 149

Basic 86
Beep 126
Between-subject manipulations 94
Bit 82, 83, 84, 91, 215-218
Bit-depth 77-79, 82-84, 100
Bitmap 58, 59, 68
Bits and bytes 216
BlockList 40, 42, 66, 71, 143
Boolean 113, 140, 141, 148, 149, 173
BorderColor 32, 45, 146,
BorderWidth 32, 45, 106, 146, 188, 194, 195, 197, 201
Browser 25
Buffer Mode 81
Buffer Size 80
Button 200, 201, 203
Byte 81, 216
c.GetAttrib 125, 126
c.SetAttrib 125, 126
Chronos 80, 90, 207, 208, 210-212, 221-229
CInt 119
Clock.Read 127, 133
Clock.Scale 24, 129, 132
Clone slidestates 62
Code substitution 120, 137, 140
Collection Mode 209
Colon Syntax 95, 103, 104
Combine conditional expressions 139
Combining Arrays and Lists 172
Commands 126
Comment 115, 129, 179, 180, 195
Comment and uncomment blocks of code 115
Comparing values 141
Comparison operators
   Equal to (=) 141
   Greater than (>) 141
   Greater than or equal to (>=) 141
Less than (<) 141
Less than or equal to (<=) 141
Not equal to (<> 141
Compiled 112
Conditional expression 136, 138-141,
148, 165-169, 182
Const 199
Context (c.) 125
Control+alt+backspace 30
Control+alt+shift 24, 30
Conversion functions
CIInt 119
Convert a String to a Single 119
Convert a String to an Integer 119
Convert numerical variables to a
String 119
CSng 119
CStr 119
Convert a String to a Single 119
Convert a String to an Integer 119
Convert numerical variables to a
String 119
Correct 35, 38, 63, 64, 157
Counterbalancing 42, 52, 77, 94
CRESP 38
CRT monitors 37, 91, 92
CSng 119
CStr 119
CursorX 190
CursorY 190
Cyberball game 135, 151, 207
Cycle 36, 37, 52-54, 92, 105, 128, 130,
182, 199, 205
Data Logging 34, 142
Debug 25, 42, 114, 115, 117, 122, 127,
129, 133, 188
Debug.Print 114, 115, 122, 127, 133
Decision making
ElseIf 155
Else 136, 148
End If statement 136, 148, 149,
155, 161, 196
If-Then-Else-EndIf statement 136,
148
Nested If-Then structures 138
Select...Case 155
Decision 22
Declaring and using Arrays 170
Devices 30, 33, 34, 77-79, 88-90, 127,
192, 202, 207-210
Digital audio 77-79, 82-84
Dim 113, 118
Display hardware 91
Display.Refreshrate 93, 125, 137
Distracting sound 96
Division (/) 116
Do – Loop Until 166, 167, 182
Do – Loop While 167
Do Until – Loop 166
Do While – Loop 167, 193
Dot operator 174, 186
Double 113
Duration 33-35, 38, 42, 44, 46, 57, 62,
68, 69, 72, 73, 81, 87, 94, 99, 102,
106, 110, 130, 133, 145-147, 156,
157, 176, 191, 192, 197, 199, 210,
215, 229
Duration tab 33, 102
DurationError 37, 38, 127
Duration tab 33, 102
Echo 130
E-DataAid 13, 17, 18, 43, 47, 103
EEG 27, 55, 56, 90, 107, 185, 208, 215,
221, 223
Else 136, 148
ElseIf 155
E-Merge 13, 17, 18, 47, 103
End 142, 143, 144, 183, 226
End Action 33, 35, 87, 193
End Movie Action 85, 102, 156
End Sound Action 82
Experiment Advisor 26
E-Object
FeedbackDisplay 26, 28, 34, 49, 63-65, 69, 73, 108, 157, 176, 233
ImageDisplay 26, 49, 58, 84-86, 128, 233
InLine 26, 34, 113, 115, 129, 131, 142, 143, 146, 147, 154, 159, 168, 177-179, 186, 190-195, 197, 218, 219, 221, 234
Label 26, 74, 86, 87, 141, 162, 190, 234
MovieDisplay 26, 84, 85, 101, 102, 155-157, 233
PackageCall 27, 34, 234
Slide 25, 62, 63, 147, 158, 186-188, 190, 192, 196, 197, 219, 233
SoundIn 26, 215, 233
SoundOut 26, 78, 80, 81, 84, 85, 99, 233
TextDisplay 25-35, 38, 57-59, 72, 74, 86, 87, 99, 102, 108, 109, 125, 127, 128, 131, 132, 158, 197, 201, 210, 215, 228, 229, 233
Wait 26, 86, 126, 128, 157, 233
EOF 225
Equal to (=) 141
E-Recovery 13, 19, 30,
E-Run 19, 24, 129, 133
E-Studio 13, 17-19, 23, 24, 39, 61, 63, 75, 79, 86, 112, 129

Event 22, 33
Experiment Explorer window 25
Experiment properties 80, 88, 124, 127, 191, 194
Experimental design 50, 54, 122, 135
Exponentiation ^ 116
Feedback 63
FeedbackDisplay 26, 28, 34, 49, 63-65, 69, 73, 108, 157, 176, 233
File types
.ebs 19
.edat 18, 19, 30, 47, 75, 93, 109, 110, 124, 125, 127, 148, 196, 204
.emrg 18, 47
.es 18
.txt 19, 225
.wav 78-83
FileExists() 225, 226
Filename 58, 80, 85
FirstFrameTime 85
Font tab 32, 87
Format() 120
For-Next loop 163-165, 169, 172, 175, 178, 181, 184
Frame 31, 32, 45, 61, 72
FrameRate 85
FramesDisplayed 85
FramesDropped 85
Full Script 112
Functions 118
General tab 30, 85, 202
Generate 25, 112
Global level 117
Global variables 113, 159
GoTo 86, 87, 141
Greater than (>) 141
Greater than or equal to (>=) 141
Green 57
Height 31

If-Then statement 136, 148, 149, 155, 161, 196
If-Then-Else-EndIf statement 136, 148
ImageDisplay 26, 58, 84-86, 128, 233
Implicit Association Task 70, 73, 207
Inheritance 21
InLine 26, 34, 114
InputBox 123, 124, 126, 130-132, 171
InputDevice 34, 45
Instance 21, 25, 174
Instant feedback after mouse clicks 193
InStr() 121, 131
Integer 113, 115, 118, 119, 154, 170, 173
Integer division (\) 116
Inter-trial interval (ITI) 22, 42, 110, 201
Jitter 77, 93, 94, 128
Jump 26, 82, 85, 87, 141, 142, 155, 196, 234
Keyboard 33, 34, 42, 45, 57, 69, 89, 90, 130, 151, 202, 208, 209, 228
Label 26, 74, 82, 85-87, 141, 142, 154, 161, 162, 190
Latent variables 50, 51, 56, 66, 97
LCase 121
LCD monitors 36, 92
Left 121
Len 120
Line Input 225
List methods
c.GetAttrib 125, 126
c.SetAttrib 125, 126, 168, 225
<ListName>.GetAttrib 169, 178
<ListName>.SetAttrib 125, 168
<ListName>.Terminate 143, 157
Log 34, 35, 37, 73, 89, 127, 131, 138, 157, 158, 192
Logging tab 34, 37
Logging timing 127
Logical operators
And 139
Not 139
Or 139
Truth tables 139
Xor 139
Long 113, 118
Loop 81, 99, 162, 163, 165, 166, 171, 180, 182, 183, 192, 193, 205, 225
Looping with Labels 161
Loops
Do – Loop Until 166, 167, 182
Do – Loop While 167
Do Until – Loop 166
Do While – Loop 167, 198
For-Next loop 163-165, 169, 172, 175, 178, 181, 184
Loops with conditional expressions 165
LPT port 215, 216, 218-220, 229
Manifest variables 50, 51, 54, 56, 66, 97
Manipulate the content of a String 120
Match Desktop Resolution At Run
Time 77, 91
Mathematical functions
Abs 118
Random 119
Sin 119, 199
Mathematical manipulations 116
Mathematical operators
  Addition (+) 116
  Division (/) 116
  Exponentiation ^ 116
  Integer division (\) 116
  Modulus division (mod) 116
  Multiplication (*) 116
  Negation (-) 116
  Subtraction (-) 116

Menu 24
Method 19, 21, 30, 46, 53, 114, 125,
  128, 142, 144, 186-189, 195, 198,
  218-220
Mid 122
Mirror 59
Modulus division (mod) 116
Mouse 18, 25, 33, 45, 89, 90, 154,
  186-198, 201-203, 205, 207
Mouse tracking task 192, 196, 197
MovieDisplay 26, 84, 85, 101, 102, 155-
  157, 233
MsgBox 122, 124, 126, 133, 137, 139,
  162, 165
Multiplication (*) 116
Negation (-) 116
Nested List 42, 93-97, 99, 103-105, 109,
  206
No repeat after 53, 54, 175
Not 139
Not equal to (<>) 141

Object 19, 21, 25-31, 33, 34, 62-65, 87,
  112, 114, 123, 125, 128, 158, 187,
  188, 190, 210, 211, 214, 224, 229
Object oriented programming 19, 186
Offset 52, 53, 56, 218, 219, 230
Offset Sync 36
OffsetTime 38, 224
Onset Sync 36, 37, 92, 128
OnsetDelay 37, 38, 127
OnsetTime 38, 81, 133, 193, 199, 224
Open 225, 226
Operator 43, 111, 116, 135, 139, 149,
  159, 174, 186
Or 139
Output window 25, 75, 114, 122, 124,
  127
PackageCall 27, 34, 234
Pan Control 82
Parallel port 215-218, 220, 223, 224,
  229
Permutation 52, 53, 95, 96
Port 209
Position Time Format 81
Position 32
PreRelease 34, 38, 65, 132, 145, 157,
  158, 191, 220
Print 226
Procedure 22, 24, 27-30, 33, 39-41, 44,
  58, 66, 68, 70, 73, 74, 78, 84, 87, 94,
  98, 99, 102, 104, 113, 116, 118, 128,
  141, 142, 157, 165, 183, 187, 189,
  190, 204, 205, 233
Procedure object 25
Properties
  ACC 38
  ActiveState 63, 70, 146, 147, 149,
  157, 187, 206
  Buffer Mode 81
  Buffer Size 80
  Collection Mode 209
CRESPI 38
Duration 33-35, 38, 42, 44, 46, 57,
  62, 68, 69, 72, 73, 81, 87, 94, 99,
  102, 106, 110, 130, 133, 145-147,
  156, 157, 176, 191, 192, 197, 199,
  210, 215, 229
DurationError 38, 127
End Movie Action 85, 102, 156
End Sound Action 82
Filename 58, 80, 85
FirstFrameTime 85
FrameRate 85
FramesDisplayed 85
FramesDropped 85
Jump Label 87
Mirror 59
OffsetTime 38, 224
OnsetDelay 37, 38, 127
OnsetTime 38, 81, 133, 193, 199, 224
Pan Control 82
Port 209
Position Time Format 81
RESP 38, 147
RT 35, 43, 64, 128, 133
RTTime 38, 133
ShowCursor 189, 191, 192, 194
Start Offset 81
Start Position 85, 156
Stop After 81, 85, 99
Stop Offset 81
Stop Position 85, 156
Stretch 85, 155
Volume 82, 100, 212
Weight 28
Properties window 25, 29, 85
Quasi-random trial selection 175
Questionnaire 73-76, 185, 186, 189, 190, 193, 196, 200, 204, 205, 207
Random 119
RandomizeArray 180
Reading a text file 225
Recover 19
Refresh rates 37, 92, 210
Reset / Exit tab 53
Reset sampling 54, 155
RESP 38, 147
Right 122
RT 35, 43, 64, 128, 133
RTTime 38, 133
Run 24
Sample rate 77, 84
Samples 52, 54, 83
Scope 117
Screen tearing 36, 37, 92
Script 174
Script window 112
Select...Case 155
Selection tab 52, 54, 94
Sending signals using the parallel port 215
Sequential 39, 49, 52, 53, 96, 98, 101, 105, 155, 175, 176
Serial port 90, 208, 209
Serial Response Box 33, 34, 90, 208
SessionProc 22, 25, 27, 40, 42, 101, 102, 104, 114, 118, 142, 154, 159, 177, 179
Set 188
ShowCursor 189, 191, 192, 194
Showing an image 58
Showing multiple images and layers of texts 62
Simon Effect 65, 70, 115
Sin 119, 199
Single 113
Size 31
Sleep 126, 127, 193, 219
Slide object 25, 28, 62, 63, 147, 158, 186-188, 190, 192, 196, 197, 219, 233
SlideState 62, 63, 147, 186-188, 198, 206
SoundDevice 78, 82, 88, 89
SoundIn 26, 215, 233
SoundOut 26, 78, 80, 81, 84, 85, 99, 233
SRBOX 90, 208-214, 224, 228, 229
Start Offset 81
Start Position 85
Startup Info 88, 89, 94
Startup 89, 125
Stop After 81, 85, 99
Stop Offset 81
Stop Position 85, 156
Stretch 85, 155
Stretching 59, 60
String 87, 98, 113, 114, 116, 119-123, 130, 131, 169-171, 173, 187, 188
String functions
Format() 120
InStr 121, 131
LCase 121
Left 121
Len 120
Manipulate the content of a String 120
Mid 122
Right 122
Trim 121
UCase 121
Stroop task 37, 49, 57, 76, 90, 101, 102, 132, 157, 200
Structure 25, 40
Structure window 25, 28
Subject 53, 80, 94, 103, 125, 138, 139
Sub-objects within a SlideState 188
Sub-procedure 22
Subtraction (-) 116
Sync tab 36
Syntax 114, 119-122, 124, 126, 136, 137, 163, 166-169, 171, 225, 226
Task events 218, 223
Terminate 24, 30, 33, 35, 82, 102, 143, 156, 157, 191, 193, 194
Test 24
Text 93
TextDisplay 25-35, 38, 57-59, 72, 74, 86, 87, 99, 102, 108, 109, 125, 127, 128, 131, 132, 158, 197, 201, 210, 215, 228, 229, 233
Throw error if invalid refresh rate 93
Time Limit 35, 64, 192, 193, 197, 198
Toolbox 24, 29, 40, 87, 114
Transparency 61
Trim 121
Truth tables 139
Tutorials 13, 112, 135, 207, 211
Type statement 173
Types 36, 51, 56, 110, 113, 120, 173, 174
UCase 121
Ultimatum game 55, 144, 207
Until 166
User-defined 113, 173, 174, 177
User interactions 190
User Script 112, 113, 117, 154, 173, 177
User tab 117
Variable 50, 51, 55, 93, 97, 113-118, 120, 122, 124, 131, 135, 136, 140, 154, 165, 169, 170, 171, 173, 177, 178, 180, 181, 183, 188, 195
Variable types
Array 161, 167, 170-175, 177-181, 183, 205
Boolean 113, 140
Double 113
Integer 113, 115, 118, 119, 154, 170, 173
Long 113, 118
Single 113
String 87, 98, 113, 114, 116, 119-123, 123, 130, 131, 169-171, 173, 187, 188
Variant 113
View 24, 67, 112, 117
Visual search 87, 96
Voice key 90, 208, 210, 212, 227, 228
Volume 82, 100, 212
Volume Control 82, 100
Wait 26, 86, 126, 128, 157, 233
Weight 28
While 167, 193
Width 31

Windowed Mode 133
Workspace 25, 63
WritePort 218, 219, 230
Writing to a text file 226
X 32, 68, 106, 198
XAlign 32
Xor 139
Y 32, 68, 106, 198
YAlign 32
E-Prime® is the leading software suite by Psychology Software Tools for designing and running psychology lab experiments. *The E-Primer* is the perfect accompanying guide: It provides all the necessary knowledge to make E-Prime accessible to everyone. You can learn the tools of psychological science by following *The E-Primer* through a series of entertaining, step-by-step recipes that recreate classic experiments. The updated *E-Primer* expands its proven combination of simple explanations, interesting tutorials and fun exercises, and makes even the novice student quickly confident to create their dream experiment!

Featuring:
- Learn the basic and advanced features of E-Studio’s flexible user interface.
- 15 step-by-step tutorials let you replicate classic experiments from all psychology fields.
- Learn to write custom code in E-Basic without having any previous experience in programming.
- Second edition completely revised for E-Prime 3.
- Based on 10+ years of teaching E-Prime to undergraduates, postgraduates, and colleagues.

**Michiel Spapé** is senior researcher in Cognition and Social Neuroscience at the University of Helsinki, Finland.

**Rinus Verdonschot** is assistant professor in Psycho- and Neurolinguistics and works in Japan.

**Henk van Steenbergen** is assistant professor in Cognitive and Affective Neuroscience at Leiden University, the Netherlands.

*I truly value this book! [...] Working through it led me to proficiency with E-Prime, something that has significantly expanded my abilities as a researcher. It’s an invaluable resource that I often refer back to when programming new experiments, and I recommend it to everyone I know who is learning to use E-Prime.*

- **Matthew Robison**, postdoctoral research associate at Arizona State University.

[www.lup.nl](http://www.lup.nl)

Leiden University Press