User's Guide

PSYCHOLOGY SOFTWARE **TOOLS**

E-Prime® 2.0
User's Guide

E-Prime® 2.0 User's Guide Manual
PST-101338
Rev 1

Copyright

Trademark

The E-Prime 2.0 User's Guide (PST-101338) is for research purposes only.

Table of Contents

Chapter 1: Introduction

1.1 E-Prime 2.0

E-Prime 2.0 is a comprehensive suite of applications offering audited millisecond-timing precision, enabling researchers to develop a wide variety of paradigms that can be implemented with randomized or fixed presentation of text, pictures, sounds, and movies. E-Prime 2.0 allows the researcher to implement experiments in a fraction of the time required. As a starting point, the *E-Prime 2.0 Getting Started Guide* provides much of the basic knowledge that is required to implement and analyze such experiments. This manual, the *E-Prime 2.0 User's Guide*, provides detailed descriptions of E-Prime 2.0's components and information regarding the more advanced system features.

1.2 What to Know Before Reading This Manual

Before reading this manual, a user should have a basic knowledge about operating personal computers and experimental research design and analysis. Familiarity with the Windows operating system is important, and at least one undergraduate course in experimental research methods is recommended. The understanding of terms such as "independent variables" and "random sampling" is assumed. If this background is lacking, we recommend that the user first become familiar with how to use the Windows operating system and reference an undergraduate text on experimental research methods. This text also assumes the user has worked through the *E-Prime 2.0 Getting Started Guide* accompanying the E-Prime 2.0 CD. *Appendix B: Considerations in Computerized Research*, provides a basic overview of experimental research and the relevant terminology.

1.3 Machine Requirements and Compatibility

E-Prime 2.0 is compatible with Microsoft Windows XP, Vista, Windows 7 and 8. The full E-Prime 2.0 installation requires approximately 100 MB disk space. A summary of the machine requirements is listed below.

Minimum	Recommended
Windows XP SP3 32-bit/Vista SP1 32 and 64-bit/7 SP1 32 and 64-bit/Windows 7 and 8.	Windows XP SP3 32-bit/Vista SP1 32 and 64-bit/7 SP1 32 and 64-bit/Windows 7 and 8.
Pentium 4 1.8 GHz Processor	Pentium Dual-Core or Multi-Core Processor 2GHz or higher
512MB RAM	1024MB RAM or higher
DirectX video card	DirectX video card with 128MB RAM or higher
USB Port	Sound hardware as recommended*
	CD-ROM
	USB Port
	Serial Port**
	Internet Connection

* Please see sound timing recommendations in <u>article 1307</u> of the E-Prime 2.0 Knowledge Base.

** Recommended hardware add-on: <u>Serial Response Box</u>. USB-to-Serial conversion cable may be used to receive serial data. Please see <u>article 1835</u> in the E-Prime 2.0 Knowledge Base for more details.

⚠**NOTE:** *Most experiments that present digital movies as stimuli will require a machine with a specification higher than the recommended configuration shown above.*

1.4 Installation, Registration, and Validation Instructions

⚠ **NOTE:** *Before using E-Prime 2.0, you must complete BOTH the Installation process described in this section, AND Registration and Validation below. Your serial number and hardware key will be required for Registration and Validation.*

Installations:

1) Close all Windows applications.
2) Insert the E-Prime 2.0 CD into the CD-ROM drive.
   The installation program will launch automatically. Alternatively, the installation may be launched manually by running the SETUP.EXE program from the CD-ROM drive, or by accessing the Add/Remove Programs option in the control panel.
3) The Setup program prompts you as needed throughout the installation process. At the first prompt, click the type of installation desired and begin the installation.
4) The Setup program displays the End User License Agreement. Read the license agreement in full and accept the terms of use. Click Next to continue.
5) The Customer Information Dialog box appears. Enter or verify your User Name and University/Company Name, and serial number. Click Next to continue.
6) Select the type of installation to run and follow the instructions on the screen.
   The complete installation is recommended. This places E-Prime 2.0 in the ...Program Files(x86)\Pst\E-Prime…folder if running the 64 bit editions of the operating system and Program Files\PST\E-Prime if running the 32-bit editions of the operating system. Select the Custom Installation option if you need to change the location of the installation folder.
7) Click Install to begin the installation.
8) At the conclusion of the installation, the InstallShield Wizard Completed dialog box appears. At this point, the installation is complete. However, prior to using E-Prime 2.0, you must complete the Registration and Validation process.

To perform the Registration and Validation process at this time, confirm that the "Launch E-Studio" box is checked, connect your hardware key, and click Finish. E-Studio then launches automatically. Continue with the below instructions, beginning with Step 3.

To perform the Registration and Validation step later, first uncheck the "Launch E-Studio" check box and then click Finish. When you are ready to perform the Registration and Validation, follow the instructions below.

Registration and Validation

In order to register and validate the E-Prime 2.0 system a serial number is needed. The serial number is located on the Registration Card that ships with your E-Prime 2.0 system. Place the serial number in an appropriate place so that it is readily accessible. Once E-Prime 2.0 is installed, the serial number can also be found in the About E-Studio dialog box located on the Help menu. Users MUST provide the serial number for Product Service and Support.

1) Connect the hardware key to the USB port of the local machine (single user) or the server machine (network license).
2) Launch E-Studio.
3) The E-Prime 2.0 Auto Update screen appears. We recommend that you accept the default settings. Click OK.
4) The E-Prime 2.0 Registration screen appears. Enter your name, institution, and serial number and click OK.

1.5 Installation Options

Complete Installation

The complete installation provides the entire suite of E-Prime 2.0 applications. The table below lists the applications included with the complete installation.

Application	Use	Relevant Input File(s)
E-Studio	Creation or modification of experiment specifications, Generation of .ebs2 files	Experiment Specification (.es2)
E-Run	Stimulus presentation, Data collection, Demonstration	E-Basic Script (.ebs2)
E-Merge	Merging of data files	Data Files (.edat2, .emrg2)
E-DataAid	Examination, Analysis, Export, Regeneration of tables or analyses	Data Files (.edat2, .emrg2) Analysis Files (.anl)
E-Recovery	Conversion of E-Run text files to .edat2 files	Text data files (.txt) generated by E-Run
Factor Table Wizard	Stimulus List creation	none
Codec Config	Configuration of codecs necessary to play movies	Movie and audio files

The table below lists the additional applications that are included with the E-Prime 2.0 Professional version:

Application	Use	Relevant Input File(s)
StartupInfo Editor	Creation or modification of StartupInfo files, used to load name/value pairs into the Context of an experiments	StartupInfo files (startupinfo)
Package File Editor	Creation or modification of package files, used to create reusable blocks of script in experiments	Package Files (epk2)

Custom Installation

The custom installation provides the ability to choose which applications will be included or omitted during installation. Although used rarely, this installation option may be used to save space on the computer's hard drive. For example, if developing an experiment is the only utility needed, then the E-Prime 2.0 Development Environment is the best option. The E-Run option is equivalent to a Subject Station or Run-Time Only install, and the E-Prime 2.0 Analysis and Data File Utilities option is ideal for just merging and analyzing data.

During installation, the installation program will prompt the user for the specific applications to be installed. Check all of the applications to be installed, and leave unwanted applications unchecked. The Description field lists the components included with a selected option. The amount of space required by an installation option and the amount of space available on the machine are displayed below the checklist.

Subject Station Installation

The E-Prime 2.0 installation includes a Subject Station install requires a smaller footprint than the full installation. The Subject Station installation supplies the E-Run application only. This option does not permit the development of new programs, or the generation of .ebs2 files from Experiment Specification (.es2) files. The E-Run application does not require a hardware key during installation. The Subject Station installation may be installed on up to 25 machines in a lab in order to collect data from more than one participant at a time. Refer to the End User License Agreement (EULA) for additional information regarding the use of the Subject Station installation.

To install the Subject Station during installation, select the Subject Station Installation option. Repeat the Subject Station Installation process for all data collection machines. You will need to enter the serial number when launching E-Run for the first time after installing the Subject Station Installation. It is highly recommended that the researcher run a program to test that the installation was completed successfully prior to attempting to run actual participants. The BasicRT.ebs2 file is included with the Subject Station installation for this purpose (…My Experiments\Samples\BasicRT\BasicRT.ebs2).

Run-Time Only Installation (Run-Time Only CD)

A Run-Time Only installation option is available by separately purchasing the Run-Time Only CD. This option is available for collecting data at a site other than where the full E-Prime 2.0 system is licensed (e.g., multi-site study). The Run-Time Only installation will allow data collection by providing the E-Run application, but none of the other E-Prime 2.0 applications are included.

The Run-Time Only installation does not require the hardware key for installation. To install the Run-Time Only version of E-Prime 2.0, insert the Run-Time Only CD into the CD-ROM drive of the computer needed for data collection, and follow the installation prompts. Please be aware, that the serial number will need to be entered when E-Run is launched for the first time.

⚠ **NOTE**: *All previous versions of the E-Prime 2.0 Run-Time installation must be uninstalled by the user before installing a new version of the E-Prime 2.0 Run-Time. A side by side multiple Subject Station can be set up with a Run-Time Only installation.*

1.6 Hardware Key and Licenses

E-Prime 2.0 is a copy-protected system requiring a hardware key that connects to the computer's USB port. The hardware key copy protection scheme limits the number of development machines to the number of licenses purchased. The hardware key is required for registration of E-Prime 2.0. Other applications require the hardware key to be installed depending on the type of license in use.

⚠ **NOTE**: **If the hardware key fails:** *Users should not experience any problems installing or using E-Prime 2.0 if the hardware key is correctly in place and the serial number was correctly entered. However, if the hardware key should fail, contact PST Product Service and Support immediately via http://www.pstnet.com/support.*

Single User License

A single user license allows the user to develop experiments on one computer at a time. The hardware key must be connected during registration of E-Prime 2.0, and while developing experiments within E-Studio. E-Studio will not load if the hardware key is not connected. With the single user license, users may install E-Prime 2.0 on multiple machines, and simply move the hardware key to the machine to be used for experiment development. If more than one concurrent development station is necessary, additional single user licenses must be purchased.

Run-Time License

The runtime application (E-Run) is not copy protected. E-Run may be installed on up to 25 machines within a single lab, and used for data collection on multiple machines simultaneously with a Run-Time Only installation.

Network License

A network license permits the development of E-Prime 2.0 experiments on multiple machines used concurrently within the licensed department, up to the number of purchased seats. The network license requires the hardware key to be connected to a network machine and the network connection to be active during the registration and validation process, and while developing experiments within E-Studio.

1.7 Useful Information

How to abort an experiment early

Press Ctrl+Alt+Shift to terminate the E-Run application. An "Abort Experiment?" prompt will then appear, asking if you want to abort the experiment. Click "Yes" to abort the experiment, or click "No" to continue running. If you click "Yes", then the E-Run application will terminate. Within the E-Studio application, an error dialog will appear displaying the "Experiment terminated by user" message.

⚠️**NOTE:** *The Ctrl+Alt+Shift key sequence is designed to be used during experiment development as a tool to help you quickly check various experiment parameters. When an experiment is terminated prematurely by using the Ctrl+Alt+Shift key sequence, the .edat2 data file is NOT generated and therefore it is likely that the data from the current experiment run will not be logged. The E-Recovery utility can sometimes be used to recover a portion of the data that has been collected to date, but it should not be used routinely in typical data collection sessions. This utility is described in the E-Prime 2.0 New Features/Reference Guide (Chapter 6: E-Recovery).*

Conditional Exit (Professional feature)

Press Ctrl+Alt+Backspace to terminate the E-Run program. Unlike the Ctrl+Alt+Shift termination method, this method of aborting an experiment will produce a data file after termination. This feature is only available for E-Prime 2.0 Professional users.

Sharing pre-developed programs

E-Prime 2.0 license owners are free to distribute any files they create through use of the system. Files created by users include Experiment Specification files (.es2), E-Basic Script files (.ebs2), data files (.edat2, .emrg2), and analysis files (.anl). However, the license agreement prohibits distribution of any part of the E-Prime 2.0 system, including the runtime application. Therefore, in order to view or run any shared files, the recipient must have access to an E-Prime 2.0 installation.

The E-Prime 2.0 Evaluation Version may be used to view experiments and run them under restricted conditions. To access an .es2 file, the recipient must have the full E-Prime 2.0 installation, or a custom installation that includes E-Studio. To run an .ebs2 file, the recipient must have the Subject Station installation, a custom installation including E-Run, or the Run-Time Only installation. To open or merge data files, the user must have an installation including the data handling applications (i.e., E-DataAid and E-Merge). For further information about sharing pre-developed files, refer to the *E-Prime 2.0 New Features/Reference Guide* (Chapter 3, Tutorial 3, Task 5: Sharing Pre-Developed Programs).

For those experiments that you have developed, you should always retain the .es2 file, as this is the source file for the experiment specification. The .ebs2 file, which contains the generated E-Script, cannot be edited and therefore is not adequate for maintaining your experiment library.
For details on archiving files and developing experiment libraries see:
• *Chapter 3: Implementing your Experiment in E-Studio (Page 28)*
• *3.10 Stage 10: Archiving Experiments and Results (Page 83)*
• *3.11 Stage 11: Research Program Development (Page 84)*

The table below shows the default paths to your personal documents folder. Again, the path on your particular machine may have been modified by your administrator:

Operating System	Path to your personal documents folder("My Documents" or "Documents")
XP	\<drive\>\Documents and Settings\\<user name\>\My Documents\
Vista	\<drive\>\Users\\<user name\>\Documents\
Windows 7	\<drive\>\Users\\<user name\>\Documents
Windows 8	\<drive\>\Users\\<user name\>\Documents

Chapter 2: Designing Your Experiment

This chapter describes the general methods of designing an experiment. The conceptual work that needs to be completed prior to implementing an experiment is reviewed in detail. The process that is described in this chapter can be followed for any experiment. This chapter will also identify design issues that should be considered prior to attempts to compete an experiment for data collection. For users who are new to experimental design, using computers to implement experiments, or to E-Studio itself, we recommend working through this chapter in its entirety prior to beginning with the staged implementation provided in *Chapter 3: Implementing your Experiment in E-Studio*. While more experienced users may choose to skim the contents of this chapter as a reminder of the key design components prior to performing the implementation in Chapter 3.

For the purposes of the next several chapters, the lexical decision experiment will be used as an example. Read the following overview of the experiment to become familiar with the basic experiment. The lexical decision experiment requires the experiment participant to make a decision about a string of letter. In this example, they need to decide if a string of letters is a word or nonsense. First the participant sees a prime presentation (e.g., "word" or "nonword"). This is followed by a the presentation of string of letters which is either an actual word, or simply a string of letters. The participant responds and is given feedback. Several trial run and then the experiment ends with displaying a Goodbye screen to the participant.

Define Your Experiment in Stages

An enormous amount of detail is required to program an experiment, and in order to create a scientifically valid experiment these details must be clearly conceptualized before implementation in E-Studio. Each of the variables, and the experimental procedures must be precisely defined. Additionally, careful consideration must be given to the data collection, and how the data will be analyzed. Once these steps are complete, the experiment should be tested and the data analyzed. It is bad practice to collect data before testing the experiment or visualizing the analysis. Conceptualizing and testing both the experiment and analysis, will save time and result in better research.

It is best to design the experiment in a series of stages, each containing a few steps. Each step describes one aspect of the experiment, and builds on the previous step like a ladder. Taking the time to map out a few trials on paper before programming begins will substantially speed development and reduce errors. The table below contains the recommended stages for developing an experiment.

The Key Stages In Experiment Development
Stage 1: Conceptualize the Core Experimental Procedure
Stage 2: Elaborate the Trial Procedure
Stage 3: Add All Conditions, Set Number of Trials and Sampling
Stage 4: Add Blocks and Block Conditions
Stage 5: Add Practice Block
Stage 6: Special Functions – Setting Timing Modes and Graceful Abort
Stage 7: Testing the Experiment
Stage 8: Running the Experiment
Stage 9: Performing Basic Data Analysis
Stage 10: Archiving Experiments and Results
Stage 11: Research Program Development

2.1 Stage 1: Conceptualize the Core Experimental Procedure

The goal of this stage is to get the basic procedure designed to the point where the experiment contains at least two different instances of the trial procedure. Be clear on the trial procedure and what data are to be collected. There are 3 major steps to Stage 1, as shown below.

Stage 1: Conceptualize the Core Experimental Procedure

1) Preparation Work
    • Provide an operational specification of the base procedure
    • Create a folder for the experiment

2) Conceptualize a Basic Trial
    • Specify the experimental design, independent variables, stimuli, and expected responses
    • Specify the trial procedure
    • Specify the non-default and varying properties of the trial events

3) Data Logging
    • Specify what data will be logged for analysis
    • Verify the core experiment
    • Visualize the data logging of the core experiment

Stage 1, Step 1: Preparation Work

Provide an operational specification of the base procedure

The process of putting your experiment into words forces you to clearly define all aspects of the experiment. In order to design a scientifically sound experiment, precisely define what is being manipulated (independent variables), the expected outcome (dependent variables) and the hypothesis before any programming takes place. Then write a draft of the abstract for the experiment being implemented, particularly detailing the procedure. For the lexical decision experiment this might be:

> *The experiment will measure the time a participant takes to make a lexical decision. The independent variable is whether a letter string is a word or a non-word. The participant will be presented with a fixation (+) displayed in the center of the screen for one second. Then a Probe Display will present a letter string stimulus in the center of the screen for up to 2 seconds. The stimulus display will terminate when the participant responds. Participants are to respond as quickly as possible as to whether the stimulus was a word or a non-word by pressing the "1" or "2" key respectively. The dependent measures are the response (i.e., key pressed), response time, and response accuracy of the Probe Display. The stimuli will be words and non-words, presented in random order in black text on a white background.*

Hypothesis:
Participants will recognize a string of letters that make up a word faster than they recognize a non-word.

Abstract:
The experiment will measure the time a participant takes to make a lexical decision. The independent variable is the string type: word or non-word. The participant will be presented with a fixation (+) displayed in the center of the screen for one second. Then a Probe Display will present a letter string stimulus in the center of the screen for up to 2 seconds. The stimulus display will terminate when the participant responds. Participants are to respond as quickly as possible as to whether the stimulus was a word or a non-word by pressing the "1" or "2" key respectively. The dependent measures are the response (i.e., key pressed), response time, and response accuracy of the Probe Display. The stimuli will be words and non-words, presented in random order in black text on a white background.

Using this rough abstract, you can define the basic procedure for your experiment.

Create a folder for the experiment

It is good practice to organize your design notes and your experiment specification files. Experiment development is highly iterative in nature, and you can quickly develop many different versions of a similar experiment. We recommend developing your experiments in the My Experiments folder that was created in your Documents folder when E-Prime 2.0 was installed. Within that folder organize your notes in folders that are named by the experiment series, e.g., LexicalDecision. We also recommend that as you make changes to your experiment you increase the number each time you save major changes to that version of the experiment, Stage1-LexicalDecision001.es2, Stage2-LexicalDecision001.es2, etc. and that you number each version of the experiment (e.g. LexicalDecision001). Do not minimize the importance of organizing your design work and experiments. A little thought at the start of your experiment development work will make retrieving specific versions of your design notes, and ultimately experiment programs, much easier in the future.

Stage 1, Step 2: Conceptualize a Basic Trial

Specify the experimental design, independent variables, stimuli, and expected responses

In order to specify the design of the experiment you will need to list all of the conditions, stimuli, and expected responses. The abstract of the lexical decision experiment includes the following details about the design: The independent variable is whether a letter string is a word or a non-word… the stimuli will be text strings of words and non-words… Participants are to respond as quickly as possible as to whether the stimulus was a word or a non-word by pressing the "1" or "2" keys respectively. The underlining in the abstract highlights key terms that will directly influence the names or settings in the E-Prime 2.0 experiment specification. This portion of the experiment design can be organized in a table, such as the following:

Condition	Stimulus	Correct Response
Word	cat	1
NonWord	jop	2

In this case, we start with one independent variable (Condition) having two levels or cells (Word, NonWord). For each cell, the table contains a separate line. We need to specify the stimulus that determines what the participant sees in that condition and the correct response to the stimulus, which determines how the response is scored. Later, more independent variables (e.g., word frequency and priming), stimuli, and associated responses can be added.

Specify the trial procedure

The core trial procedure of an experiment is a minimal, repetitive portion in which different conditions are selected, stimuli are presented, and responses are collected. The core trial procedure typically defines the sequence of events the participants experience. It is useful to make a diagram of the sequence of events and connect the design to the events.

For example, in the lexical decision experiment, the core trial procedure was described in the abstract: The participant will be presented with a fixation (+) displayed in the center of the screen for one second. Then a Probe Display will present a letter string stimulus in the center of the screen for up to 2 seconds. The stimulus display will terminate when the participant responds. The lexical decision procedure can be specified in a list of events as follows:

• Select the Stimulus from the list of trial stimuli.
• Present the Trial Procedure, which contains: Fixation, then ProbeDisplay, and collect the response.

Select Stimulus	⇒	Fixation	⇒	Probe Display/ Collect Response

Specify the non-default and varying properties of the trial events

The specific nature of each stimulus can be specialized by setting critical properties of the stimuli. E-Prime 2.0 provides defaults for dozens of properties for each stimulus (e.g., font style, forecolor, background color, location, duration, etc.). The properties that do not have a default value must be set (e.g., text in the fixation display to be a "+"), as must any properties that differ from the default. Also, the properties that vary in the experiment (e.g., based on the condition) must be set. The abstract specified a series of properties that must be set:

The stimuli will be words and non-words, presented in random order in black text on a white background.

The participant will be presented with a fixation (+) displayed in the center of the screen for one second. Then a Probe Display will present a letter string stimulus in the center of the screen for up to 2 seconds. The stimulus display will terminate when the participant responds. Participants are to respond as quickly as possible as to whether the stimulus was a word or a non-word by pressing the "1" or "2" key respectively.

For the lexical decision experiment, the fixed and varying properties can be summarized in the following table:

Object	Fixed Properties	Varying Properties
Fixation	• Present "+" in the center of the screen • Duration = 1 second (default) • Foreground color = black (default) • Background color = white (default)	(none)
Probe	• Display in center of the screen (default) • Duration = 2 seconds • Input keys = "1" and "2" • Terminate display upon response • Foreground color = black (default) • Background = white (default)	• Stimulus (e.g., "cat", "jop", etc.) • Correct response = "1" or "2"

Stage 1, Step 3: Data Logging
Specify what data will be logged for analysis
For a research experiment, the experimenter has to record and analyze data. This requires clarity on what variables will be logged. Be able to answer, "What dependent measures will be recorded in reference to specific objects in the experiment?" The abstract above stated: *The dependent measures are the response, response time, and response accuracy of the Probe Display.* After a trial, expect to record the following information for the Probe Display: What was the response that the participant entered (i.e., 1 for word and 2 for nonword)? What was the response time? What was the accuracy (i.e., 1 for correct and 0 for wrong)?

Verify the core experiment
At this point, the minimal core of the experiment is designed. Once you have this much of the experiment designed, it is a good idea to verify your design. Of course, when you are implementing the experiment in E-Prime 2.0, as in *Chapter 2: Designing Your Experiment*, you would run the experiment at this point to see if it works as intended. Once you have the experiment programmed, you would simply run the experiment to see if it works as intended. Now we will walk through the proposed layout to make certain it is logical.

Recall the experimental procedure abstract:
The participant will be presented with a fixation (+) displayed in the center of the screen for one second. Then a Probe Display will present a letter string stimulus in the center of the screen for up to 2 seconds.

It is helpful to draw out the displays the participant would see. In this case, expect to see two trials, with the first presenting a fixation "+" for one second and then the word "cat" for 2 seconds, or until a response is entered. Then the "+" is presented again, and the second stimulus "jop" is displayed until a response. For example:
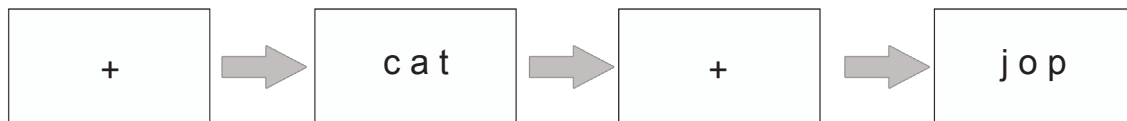
Prior to running the experiment you should also be able to envision the ways in which the trial event ends. In this case, when the participant presses a valid response key, the text string will disappear, and the experiment will go to the next trial.

Visualize the data logging of the core experiment
An experiment is only useful if the critical data are recorded for later analysis. Be clear about what you expect to see as input data for the analysis. From the abstract:

The independent variable is whether a letter string is a word or non-word… The dependent measures are the response (i.e., key pressed), response time, and response accuracy to the Probe Display.

In this case, for each trial you would expect to see the condition and unique stimulus selected, plus the dependent measures (participant response, response time, and accuracy of the response) in the data file. Prior to actually collecting any data, you should be able to visualize the information that is being logged and how you will analyze the data to investigate the experimental questions of interest. For example, the trials described above might appear like the following:

Independent Variables			Dependent Variables		
Condition	Stimulus	Correct Response	Probe Response	Probe RT	Probe Accuracy
Word	cat	1	1	586	1
NonWord	jop	2	1	1224	0

For trial 1 of this run of the experiment, the independent variables were Condition = Word, Stimulus = cat, and Correct Response = 1. The dependent variables were the response key (Probe Response = 1), response time (Probe RT = 586), and response accuracy (Probe Accuracy = 1, indicating a correct answer). For trial 2, the Condition = NonWord, Stimulus = jop, Correct Response=2, Probe Response =1 (which mismatches the expected response), Probe Reaction Time =1224, and Probe Accuracy=0 (indicating an error).

Stage 1 Summary: In Stage 1, we conceptualized and planned how to verify the data logging of the core experiment, limiting the design to two cells.

Exercise: For experiment development, draw the figures expected to occur, as above, for your experiment. In particular, specify the following:

• Abstract of the core experiment
• Folder name for the design notes and naming convention for the experiment files
• List of the conditions, stimuli, and correct responses
• Core experimental procedure including when stimuli are selected
• Trial procedure, including a sequence of displays and participant responses
• Non-default properties of the trial events, including the fixed and varying text displays and the properties for how to display the stimulus, its duration and termination condition, and the correct response
• Expected participant displays (i.e., draw them), when responses are to be made, and if they clear the screen
• Independent and dependent variables that are logged for the experiment

2.2 Stage 2: Elaborate the Trial Procedure

This section builds upon the basic design that was laid out in Stage 1. Following the guideline of iterative design and development, Stage 1 defined the most simple trial procedure: present a probe stimulus and collect a response. Now we are ready to elaborate the trial procedure by adding a Prime Display. In a priming version of a lexical decision task, the participant is presented with a text string prior to seeing and responding to the Probe Display. The question of interest is whether the nature of the Prime Display influences the response to the probe word.

As always, start with the abstract. The original abstract is copied below, with the new prime manipulation information underlined:

The experiment will measure the time to make a lexical decision after first seeing a priming word. The independent variables are whether a letter string is a word or a non-word, and whether the Prime Display is the string "Word" or "Nonword". The participant will be presented with a fixation (+) displayed in the center of the screen for one second. Next, a priming word will be displayed for one second. Then a Probe Display will present a letter string stimulus in the center of the screen for up to 2 seconds. The stimulus display will terminate when the participant responds. Participants are to respond as quickly as possible as to whether the stimulus was a word or a non-word by pressing the "1" or "2" key respectively. The dependent measures are the response (i.e., key pressed), response time, and response accuracy of the Probe Display. The stimuli will be words and non-words, presented in random order in black text on a white background.

The goal of Stage 2 is to elaborate the basic Fixation-Probe trial procedure by adding a priming manipulation. A display will be added at the beginning of the trial procedure to permit the participant to pace the trials and instructions will be added to the Fixation display to remind the participant of the response keys. Also, a Prime Display will be added prior to the Probe Display, and Feedback will be added to the end of the trial procedure.

There are two major steps to Elaborating the Trial procedure, as shown below:

Stage 2: Elaborate the Trial Procedure

1) Add the Missing Delays and add Instructions to Existing Displays
   • Add Get Ready Display
   • Add Fixation and Probe Display Instructions
   • Add Prime Display
   • Add Feedback Display

2) Verify the Get Ready, Prime, and Feedback Objects

Stage 2, Step 1: Add the Missing Delays and add Instructions to Existing Displays

In Stage 2, three new displays are added to the trial sequence, and instructions are added to the existing prime and Probe Displays. A trial now looks like the following:

Select Stimulus → Get Ready → Fixation → Prime Display

Probe Display → Collect Response → Trial Feedback
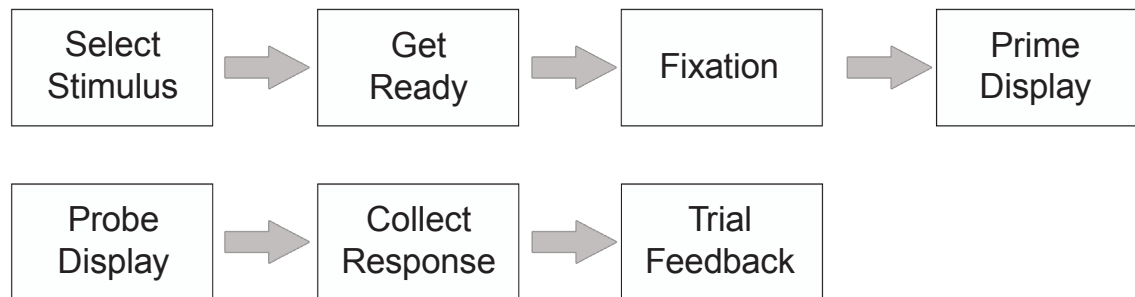
Get Ready Display

The Get Ready Display will enable the participant to begin the trial when he or she is ready; the "Get Ready" message will remain on the screen until the participant presses the designated key.

Fixation and Probe Display Instructions

Instructions will be added to the Fixation display to remind the participant of which keys they are to press for which response:

Press the 1 key for a word
Press the 2 key for a nonword

These instructions will remain on the screen for the Prime and Probe Displays. The instructions should be positioned on the screen so that they do not interfere with the presentation of the Probe Display.

Prime Display

The Prime Display is the string "word" or "nonword"[1]. For data analysis purposes, the string that was presented on the Prime Display will be logged.

Feedback Display

The Feedback Display tells the participant whether or not their response to the trial was correct. It is typical to include feedback on experiments measuring reaction time or response accuracy as a way to help keep the participant motivated and focused on the task. It is typical to display feedback as follows:
• A message indicating whether the participant's response was correct or incorrect
• Reaction time to the stimulus
• Cumulative response accuracy

---

[1] There are numerous variations of the Lexical Decision task. In many of them, the Prime manipulation is a more complicated one, whereby the Prime is a word that is related to the probe, a word that is unrelated to the probe, or a nonword. For the purposes of this chapter, the Prime manipulation is a more basic one, and serves to illustrate the general trial procedure for including a Prime Display prior to the probe.

Stage 2, Step 2: Verify the Get Ready, Prime and Feedback Objects

At this point, the three new displays and the two modified displays have been designed. Again, you should be able to envision and draw out the displays that the participant would see. In the case of the Stage 2 version of this experiment, expect to see displays like the following:
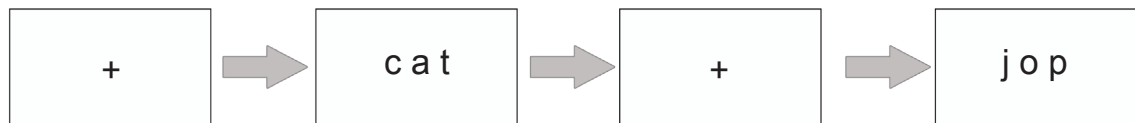
2.3 Stage 3: Add All Conditions, Set Number of Trials and Sampling

The goal of Stage 3 is to fully design the set of stimuli to be presented to the participant. This involves identifying the complete stimulus set and the manner in which the stimuli are sampled (sampling method). For this particular example, expanding the stimulus set also involves identifying a new independent variable. The four steps for Stage 3 are identified in the table below:

Stage 3: Add All Conditions, Set Number of Trials and Sampling
1) Add All Conditions
2) Set the Weights
3) Set the Sampling Mode and Exit Condition
4) Verify

Stage 3, Step 1: Add All Conditions

In Stage 2, we added a Prime Display to the trial procedure. As is typical in priming studies, a priming stimulus is presented to the participant prior to the presentation of the probe stimulus. For the purposes of this example, we are keeping the Prime manipulation simple: the Prime stimulus is either the string "word" or "nonword". More advanced priming versions of the lexical decision task might manipulate the nature of the Prime more extensively, for example by presenting 3 different prime types: words related to the probe, words unrelated to the probe, and nonword strings. Nevertheless, the Prime manipulation implemented in Stages 2 and 3 here illustrates the basics of how to include a priming display in the trial procedure, along with how to log the relevant independent variable information.

The new priming manipulation requires a change in the terminology with which we describe the trial stimuli. Back in Stage 1, we identified a single independent variable, Condition, which was used to identify the sole condition of the experiment: whether the Probe Display is a word or nonword. Now that we no longer have a single manipulation, using a single independent variable to define a trial no longer makes sense.

The new design is a 2 by 2 design which results in four types of trials. The first factor is the Prime type (the string "word" or "nonword"), and the second factor is the probe type (a word or nonword). The independent variable that we identified as "Condition" in Stage 1 now identifies the probe type, and we need a new independent variable that identifies the Prime type. (In this simplified design, the Prime type also happens to be the actual string that is presented to the participant on the Prime Display.) The combination of four trial types is shown on the following page:

After identifying all of the PrimeTypes and ProbeTypes, we need to identify all possible independent variables. In other words, all possible combinations of the PrimeType and ProbeType that can constitute the independent variables. We will use two words and two nonwords to illustrate:

Independent Variables	
PrimeType	**ProbeType**
Word	word
NonWord	word
Word	non-word
NonWord	non-word

When you have a small number of independent variables/stimulus combinations they can be listed easily in a table as shown above. If instead there are a large number of independent variables/stimulus combinations, you may not want to create such a table. For example, if your study consisted of 50 unique word strings and 50 unique nonword strings, a table of all possible combinations of prime and probe would contain 200 rows!

Trial		
PrimeType	**ProbeType**	**Stimulus**
Word	word	cat
Word	word	dog
NonWord	word	cat
NonWord	word	dog
Word	non word	jop
Word	non word	fuame
NonWord	non word	jop
NonWord	non word	fuame

Instead, if you are fully crossing all combinations of independent variables and stimuli, you may choose to identify the unique stimuli and then describe the method in which the stimuli are to be sampled and paired with the independent variables. Such a description might look like the following:

PrimeType - The list of primes to be presented is:
• Word
• NonWord

Stimulus - The list of stimuli to be presented is:
• cat
• dog
• jop
• fuame

Sampling – Each ProbeType is paired once with each Stimulus.

Stage 3, Step 2: Set the Weights

In some experiments, you may want to explicitly set the relative frequency of trial types. In the table of conditions that was shown in *2.3 Stage 3: Add All Conditions, Set Number of Trials and Sampling (Page 18)*, above there were an equal number of word and non-word trials. However, we are going to change that specification, and present each of the non-word exemplars twice; this will result in the participant seeing twice as many non-word trials (8) as word trials (4). *Stage 3, Step 2: Set the weights (Page 54)*, shows how to change the relative frequency of trial types in E-Studio.

Stage 3, Step 3: Set the Sampling Mode and Exit Condition

The sampling mode allows the altering of the order in which levels or conditions are run. Sampling modes include sequential, random (without replacement), random (with replacement), counterbalance, offset, and permutation. Each of these modes is described below. E-Prime 2.0 supports all of these sampling modes. *Chapter 3: Implementing your Experiment in E-Studio (Page 28)*, and *Stage 3, Step 3: Set the Sampling Mode and Exit Condition (Page 54)*, describe how to specify each mode.

The simplest sampling mode is sequential presentation of items. When sampling from your stimulus list this way, all participants see all of the stimuli in the same order as you enter them. While sampling a list of stimuli in sequential order can be very useful when debugging and testing an experiment, it is rarely used when collecting experimental data.

Random sampling, without replacement, is a frequently used sampling technique, and is what we will use for our lexical decision experiment. Stimuli are sampled randomly without replacement until the specified number of stimuli have been presented.

For our example, we have 8 nonword and 4 word stimuli, for a total of 12 trials. After the last stimulus has been selected from the list, then all stimuli are re-randomized and become available for sampling again. If you set the number of trials equal to the total number of exemplars, then sampling randomly without replacement ensures that all participants see all exemplars within a single run of the experiment, and all participants see a different randomized order. For our lexical decision experiment, we will present a total of 24 trials, so the full list of exemplars will be sampled randomly without replacement until the 12 exemplars are sampled; then the exemplars will be returned to the pool of available samples, and will be sampled again randomly without replacement for another 12 trials.

Stimuli can also be sampled randomly, but with replacement. In such scenarios, each stimulus has an equal chance of being selected on a given trial. However, participants are not assured of seeing all exemplars, because after each trial the chosen exemplar is returned to the pool of eligible exemplars.

Another useful sampling technique is counterbalancing. When you counterbalance conditions, one exemplar is picked based on a selector variable, such as subject number or session number. For example, a design running six conditions with counterbalance by participant would result in only the third condition being presented to subject number 3. This might be used for a Latin Square design between participants.

In addition to counterbalancing, there are two other sampling methods that utilize a selector variable. Both of these methods are supported by E-Prime 2.0. When sampling with the offset method, the first exemplar sampled from the list of stimuli is determined by the offset factor, and then subsequent exemplars are sampled in fixed order from that location for the specified number of samples, and wrapping around to the beginning of the list when the end of the list is reached. For example, a design running six blocks of trials with offset by participant would result in the first sampled item being determined by the subject number. For subject number 3, the third item would be selected first; the list selection would then continue in fixed order (i.e., the fourth item would be selected next, followed by the fifth, etc.) until the end of the list is reached. The sampling would then wrap to continue with the first item in the list, then the second item, and conclude (because all of the items in the list have been sampled). This might be used for a Latin Square design within participant assignment of conditions.

Lastly, with the permutation sampling method, all possible combinations of conditions are created. Then, from the pool of possible combinations, one combination is chosen based on the value of the selector variable. For example, a design running three blocks of trials (A, B, and C) with Permutation by Participant would result in the generation of six possible combinations of conditions (i.e., ABC, ACB, BCA, BAC, CAB, CBA). From those possible combinations, subject number 3 would receive the third combination (i.e., BCA). Care should be taken when sampling with permutations, since the generation of all possible conditions increases factorially. That is, a large number of conditions will result in a tremendously large number of combinations of those conditions (e.g., 5 conditions result in 120 combinations, 6 conditions result in 720 combinations, etc.). The permutation option is best used with a small number of conditions.

The table below illustrates how a List with three elements (values A, B, and C) might be sampled using the different methods described above. The last three methods (counterbalance, offset, and permutation) assume that subject number is used as the selector variable. For example, considering the counterbalance method, participants 1, 4, and 7 would sample from row 1, participants 2, 5, and 8 would sample from row 2, and participants 3, 6, and 9 would sample from row three. In contrast, the permutation condition with 3 conditions has 6 possible sequences. Participants 1, 7, and 13 would have the first order, participants 2, 8, and 14 would have the second, etc.

Selection						
Sample	Sequential	Random (without replacement)	Random with replacement	Counter-balance	Offset	Permutation
1	A	B	B	Sub 1 = A	Sub 1 = ABC	Sub 1 = ABC
2	B	C	C	Sub 2 = B	Sub 2 = BCA	Sub 2 = ACB
3	C	A	B	Sub 3 = C	Sub 3 = CAB	Sub 3 = BCA
4	A	C	C	Sub 4 = A	Sub 4 = ABC	Sub 4 = BAC
5	B	B	B	Sub 5 = B	Sub 5 = BCA	Sub 5 = CAB
6	C	A	A	Sub 6 = C	Sub 6 = CAB	Sub 6 = CBA

Stage 3, Step 4: Verify

As always, at the end of an incremental design stage, you should be able to envision the displays that the participant would see and anticipate how to analyze the data. The trial procedure was not altered in Stage 3. However, we added more stimuli and changed our sampling, so that a total of 24 trials are presented, in random order with non-words occurring twice as frequently as words.

The data analysis has changed, with the changes in the independent variables (renaming "Condition" to the more useful "ProbeType" and adding "PrimeType"). The table below shows the expected data file structure, with simulated values entered for the dependent variables:

Independent Variables				Dependent Variables		
PrimeType	ProbeType	Stimulus	Correct Response	Probe Response	Probe RT	Probe Accuracy
NonWord	word	dog	1	1	768	1
Word	non-word	fuame	2	1	942	0
NonWord	non-word	jop	2	2	1125	1
Nonword	non-word	fuame	2	2	899	1
Word	word	dog	1	1	724	1
Word	non-word	jop	2	2	1043	1
NonWord	non-word	jop	2	1	942	0
NonWord	non-word	fuame	2	2	756	1
NonWord	word	cat	1	2	689	0
Word	non-word	jop	2	1	854	0
Word	non-word	fuame	2	2	781	1
Word	word	cat	1	1	815	1
NonWord	non-word	jop	2	2	1203	1
NonWord	non-word	fuame	2	2	1578	1
Word	word	dog	1	1	924	1
Word	non-word	fuame	2	1	891	0
Word	non-word	jop	2	2	863	1
NonWord	word	cat	1	1	699	1
Word	non-word	jop	2	2	907	1

2.4 Stage 4: Add Blocks and Block Conditions

The goal of this stage is to add a new block manipulation which varies the duration of the Prime Display across blocks. In this example, we will run one block where the Prime is displayed for 500 ms and a second block where the Prime duration is 100 ms. We will also add a new display to the start and end of the experiment, to first welcome the participant and then thank them for their participation.

| Stage 4: Add Blocks and Block Conditions |
|---|
| 1) Verify setup of the Block Manipulation
  • Add the block procedure
  • Move the trial stimuli from the Session level to the Block level |
| 2) Add Block Instructions |
| 3) Add Introduction and Goodbye displays |
| 4) Special Notes: Multiple Methods to Divide a Design Between Levels |

Stage 4, Step 1: Verify setup of the Block Manipulation
Add the block procedure
The experiment will now present two blocks of trials. The new block independent variable named "PrimeDuration" will log the value of the Prime duration for each block, either 500 or 100 ms. The Prime duration will be selected randomly without replacement for each block of trials.

Move the trial stimuli from the session level to the block level
The outline of our experiment needs to change with the addition of the block manipulation. We want to sample all of the trial stimuli randomly, without replacement, for each block of the experiment. The order of events within an individual trial is unchanged from what we diagrammed in *Stage 2, Step 1: Add the Missing Delays and add Instructions to Existing Displays (Page 17)*. But at the session level, the order of events has now expanded to the following:
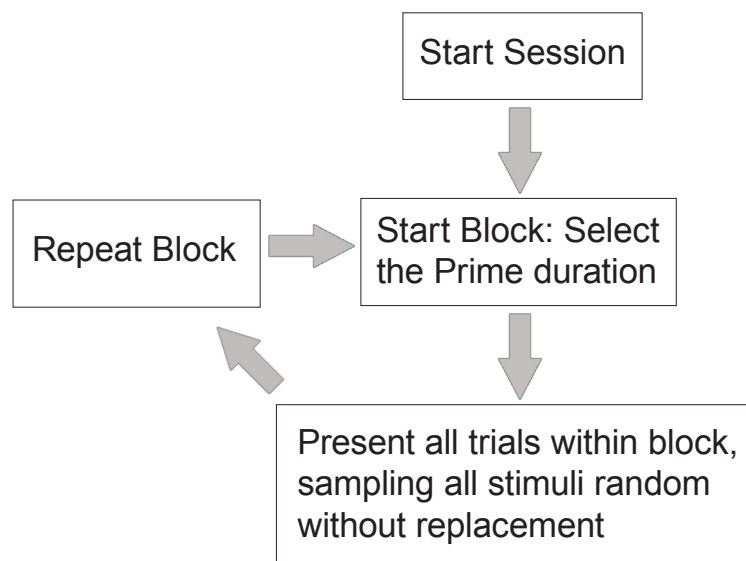
Stage 4, Step 2: Add Block Instructions

The block procedure will be modified to first present a block introduction screen to the participant which will present the following information to the participant:

For the next block, the duration of the Prime will be <duration of prime> milliseconds long.

Press the spacebar to begin.

Stage 4, Step 3: Add Introduction and Goodbye displays

It is important to both instruct the participant as to the task at the beginning of the experiment and to inform the participant when the experiment is completed. The experiment will now include the following welcome message:

Welcome to the experiment.
Press the spacebar to continue.

When the experiment has completed, the participant will now see the following message:
Thank you for participating.
Please leave the room quietly.

It is important to instruct the participant as to what they should do at the end of the experiment. If your laboratory has multiple participant running stations, it is good practice to remind the participant to not disturb the other participants as they leave the lab.

Stage 4, Step 4: Special Notes: Multiple Methods to Divide a Design Between Levels

There are a large variety of ways in which experimental design components are varied between blocks within a single session. The current example varies one of the display characteristics, the duration of the Prime. There are other block manipulations that are commonly used and which are readily supported in E-Prime 2.0. Two such block manipulations are mentioned briefly here, and implemented in *Chapter 3: Implementing your Experiment in E-Studio*.

Select exemplars from different lists
You may find it useful to select stimuli from different lists of exemplars per block. For example, you could run one block of trials in which all of the words are from the Animal category, and a second block in which all of the words are from the Fruits category. This type of design utilizes a block attribute that defines the master list of exemplars to be used for the current block of trials.

Run different trial procedures
You may want to run different trial procedures in different blocks. For example, in a priming study you may want to run some blocks where the Prime Display is masked prior to the Probe Display and other blocks where the display is not masked. This type of manipulation is readily supported in E-Studio, and is described in Chapter 3.

2.5 Stage 5: Add Practice Block

The goal of Stage 5 is to add a block of practice trials prior to the experimental trials. The practice block will be run repeatedly until the desired block accuracy of 80% is obtained.

There are four steps involved in adding a practice block, as shown in the following table:

Stage 5: Add Practice Block
1) Define Practice Block
2) Add the Practice Block Variable
3) Define the Practice Performance Criterion
4) Verify

Stage 5, Step 1: Define Practice Block

From the participant's perspective, the practice block is just like the "real" experimental block; the same events happen on each trial and the same exemplars are sampled. The only differences between the practice and experimental trials are:

• The practice block of trials needs to be identifiable as such, so that the practice data is not included in the data analysis. This requires a new block independent variable, which is set up in Step 2 below.
• The participant's overall accuracy at the conclusion of the practice trials needs to be checked to see if the participant has met the performance criterion.
• If the participant has not met the performance criterion, then the practice block is presented again. If the participant has met the performance criterion, then the experimental trials are presented.

Stage 5, Step 2: Add the Practice Block Variable

In order to easily analyze the practice data separately from the main data, add a variable that codes the information as practice or experimental trials. The new variable will be named "PracticeMode". There are many different ways that practice trials can be identified; in this case, we will identify all of the trials in the practice block as "Practice" and all of the trials in the experimental block as "Real":

New Independent Variable	
Block condition	**PracticeMode**
Practice trials	Practice
Experimental trials	Real

Stage 5, Step 3: Define the Practice Performance Criterion

It is good practice to check the participant's performance after completing some practice trials to ensure that the participant has understood the task and is performing at a level that will enable meaningful data analysis (e.g. reaction time analysis is filtered on correct responses only, so reasonable numbers of trials need to be included for each participant). For the lexical decision task, we will require participants to perform at greater than 80%; otherwise, the practice block will be presented again.

A corresponding change is to add instructions to explain the altered experiment flow. Participants need to be informed of their performance level at the conclusion of the practice block and if the practice blocks are to be repeated.

Stage 5, Step 4: Verify

As always, at the end of an incremental design stage, you should be able to envision the displays that the participant would see and anticipate how to analyze the data. The trial procedure was not altered in Stage 5. However, the block procedure was expanded to present a practice block, and to inform the participant about their practice block performance.

The data analysis will need to change as well. The addition of a new block independent variable named "PracticeMode" results in a new column of data added to the expected data file structure. The new variable will need to be used in the data analysis to exclude all practice trials; for details see *3.5 Stage 5: Add Practice Block (Page 66)*.

2.6 Stage 6: Special Functions – Setting Timing Modes, and Graceful Abort

The design that has been worked out in Stage 1 – 5 above, and correspondingly implemented in *Chapter 3: Implementing your Experiment in E-Studio (Page 28)*, results in a functioning experiment. However, there are two additional considerations that should become a standard part of any experimental design work for a computerized experiment, as shown in the table below:

Stage 6: Special Functions – Setting Timing Modes, and Graceful Abort
1) Checking the Timing Modes
2) Providing Early Graceful Abort of a Block

Stage 6, Step 1: Checking the Timing Modes

Chapter 4: Critical Timing details methods to specify and check timing. As that chapter explains in detail, aspects of both the computer hardware and software influence the way in which computerized displays must be programmed to maintain the greatest timing accuracy possible. The accuracy of the display durations are of particular concern with brief display durations, e.g. less than 50 ms.

With some experimental designs, the most critical aspect of the display durations involves the length of time that a stimulus is displayed. We refer to this as event mode. In other designs, maintaining accurate inter-stimulus intervals is of primary concern, particularly if the experiment involves synchronization with external biological monitoring equipment such as evoked potentials. We refer to these types of designs as cumulative mode.

For this experiment, the Prime duration is either 50 or 100 ms, and there are no critical inter-stimulus intervals that need to be maintained. *3.6 Stage 6: Special Functions – Setting Timing Modes, and Graceful Abort*, describes how to properly utilize the E-Prime 2.0 critical timing features to properly specify these experimental design components. *Chapter 4: Critical Timing* of this document provides important, detailed information about both the mechanics that affect critical timing and accurate data collection and the advanced tools provided by E-Studio to both specify the most accurate timing possible and also collect data to confirm that timing.

Stage 6, Step 2: Providing Early Graceful Abort of a Block

Ideally, when participants run through a computerized experiment, they run through the complete experiment, experiencing all of the trials that have been defined and leaving the experimenter with a complete data file. In reality, however, numerous circumstances require that the participant is unable or unwilling to complete the experiment. There needs to be some mechanism by which the experiment can be terminated early. E-Prime 2.0 provides three different ways in which an E-Prime 2.0 experiment can be aborted while running; these are described in *Stage 6, Step 2: Providing early graceful abort of an experiment (Page 72)*.

2.7 Stage 7: Testing the Experiment

Once the experiment is "running" without error, a thorough process of testing must take place to reduce the threat of problems during data collection that may lead to incomplete or lost data.

The need to thoroughly test the experiment prior to beginning data collection cannot be stressed enough. Testing is critical to ensure the following:

• Responses to the stimulus display are scored correctly
• Trial and block data is logged as expected
• Accurate timing is maintained for all critical timing displays and response latency collection
• Participants understand the instructions and task
• E-Prime 2.0 provides multiple tools to aid in the testing of the experiment. See *3.7 Stage 7: Testing the Experiment (Page 73)*, for important details.

2.8 Stage 8: Running the Experiment

Once experiment development is completed and you have thoroughly tested the experiment and examined the data file to determine that the necessary information is logging appropriately, data collection is ready to begin. *3.8 Stage 8: Running the Experiment (Page 74)* describes how to run the completed experiment and collect data, including how to collect data for the same experiment when running across multiple data collection PCs.

2.9 Stage 9: Perform Basic Data Analysis

E-Prime 2.0 collects only single participant data files. For group analysis, the single participant data files must be merged into a master file using the E-Merge application. *3.9 Stage 9: Basic Data Analysis (Page 76)* provides an overview of how to combine individual participant data files into a single merged data file that can be analyzed with E-DataAid and/or exported for analysis in another software package. Detailed information about data analysis is provided in *Chapter 6: Data Handling (Page 158)*.

2.10 Stage 10: Archiving Experiments and Results

For the purposes of replication, collaboration, and verification, it is necessary to keep thorough records of experiment files and methods. *3.10 Stage 10: Archiving Experiments and Results (Page 83)*. Results provides guidelines for archiving your experiments and their results.

2.11 Stage 11: Research Program Development

3.11 Stage 11: Research Program Development (Page 84), provides important guidelines and reminders regarding how to develop experiment series and share them with your colleagues.

Chapter 3: Implementing your Experiment in E-Studio

This chapter describes the general methods of implementing an experiment in E-Prime 2.0. This chapter provides a concrete guide to working in E-Studio that builds on the conceptual work that is laid out in *Chapter 2: Designing Your Experiment*. Whereas Chapter 2 maps out the "what to do", this chapter maps out the "how to do it". This chapter can be used in many ways, depending on your needs and skill level. If you are new to using computers to implement experiments, you may want to work through this chapter in tandem with Chapter 2. If you are more experienced with experimental design but new to E-Studio, you may wish to skim or skip Chapter 2 and begin here. Or, if you are experienced with experimental design, data analysis, and E-Studio, but haven't used E-Prime 2.0 in awhile, you may use this chapter as a quick reference. This chapter repeats the critical design decisions that are made in Chapter 2, and illustrates how the design maps directly to the E-Studio interface.

Regardless of your background, this chapter is appropriate to read before implementing the first experiment from scratch. This chapter assumes the user has worked through the *E-Prime Getting Started Guide*. The user should, by now, have a basic understanding of each application within E-Prime 2.0, and general knowledge concerning working in a Windows application.

Chapters 2 and 3 use as an example a lexical decision experiment. In this experiment, the experimenter might present text strings that are either words or non-words, and record the reaction time for the participant to categorize the stimulus as a word or a non-word. The question might be whether it is faster to recognize a string of letters as a word or a non-word. This chapter illustrates how to implement a lexical decision experiment in E-Studio, collect data and analyze the data. Although the lexical decision paradigm is an example in this chapter, the 11 stage model for designing and implementing experiments which is followed here can be followed for any experiment. Further, this chapter shows how to define key components of an experiment, such as how to specify the trial events, define the fixed and varying attributes of the trial display, sample trial and block stimuli, and log data. More advanced features of E-Studio are described in *Chapter 4: Critical Timing*, and Appendices A, C, and D.

Before you begin with E-Studio

Regardless of your particular background and experience, the point that was stressed in Chapter 2 bears repeating: To program a good experiment, clearly conceptualize the experiment before implementing it in E-Studio. Do not begin to implement any experiment in E-Studio without having first mapped out the critical design and data analysis issues that are highlighted in Chapter 2.

The key stages in experiment development

The 11 stages that were identified in Chapter 2 are repeated here:

The Key Stages In Experiment Development
Stage 1: Conceptualize the Core Experimental Procedure
Stage 2: Elaborate the Trial Procedure
Stage 3: Add All Conditions, Set Number of Trials and Sampling
Stage 4: Add Blocks and Block Conditions
Stage 5: Add Practice Block
Stage 6: Special Functions – Setting Timing Modes and Graceful Abort
Stage 7: Testing the Experiment
Stage 8: Running the Experiment
Stage 9: Performing Basic Data Analysis
Stage 10: Archiving Experiments and Results
Stage 11: Research Program Development

We recommend an iterative approach to experiment development, as is done in the remaining steps below. It is best to design a stage, then implement and test that stage before implementing the next stage.

3.1 Stage 1: Conceptualize the Core Experimental Procedure

In this section we will go over the specifics of performing each of the Stage 1 operations. To perform the Stage 1 operations, we recommend reading through each step below and actually implementing the specified feature of the lexical decision experiment on a computer using E-Prime 2.0. A completed version of the experiment at each stage described in this chapter is included with the installation (default installation is …My Experiments\Tutorials\Using E-Studio Stages). However, rather than going directly to the completed version, we recommend going through each step sequentially, reading how to do the step, performing the step in E-Prime 2.0, and verifying that the displays look as intended. Throughout this chapter, we walk through the full E-Prime 2.0 graphical displays to show where the information is located and how to perform each stage.

The goal of this stage is to get the basic procedure implemented to the point where the experiment presents at least two different instances of the trial procedure. We present the graphical objects from E-Prime 2.0 to help associate the experimental specification with the visual interface in E-Prime 2.0.

Stage 1: Conceptualize the Core Experimental Procedure

1) Preparation Work
   • Provide an operational specification of the base procedure
   • Create a folder for the experiment and load E-Studio

2) Create a Basic Trial
   • Specify the experimental design, independent variables, stimuli, and expected responses
   • Add the attributes for the design
   • Add the needed rows and fill in the attributes for the experiment
   • Specify the core experimental procedure
   • Put the trial events on the TrialProc timeline and rename the objects
   • Specify the stimuli that the participant will see
   • Set the non-default and varying properties of the Trial events

3) Data Logging
   • Specify what data will be logged for analysis
   • Logging Property Page: Set specific logging as needed
   • Run and verify the core experimental procedure
   • Run trials responding correctly, incorrectly and not responding
   • Verify expected displays
   • Verify the data logging of the core experiment
   • Verify the number of rows are the expected number of trials
   • Determine List attributes and numbers of trial conditions occurred as expected
   • Find the dependent variables and verify they are set reasonably
   • Check timing duration accuracy on fixed length displays
   • Verify the data logging of the core experiment

Stage 1, Step 1: Preparation Work

Provide an operational specification of the base procedure

In this step, you should operationally identify the experiment that you are trying to develop. This was done in *Chapter 2: Designing Your Experiment*, *2.1 Stage 1: Conceptualize the Core Experimental Procedure (Page 12)*, and amounts to writing the abstract for the experimental procedure. Below is the abstract again, indicating those aspects of the experimental specification that set specific properties for the specification of the experiment using E-Prime 2.0:

The experiment will measure the time to make a lexical decision. The independent variable is whether a letter string is a word or non-word. The stimuli will be text strings of words and non-words, presented in random order in black text on a white background. The participant will be presented with a fixation (+) displayed in the center of the screen for 1 second. Then a Probe Display will present a letter string

stimulus in the center of the screen for up to 2 seconds. The stimulus display will terminate when the participant responds. Participants are to respond as quickly as possible as to whether the stimulus was a word or a non-word by pressing the "1" or "2" key respectively. The dependent measures are the response (i.e., key pressed), response time, and response accuracy of the Probe Display.

Create a folder for the experiment and load E-Studio

A place to develop the experiment on the computer is now necessary. This involves creating a folder for the experiment. We recommend developing the experiments within in the "My Experiments" folder, which E-Prime 2.0 creates in your personal documents folder (see *1.7 Useful Information (Page 9)* for details on finding the "My Experiments" folder). We suggest that the experiment be placed in a folder named for the experimental series (e.g., ...My Experiments\LexicalDecision) and that the experiment name include a number for the version of the experiment (e.g., LexicalDecision001).

For this experiment, create a "Lexical Decision" folder under the C:\...\My Experiments folder on the computer in which the experiment is being implemented. To create the new folder, select the "My Experiments" folder, right-click, and then select "New - > Folder ".
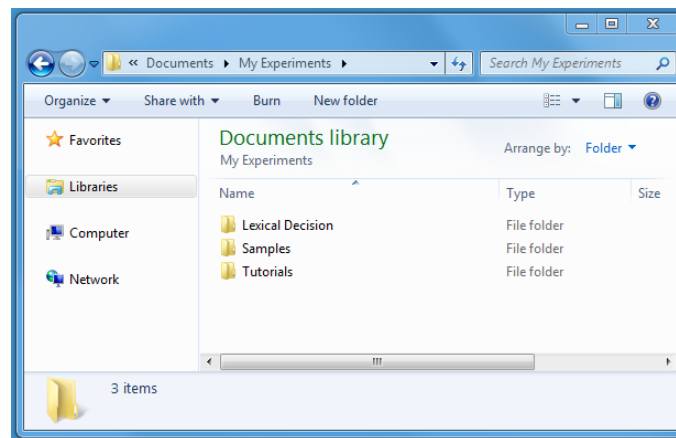
E-Studio is the application in E-Prime 2.0 that is used to design an experiment. The *E-Prime Getting Started Guide* shows how to load E-Studio and open a file. Launch E-Studio, and see a screen such as the following:
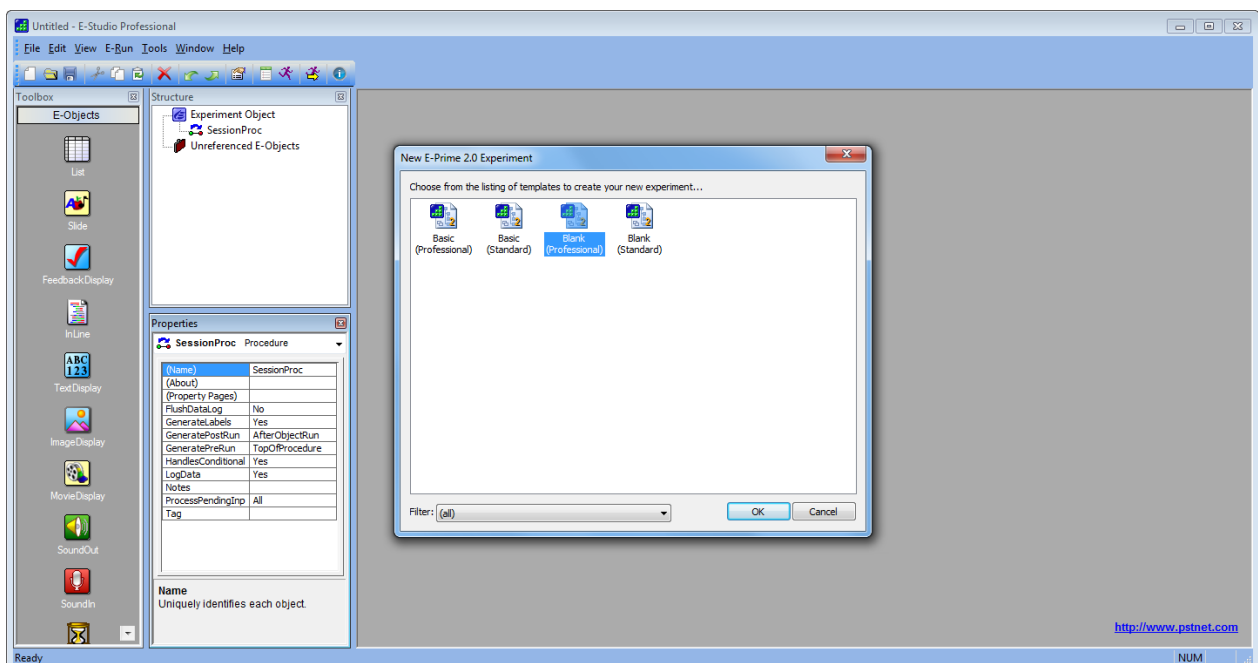
For this example, open a Blank (Professional) (or Blank (Standard) if using E-Prime 2.0 Standard) experiment and use the File menu, Save As option to save the experiment as LexicalDecision001 in the …\My Experiments\Lexical Decision folder. Most experiments go through many variations. We recommend putting a number on the end of the name to document the sequential history of the experiment.

Stage 1, Step 2: Conceptualize a Basic Trial

Specify the experimental design, independent variables, stimuli, and expected responses
In E-Prime 2.0, the design is specified using List Objects. Drag a List object from the Toolbox to the SessionProc timeline. Rename the List1 object to DesignList by clicking on List1, pressing the F2 key (to enter Edit mode) and typing in a new name for the object. The Structure window appears as the following:
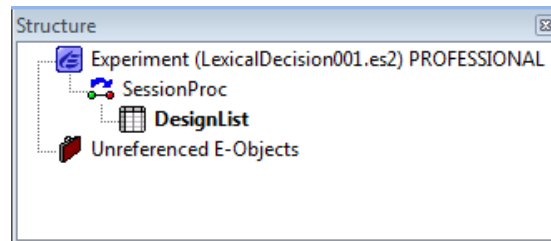
Add the attributes for the design
Recall from the abstract that we wanted to implement the following condition table:

Condition	Stimulus	Correct Response
Word	cat	1
NonWord	jop	2
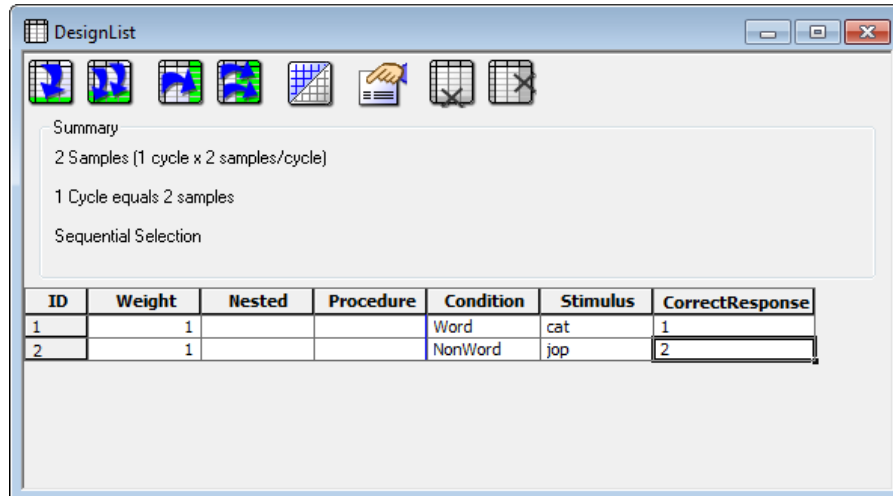
To do so, first open the DesignList object (double click on it in the Structure window). Then, click the Add Multiple Attributes button. When prompted, specify that there are 3 attributes to add. Double click each column header to change the attribute names for Attribute1, Attribute2, and Attribute3 to Condition, Stimulus and CorrectResponse respectively. As you edit each attribute, accept the default value of "?".

⚠ **NOTE:** *No spaces are permitted in the attribute name.*
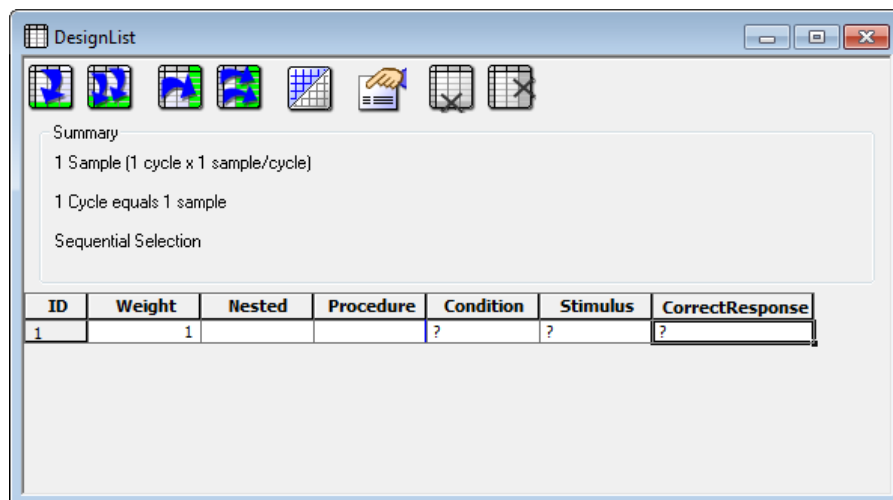
## *Add the needed rows and fill in the attributes for the experiment*

At this early stage in the implementation, the goal is to implement the simplest version that illustrates the varying aspects of the experiment. In the case of the lexical decision task, this requires having 2 rows, one for the Word condition and the other for the Non-Word condition. Click the Add Level tool button to add a second row to the DesignList. Then fill in the values for the levels of the attributes. Enter the values of the Condition, Stimulus, and CorrectResponse attributes for two trials as shown in the next figure.

ID	Weight	Nested	Procedure	Condition	Stimulus	CorrectResponse
1	1			Word	cat	1
2	1			NonWord	jop	2

Specify the core experimental procedure
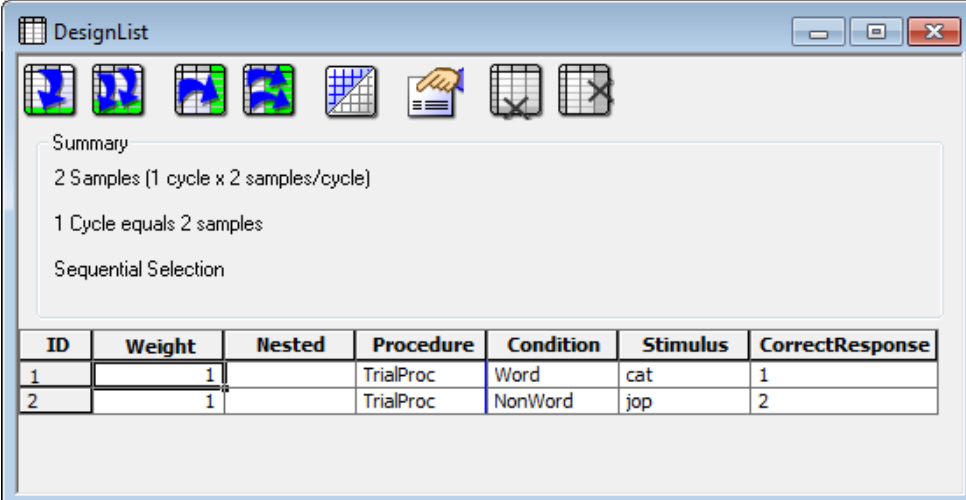
The core experimental procedure is a minimal, repetitive portion of an experiment in which different conditions are selected, stimuli are presented, and the participant responds. The core procedure typically defines the sequence of trials the participant experiences. In Chapter 2, we took the key elements of the experimental procedure as described in the abstract and created the following diagram:

Select Stimulus → Fixation → Probe Display/ Collect Response
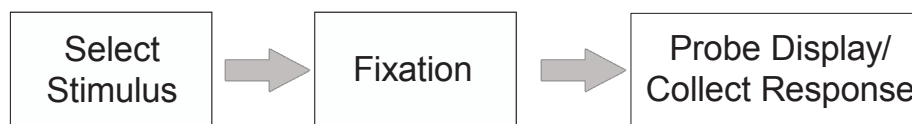
In E-Prime 2.0, the core structure of the experiment is specified through the combination of a List object (e.g., DesignList) to select the conditions, a Procedure object to specify the sequence of events in a trial, and events specific to that procedure, such as fixation and stimulus Probe Displays. The E-Prime 2.0 Structure window below shows the outline of the experimental procedure. The structure shows the outline of the experiment with the core being the DesignList and the TrialProc, including the Fixation and Probe events. We will create this structure now.
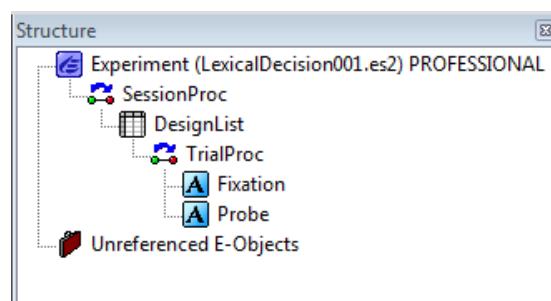
The Procedure attribute within a List object specifies what procedure to run for the selected condition. In this case, set it to TrialProc as shown in the figure below. After typing "TrialProc" in the Procedure attribute column and hitting the Enter key, E-Studio will indicate that TrialProc has not yet been created and will ask to create TrialProc. Answer "yes" to this prompt. You will then see a prompt to make TrialProc the default value for the Procedure attribute for all new levels; answer "yes" to this prompt. (Since row ID #2 was created before you made TrialProc the default procedure, you must also enter TrialProc as the value of the Procedure cell for the second row.) The List object should look like this:

In Stage 1, we will not modify the sampling settings, and will leave the List with the default settings (one cycle of two samples with sequential selection). We will change this in Stage 3 .

Put the trial events on the TrialProc timeline and rename the objects
Double click the TrialProc in the Structure window to open the trial procedure timeline in the Workspace. Displays are added to the Procedure by clicking an object from the E-Object Toolbox (TextDisplay, ImageDisplay, Slide, SoundOut, MovieDisplay, FeedbackDisplay) and dragging it to the TrialProc timeline. For the lexical decision experiment, we need two TextDisplays, one for the Fixation and another for the Probe. To create the Fixation and the Probe Display objects, click the TextDisplay object icon in the Toolbox (you may need to scroll down through the Toolbox to see the TextDisplay object) and drag a TextDisplay object to the TrialProc for the fixation. Repeat to create the Probe Display. The figure below shows both the Structure window and the TrialProc after putting the TextDisplays on the TrialProc and renaming them appropriately.
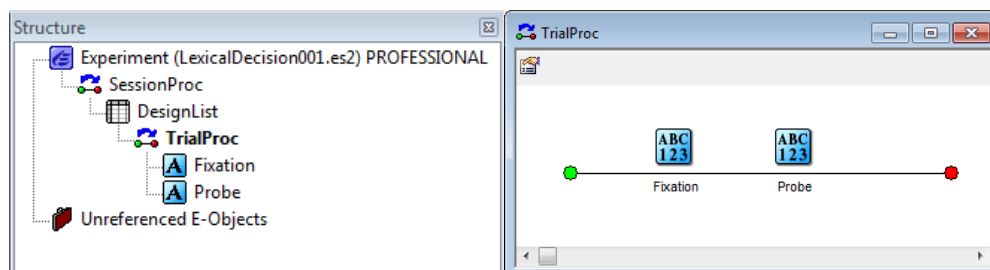
Specify the stimuli that the participant will see

Each object has many properties. The properties of each trial object/event must be modified if the desired settings differ from the default settings. If the display property of a display object (i.e. what appears on the screen) does not vary throughout the experiment, it is set to a constant (e.g., the text "+" for fixation). When the display property varies (e.g., the stimulus of "cat" or "jop"), it is set to refer to an attribute value (such as [Stimulus]) defined on a List object) see below.

In Chapter 2, we identified the following fixed and varying display properties:

Object	Fixed Properties	Varying Properties
Fixation	Present "+" in the center of the screen	(none)
Probe	Display in center of the screen (default) Duration = 2 seconds Input keys = "1" and "2" Terminate display upon response Foreground color = black (default) Background = white (default)	Stimulus (e.g., "cat", "jop", etc.) Correct response = "1" or "2"

In the lexical decision experiment, the participant sees a "+" for the Fixation. Double-click on the Fixation TextDisplay object to open the object in the workspace. Then, type a "+" in the text area of the Fixation TextDisplay object.
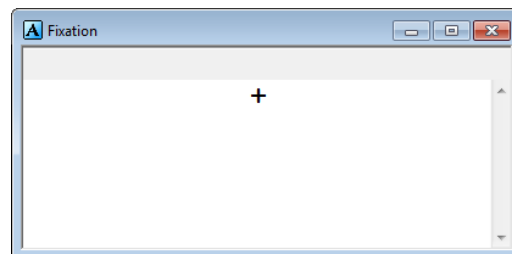
The default display duration for TextDisplay objects is 1000 ms, which matches our experiment specification. Therefore, no other changes are needed on the Fixation TextDisplay object.

The participant sees a value of the Stimulus attribute for the Probe Display. For the Probe stimulus, the text varies from trial to trial. In general, a property that varies is specified by first creating an attribute in a List object, then assigning values to the levels (i.e., rows) of that attribute, and lastly referring to the attribute in other objects. In this case, we have already completed the first two items; the text string is specified in the Stimulus attribute (column) of the DesignList. Select the Probe object by either double-clicking the object from the Structure window or select the Probe window in the workspace. To reference the attribute, enter the attribute name enclosed in square brackets [ ], i.e. [Stimulus]. In E-Prime 2.0 Professional, when you type "[" AttribSense will bring up a list of existing attributes. Click "Stimulus" to automatically enter the attribute reference (i.e., [Stimulus]) on the Probe Display. Entering [Stimulus] refers to the attribute that we named Stimulus that and which can be assigned multiple values (e.g., "cat" or "jop"). The Stimulus attribute (i.e., [Stimulus]) is entered into the Text field of the Probe object to present different strings across trials. On a given trial, a row is selected from the List object, the value of the Stimulus attribute is determined (e.g., "cat" or "jop"), and this value is substituted for [Stimulus] on the Probe object.
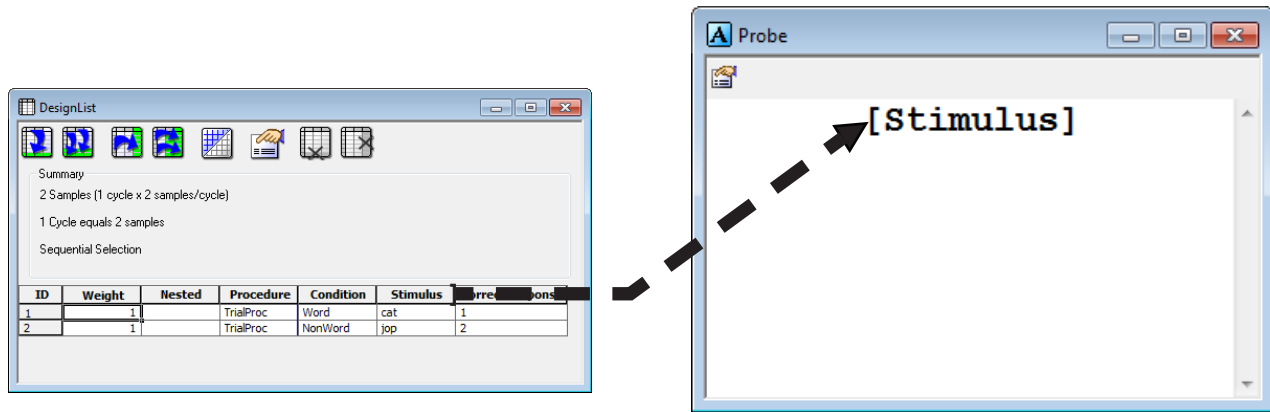
Set the non-default and varying properties of the Trial events

Each object's properties can be set using either the Properties window or the properties pages. The Properties window lists the properties related to a specific object in alphabetical order. The Property pages arrange the properties in meaningful groupings under a common tab (e.g. the tabs for the TextDisplay object are Common, General, Frame, Font, Duration/Input, Sync, Logging, Task Events and Experiment Advisor.) In general, it is recommended that the user use the Property pages to set the properties.

For the lexical decision experiment, the Probe Display requires some non-default properties. The arrows in the figure below show the settings for these properties. First, if it is no longer open, double click the Probe object (in the Structure window) to open it in the Workspace. Then, click the Property pages tool button (top left corner of the Probe window) to open the Property pages. Select the Duration/Input tab by clicking on it.

The Duration/Input Property page should be modified as follows; see the arrows in the figure to locate the fields of interest.

Set the Duration to 2000 in order to allow up to 2 seconds (2000 ms) for the stimulus to be displayed.

Designate the keyboard as the Input Mask device. To do so, click the Add button in the Input Masks list (lower left) to add an input device. Select the keyboard option. (Notice that this results in new default values appearing in the Allowable field.)

Set the allowed response keys in the Allowable field to 1 and 2.
Set the Correct field to refer to the CorrectResponse attribute (which we defined previously on the DesignList object by entering [CorrectResponse]).

Data logging prompt occurs as soon as you enter the correct response.
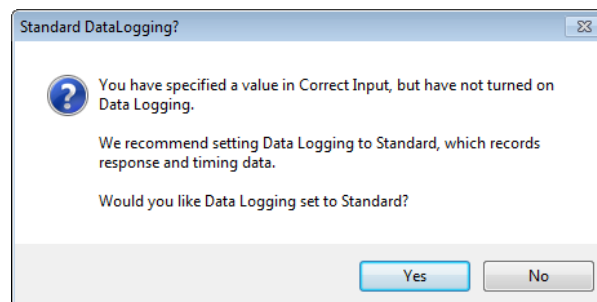
Notice that the End Action is set to the "Terminate" option to terminate the display when the participant responds.
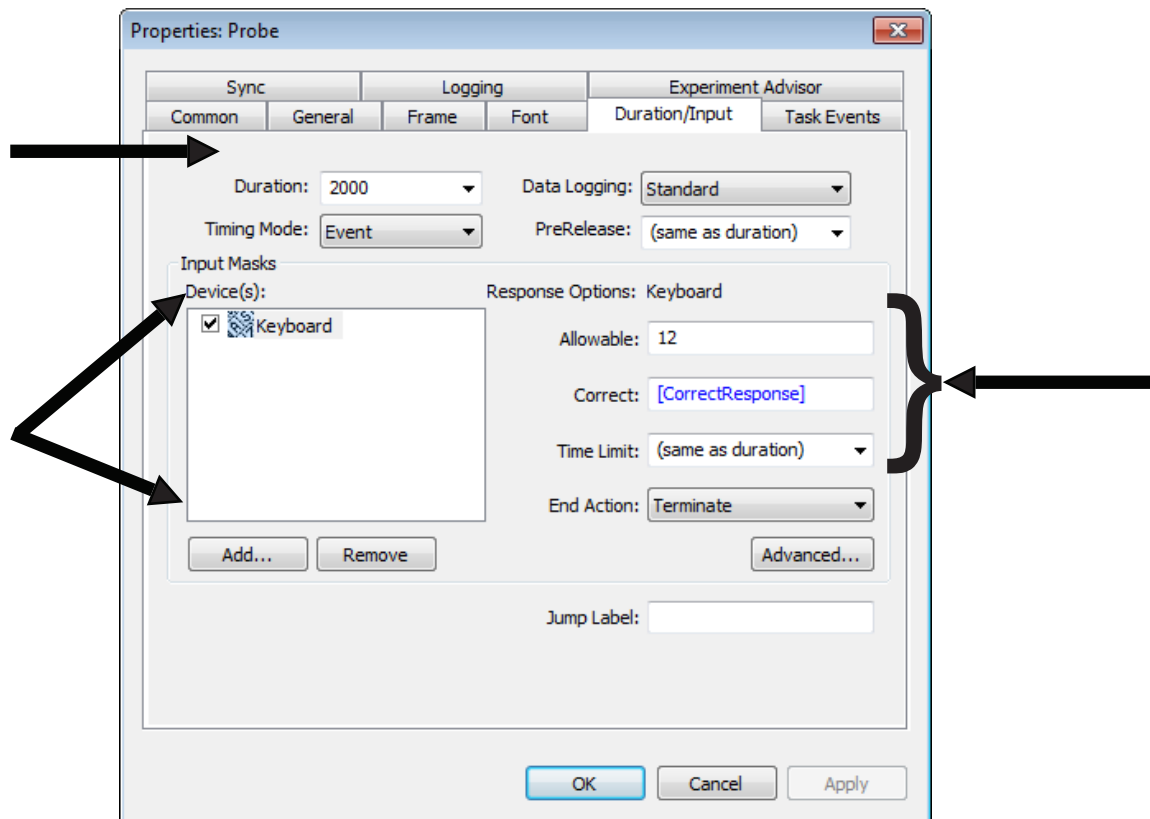
Since we entered a value in the "Correct" field but have not yet enabled data logging (the default value is "none"), E-Prime 2.0 flags this situation for you. You are prompted to turn on standard data logging. Answer "yes" to the prompt, and the Data Logging field value will now contain "Standard", as shown above. Data logging is described in detail in *Stage 1, Step 3: Data Logging* below. Click "OK" or "Apply" to save these settings.

There are more Property pages for the Probe object in addition to the Duration/Input page. Click on each of the Property pages and examine the fields. These fields are described in the *E-Prime 2.0 New Features/Reference Guide* and in the E-Basic Online Help.

If the stimulus is something other than simple text, other objects can be used to be present the probe stimulus: ImageDisplays present bitmaps, SoundOut objects present audio file output, Slide objects present a combination of text, bitmaps, and sounds, and MovieDisplay objects present movies. As with the TextDisplay object, all of the preceding objects can also collect responses via the Duration/Input property page.

Stage 1, Step 3: Data Logging

In E-Prime 2.0, each stimulus presentation object is capable of logging multiple properties; it is the experimenter's responsibility to explicitly set the logging option for each critical object so the data will be logged for analysis. In *2.1 Stage 1: Conceptualize the Core Experimental Procedure*, we identified the following information to be logged for each trial of the lexical decision experiment:

• What was the response that the participant entered (i.e., 1 for word and 2 for nonword)?
• What was the response time to the Probe Display?
• What was the accuracy (i.e., 1 for correct and 0 for wrong)?

In E-Prime 2.0, each stimulus presentation object (e.g. TextDisplay, Slide, MovieDisplay) has two tabs that control data logging: the Duration/Input Page and the Logging Page. Each type of Property page is described below.

*Duration/Input property page: Standard data logging for objects collecting data for analysis*
By default E-Prime 2.0 logs all of the attributes defined on a List Object. This means that for each Attribute that has been created, each time that a row is sampled from the List object, E-Prime 2.0 logs the value in each Attribute cell that has been selected. For the lexical decision experiment, this means that E-Prime 2.0 will automatically log the values that were selected for the attributes "Condition", "Stimulus", and "Correct Response" on each trial.

The experimenter must go to each object that collects a response and specify what response data is to be logged. For most experiments, this will involve setting the data logging option on the Duration/ Input tab to the Standard option. This logs the response variables of RESP and ACC for the response key, and RT for the response latency. As a reminder, E-Prime 2.0 asks to set the Data logging option to Standard whenever a value is specified for the Correct field, as you saw above in the Stage 1, Step 2. E-Prime 2.0 does this because of the high likelihood that if there is a correct answer to the display object, then you are likely to be interested in analyzing the response. In addition, the Standard data logging option records Time Audit data to verify the timing of the experiment. This includes the OnsetDelay and DurationError properties, which list timing data to check timing precision (see *Chapter 4: Critical Timing*).
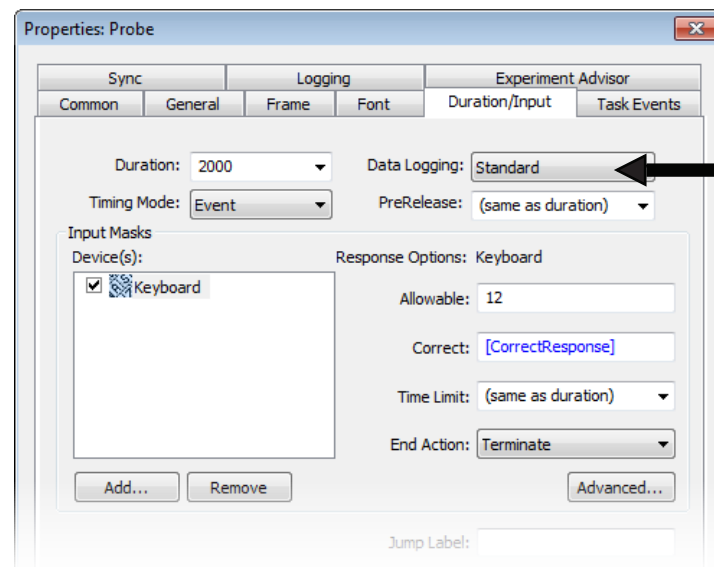
Logging Property Page: Set specific logging as needed
The Logging tab enables data logging options to be fine-tuned to your specific needs. The data logging properties that can potentially be logged for the object are presented in a list, and are arranged in categories (Dependent Measures, Time Audit, etc.). If the Data Logging field on the Duration/Input page has been set to Standard, Response Only, or Time Audit Only, the corresponding properties on the Logging property page will be checked. However, it is possible to select the properties independent of the Data Logging options by clicking the checkbox next to a specific property. You can also select all of the properties within a category, or any contiguous group of properties within or across categories. To do so, click on the first property name in order to select it. Next, hold down the Shift key and click on the last property name to be selected; the list of items from the first to the last item inclusive will be highlighted. Last, release the shift key and select any one of the check boxes; all of the items that were highlighted will now be checked. A fuller description of each data logging property is provided in the *E-Prime 2.0 New Features/Reference Guide*.
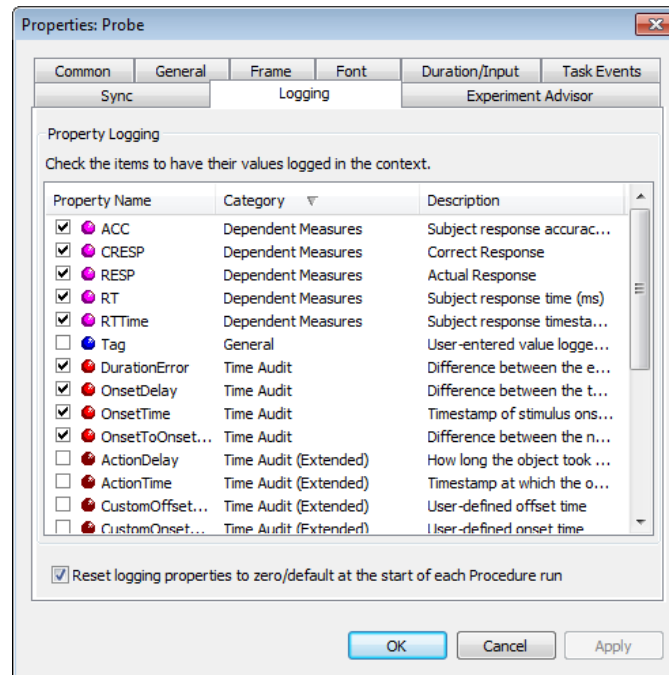
Why doesn't E-Prime 2.0 simply log everything? Well it could, and technically, if Data Logging is set to Standard on every object in the experiment, it would. However, logging everything greatly expands the data that is stored in an experiment and, more seriously, increases the number of variables to go through when analyzing an experiment. Note, logging every property of every object in a typical experiment could require logging several hundred variables per trial. The vast majority of these variables will never be examined (e.g., the response of hitting the spacebar to advance the instruction display). The default settings in E-Prime 2.0 log all of the design variables (specified in List objects) and require the user to specify logging on all objects that collect critical data.

Run and verify the core experimental procedure
At this point, the core experiment has been constructed, so it is time to test the experiment. Get into the habit of saving your changes in E-Studio prior to generating and testing your changes. To save, either select "Save" from the File menu or use the short-cut key combination of Ctrl+S.

After saving your changes, click the Generate button to generate the experiment. (If the toolbar is not visible, then select "Toolbar" from the "View" menu.) Alternatively, you could select the "Generate" option from the E-Run menu.

It is useful to view the generate results using the Output window in E-Studio. If it is not already visible, the Output window may be displayed by choosing the Output option from the View menu or via the Alt+3 shortcut. You want to resize the Output window in order to make more of the information in the window visible. If no errors are generated during the generation process, a message indicating a successful generation will be displayed in the Output window. The Experiment Advisor tab will also appear on the output window with any warnings.
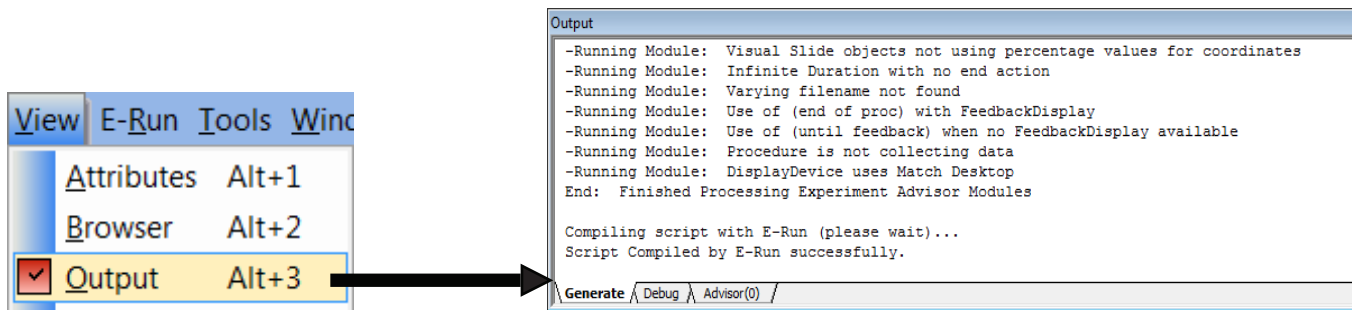
If errors occur during generation, an error dialog is displayed, and the error message is sent to the Debug tab in the Output window. The error message will indicate the line in the script at which the error occurred, and the Script window opens to display the line at which the error was encountered. In addition, when working with the Professional version, the InLine object containing the problem script is also opened in the workspace and given the focus, which further clarifies where the problematic script is located and enables you to immediately begin editing the script.
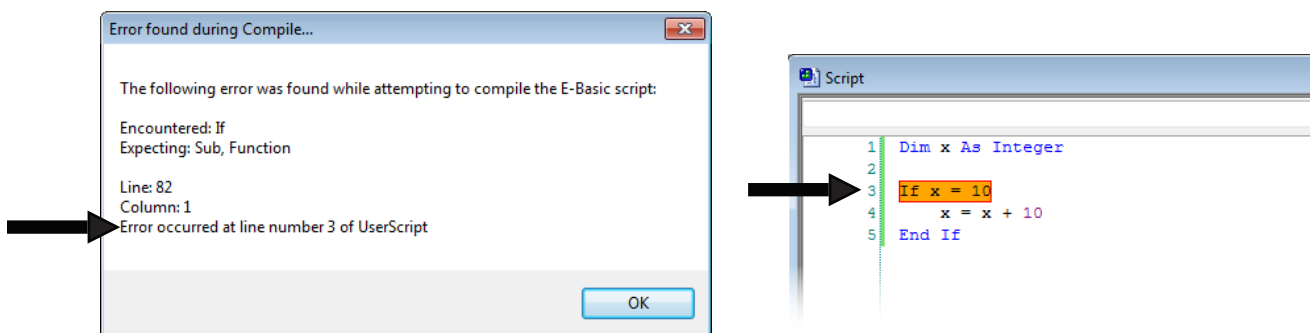
Run trials responding correctly, incorrectly and not responding

Run the experiment to test the core procedure. Click the Run button to launch E-Run and run the experiment. In E-Prime 2.0 Professional, to start an automatic test, select E-Run Test, choose a speed, and click test.

⚠️ **NOTE:** *Each experiment run begins by asking for the subject number and session number (not shown in this example).*

It is important to test the limits of an experiment to ensure that it behaves desirably, and that scoring is occurring correctly. As the experiment is specified in the current example, the "1" and "2" keys are the valid input keys, and the stimulus display is set to terminate upon valid input. First, it is important to run "normal" trials, responding both correctly and incorrectly in order to verify that the valid keys are being accepted as input and that the display is terminating upon response. If the participant presses a valid response key during the 2000 ms display, the display will terminate and the next trial will be presented.

However, participants don't always follow the instructions, so it is necessary to test the deviant cases as well. For example, what happens when a participant presses a key that is not a valid response key, or perhaps does not respond at all?   In the current example, pressing the wrong key or a non-response will not terminate the display, but the stimulus display will terminate when the maximum duration for the stimulus object is reached. If the participant presses an invalid key, or does not respond within 2000 ms, the experiment will simply continue with the next trial. At this point, it is important to verify that this is how the trial procedure is working. Later, it will also be necessary to verify the distinction between these trials (e.g., valid response versus invalid or non-response) through examination of the data file.

Verify expected displays

Verification of the core experiment also involves the examination of each of the displays. The Fixation display presents a "+" for 1000 ms. This display should be consistent in that the display does not vary, nor does it terminate if the participant presses a key. However, the Stimulus display should vary. The stimulus string presented should be different for each trial, and the duration of the display is dependent upon the participant's response (i.e., the stimulus is displayed until the participant responds by pressing a valid key, or until the maximum duration of 2000 ms has expired).

Start E-DataAid and load the data

Once the core procedure has been verified, it is necessary to examine the data file in order to determine that the information is logging correctly. The application that allows viewing and editing of data files is E-DataAid. Open E-DataAid using the Tools menu in E-Studio. In E-Prime 2.0 Professional, to view the most recent data file, click the hyperlink in the Advisor tab of the Output window.
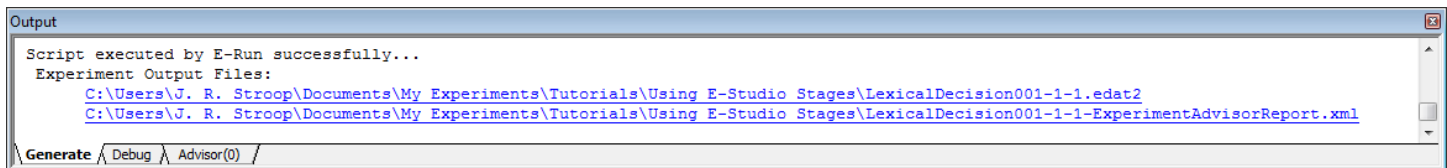

When E-DataAid is launched, a screen such as the following will appear:
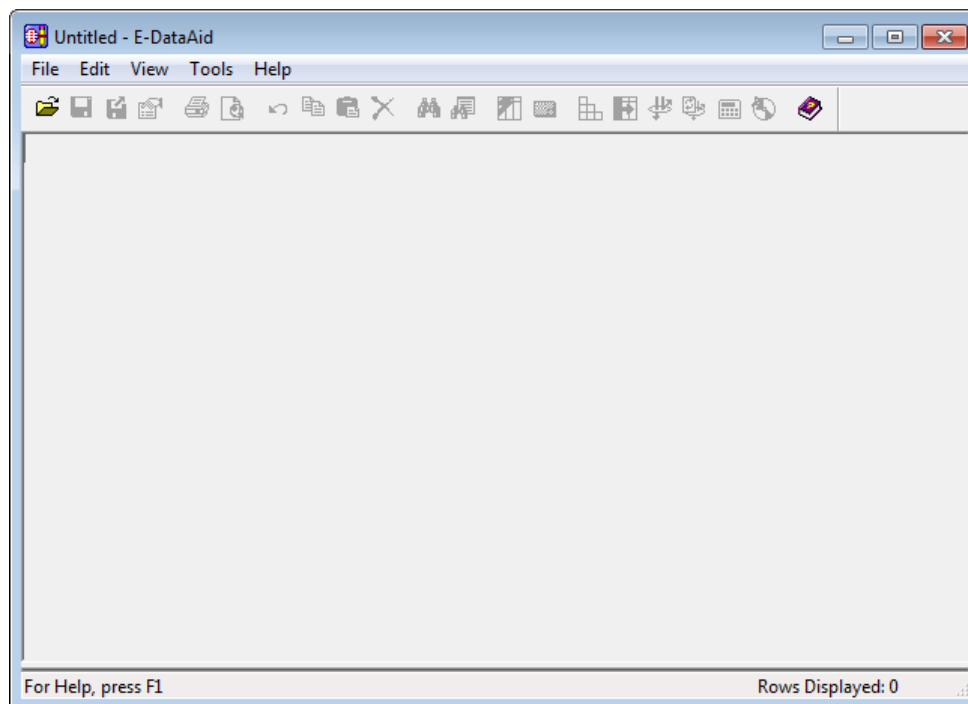
Until a data file is opened, no data can be displayed. Use the Open tool button to open the data file for the experiment. The data file name is determined by concatenating the Experiment name, the subject number, and the session number. If you followed the suggestion for naming the experiment LexicalDecision001 and ran as participant #1, session #1, the data file would be named LexicalDecision001-1-1.edat2. The data file is created in the same folder as the .ebs2 file[1].

---

[1] E-Prime 2.0 provides three mechanisms to change the location of the .edat2 file: setting the E-DataPath environment variable, using a PackageFile call or specifying an alternative location in the StartupInfo editor. Details on each of these methods are provided in the *E-Prime 2.0 New Features/Reference Guide.*
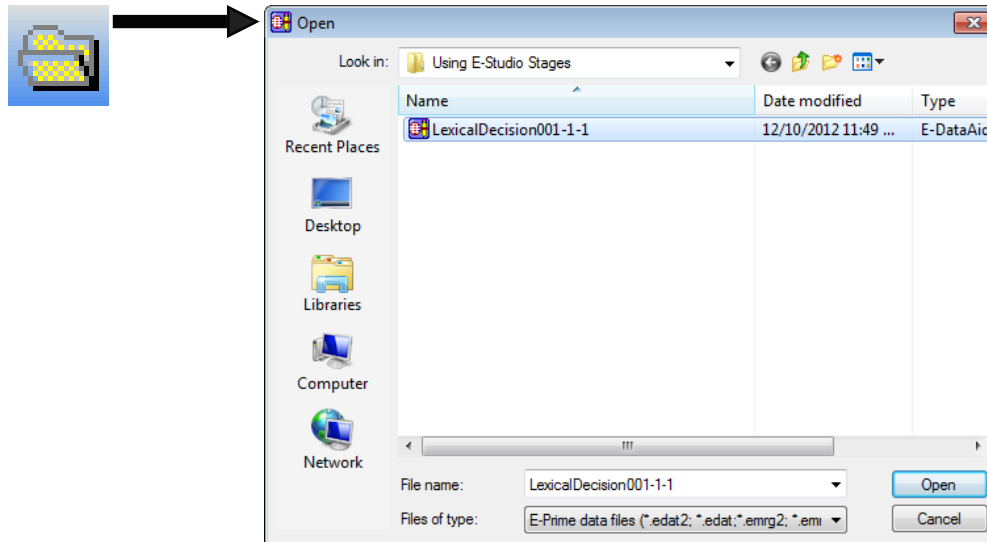
Verify the number of rows are the expected number of trials

The number of rows in the data file is equal to the number of trials run during the experiment (assuming the trial level is the lowest level). For our experiment, only two trials were run, so only two rows are included in the data file.

Determine List attributes and numbers of trial conditions occurred as expected

Within the data file, the variables are arranged in alphabetical order by level (e.g., Block, Trial, etc.) and may require some searching in order to find those columns of utmost interest. The Arrange Columns dialog will allow hiding or moving columns to display only those wanted to view, and to display them in a specific order.
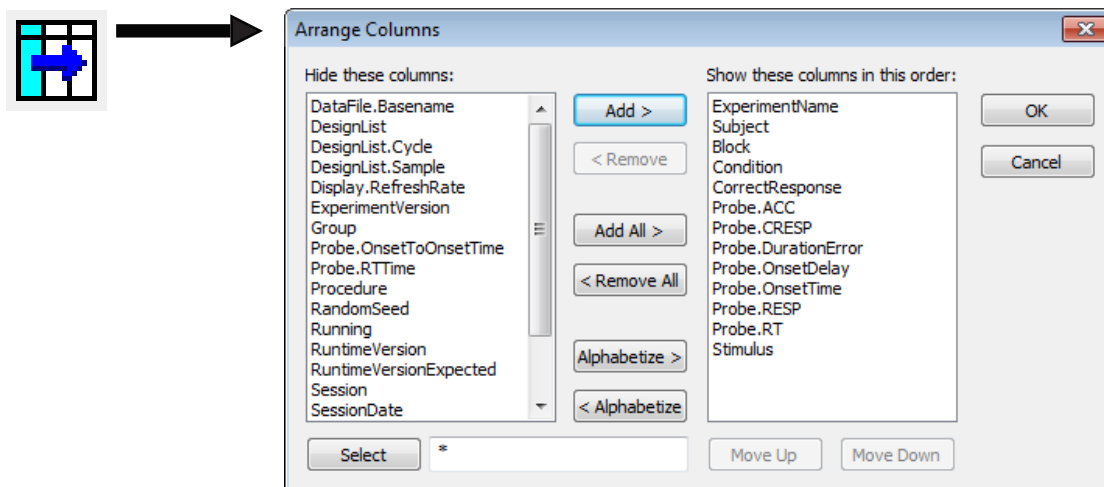
Arrange the columns as shown in the figure above, and click OK. Verify that the conditions, stimuli, and correct responses (i.e., independent variables) are logged as expected.

Find the dependent variables and verify they are set reasonably
Verify the logging of the dependent measures. The RESP property is the actual response entered, and is logged as NULL for a non-response. The CRESP property is the correct answer, which varies per trial, and ACC is the accuracy of the response (RESP) in relation to the correct answer (CRESP). ACC is set to "1" for correct answers and to "0" for incorrect answers or omissions. RT, of course, is the reaction time for the response measured from the onset of the object collecting the input. Verify that the values logged for the dependent variables are as expected for the test trials (e.g., RT values are appropriate for short and long response times, "0" for omissions, etc.).

Check timing duration accuracy on fixed length displays
Verifying timing precision is a critical function. Issues related to timing are detailed in *Chapter 4: Critical Timing*. Here we will only provide brief accounts of timing issues. In general, in E-Prime 2.0, the time from stimulus to response is accurately recorded. The duration from stimulus to stimulus may be somewhat longer than specified in the Duration field. This is because the display is written at the time of the vertical retrace (when the video monitor output of the image goes to the top line of the display). The time from one display to the next is equal to the duration of the first display plus the time until the next vertical blank interval. In this experiment, the Fixation had a duration of 1000 ms. The actual time from the Fixation to the stimulus would be equal to the time of the duration of the Fixation plus the OnsetDelay of the Probe object. It is important to verify this duration in order to accurately report the measured fixation duration. In our experiment, use E-DataAid to examine the Probe.OnsetDelay variable (should range from 0 to 17 ms). To illustrate, if the Probe.OnsetDelay is 12, then the fixation duration would be the duration (1000 ms) plus 12 ms, or 1012 ms in length.

3.2 Stage 2: Elaborate the Trial Procedure

⚠ **NOTE:** *This section builds on the experiment created during the previous section. A finished version of Stage1-LexicalDecision001.es2 is located in the …\My Experiments\Tutorials\Using E-Studio Stages folder.*
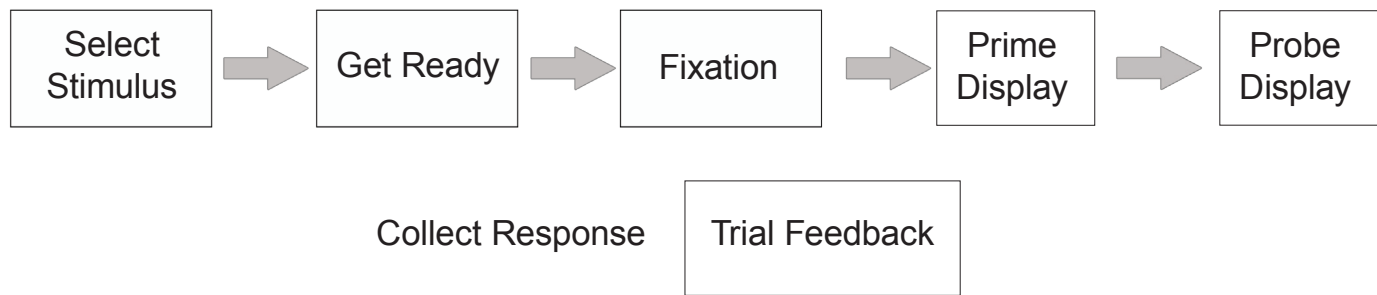
The goal of Stage 2 is to elaborate the basic Fixation-Probe trial procedure. Three displays will be added and existing displays will be modified. A display will be added at the beginning of the trial procedure to permit the participant to pace the trials and instructions will be added to the Fixation display to remind the participant of the response keys. Also, a Prime Display will be added prior to the Probe Display, and Feedback will be added to the end of the trial procedure. After these displays are added, the experiment will be re-generated and run, and the new displays will be verified.

There are two major steps to the Elaborate the Trial Procedure, as shown below:

| Stage 2: Elaborate the Trial Procedure |
|---|
| 1) Add the Missing Displays
    • Add Get Ready Display
    • Add Fixation and Probe Display Instructions
    • Add Prime Display
    • Create the new PrimeType Attribute
    • Add Feedback Display |
| 2) Run and Verify the Get Ready, Prime, and Feedback Objects |

Stage 2, Step 1: Add the Missing Displays

In *Chapter 2: Designing Your Experiment* we modified the Trial Procedure as follows:

| Select Stimulus | ➡ | Get Ready | ➡ | Fixation | ➡ | Prime Display | ➡ | Probe Display |

| Collect Response | Trial Feedback |

Add Get Ready Display

Add a TextDisplay object as the first object in the TrialProc which allows the participant to set the trial pace. Rename this TextDisplay to "GetReady."
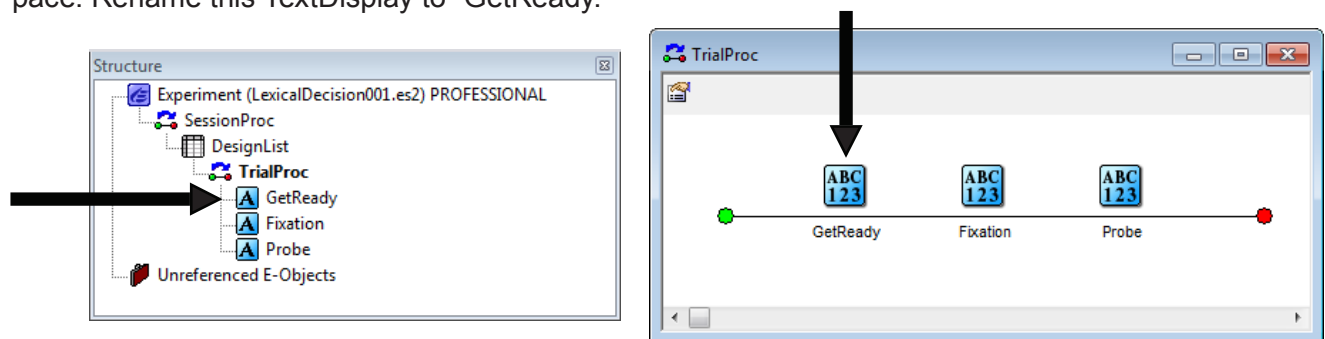
In the Text field, type "Press the spacebar when ready". To allow the participant to begin the trial when ready, set the duration to "infinite ", enable keyboard input, and enter the spacebar as the allowed input key. Recall that these response option values are accessed via the Duration/Input tab of the Property Pages for the TextDisplay object. These settings result in the GetReady display remaining on the screen until the spacebar is pressed. To specify a special key such as the spacebar as the allowed response, enter the word "SPACE" (all capitals) surrounded by braces (i.e., {SPACE}). Set data logging to "standard" to record participant wait time.
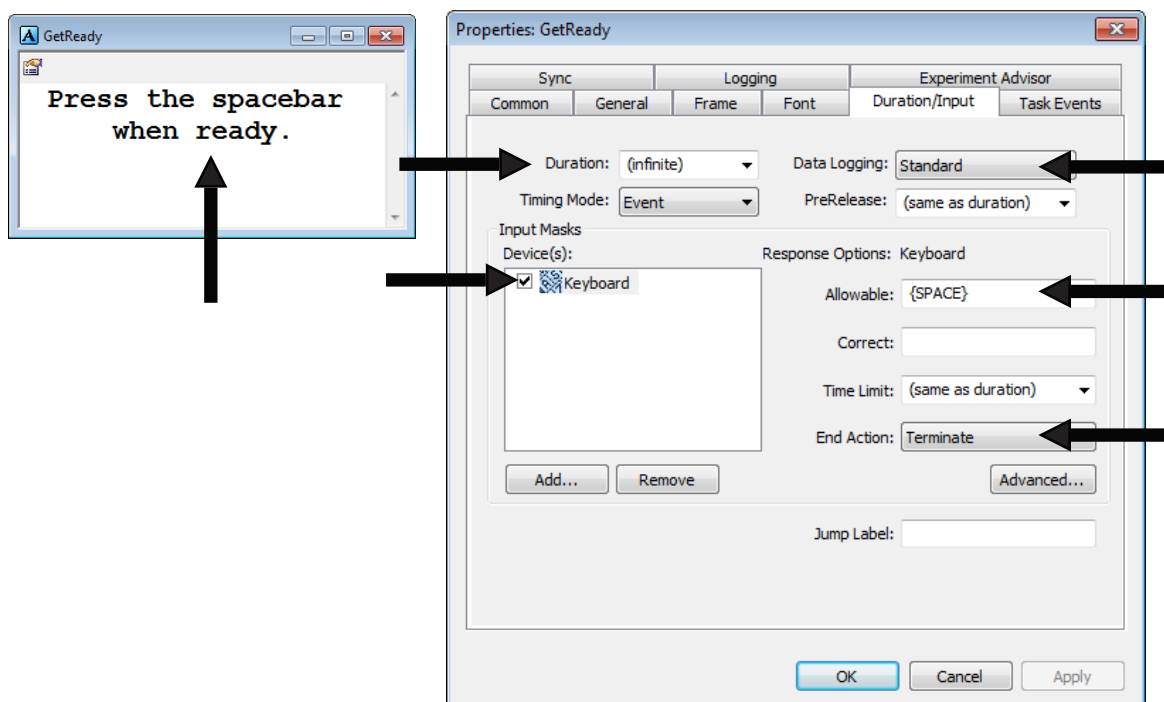
Add Fixation and Probe Display Instructions

Add instructions to the Fixation and Probe objects in the form of constant text (i.e., it will not vary from trial to trial). Open the Fixation object and do the following:

• Add five blank lines ABOVE the fixation character, so that the fixation character appears on line six
• Add four blank lines BELOW the fixation character
• Enter the text shown below to remind the participant of the response keys.

Next, open the Probe object and set the height of the display area (Frame tab) to 25%. This reduces the displayed area to 25% in the center of the screen. These settings will permit the stimulus to overwrite the fixation, but will not erase the response instructions.
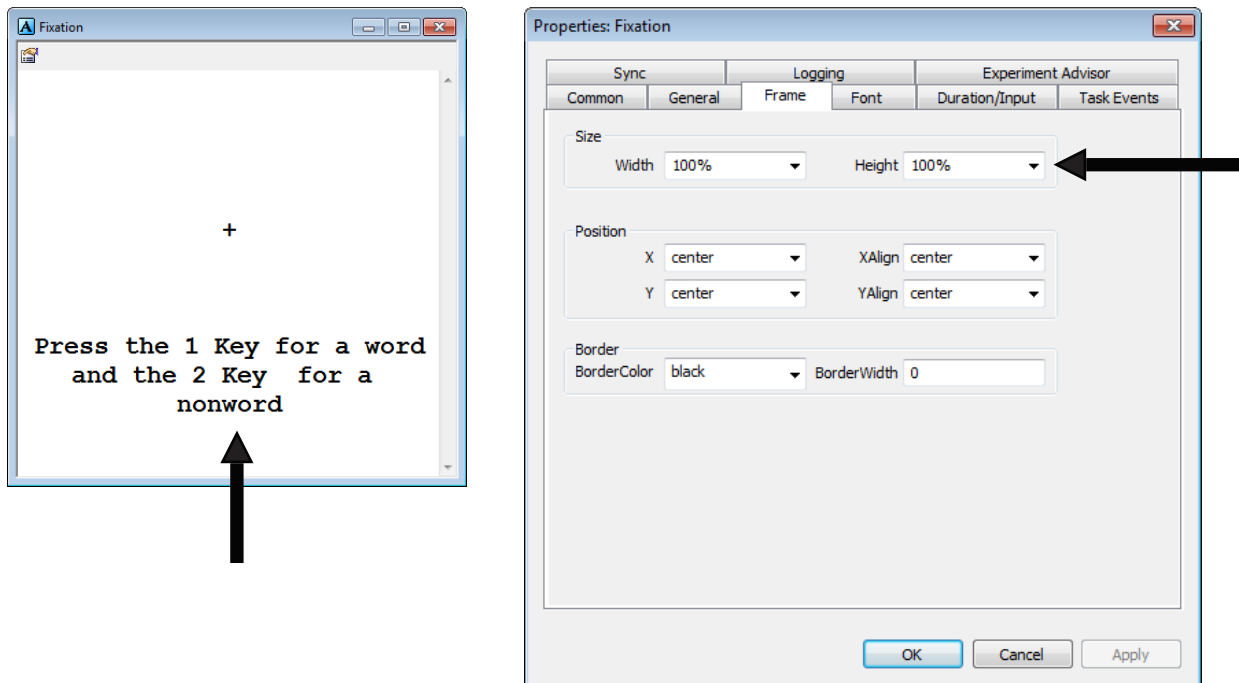
Add Prime Display

Add a TextDisplay object following the Fixation object in the TrialProc, and rename this object "Prime."
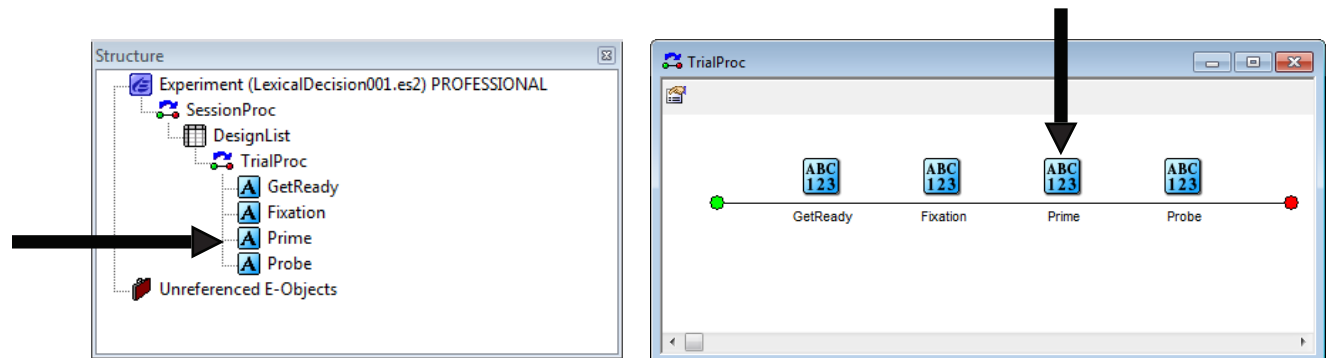
Open the Prime object and type "[PrimeType]" into the Text field (i.e., the Prime text will be entered as an attribute value), as below. On the Duration/Input tab, accept the settings of duration 1000 ms, and no input enabled (these are the default values). Lastly, as you did for the Fixation, set the frame height for the Prime object to 25% so that the instructions from the Fixation are not overwritten. Click 'OK' to accept these settings for the Prime TextDisplay object.
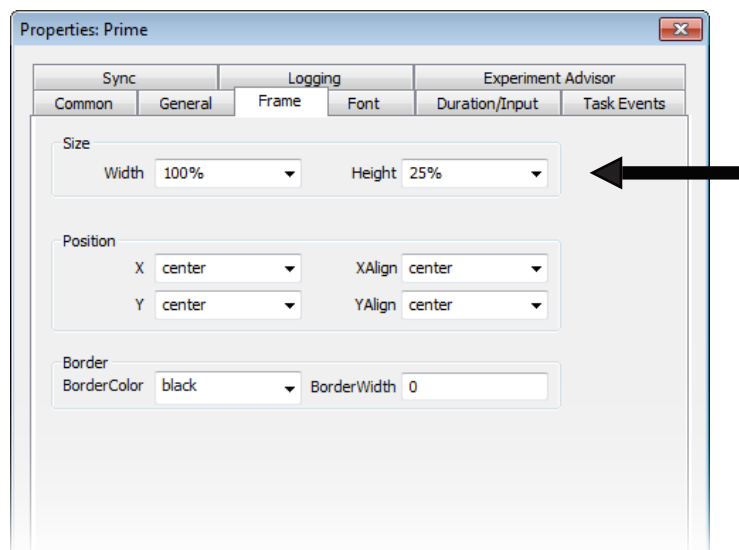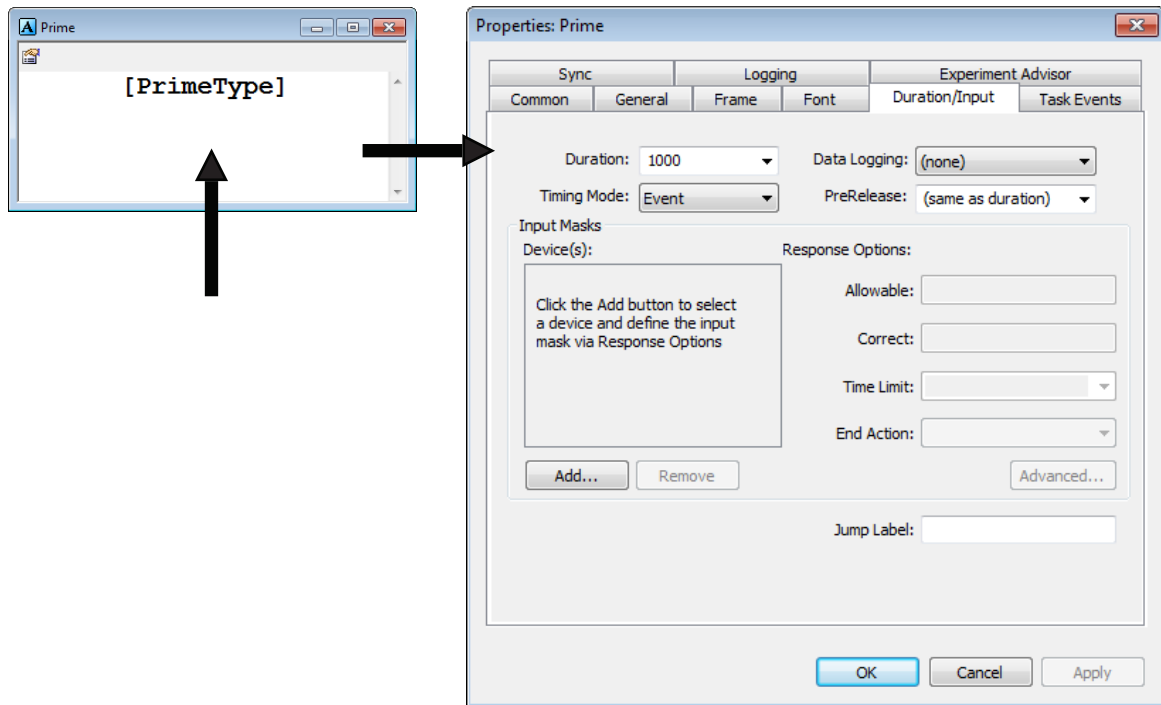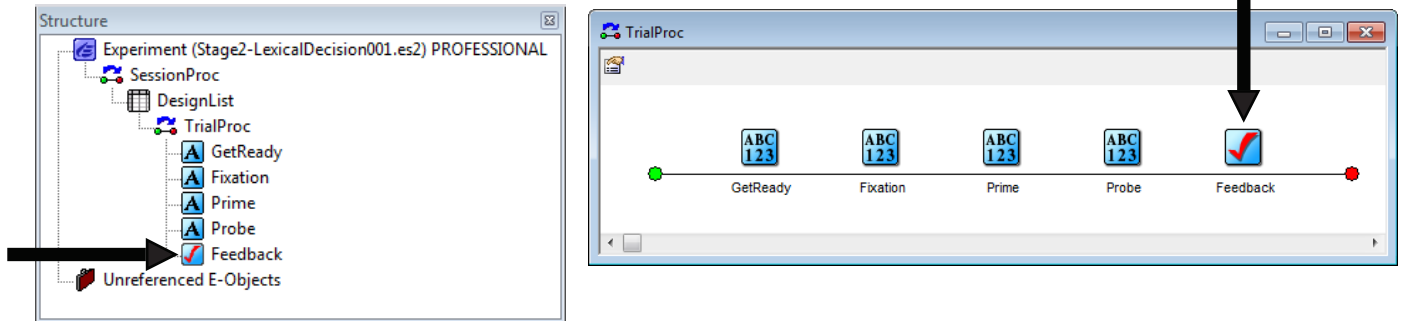
Create the new PrimeType Attribute

Add an independent variable for coding PrimeType. Open the DesignList, add an attribute, and name it PrimeType. Enter the values for PrimeType to be "word" and "non-word" as illustrated below:
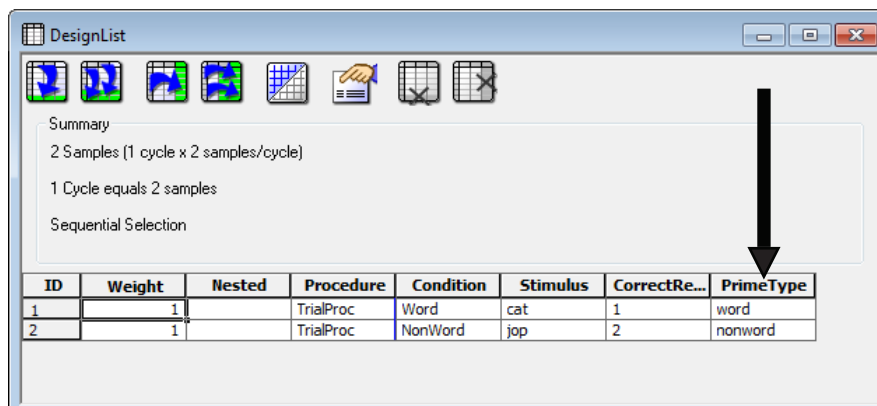
Add Feedback Display

Add a FeedbackDisplay object as the last event in the TrialProc, and rename it to Feedback.

Double-click on the Feedback object in the TrialProcedure to open it in the workspace. The Feedback object is used to present feedback to the participant after each trial. A Feedback object is a specialized type of Slide object. A slide object contains sub-objects. Sub-objects contain independently specified content and formatting. The master object that controls the sub-objects, the Feedback object in this case, contains general settings that affect all of the sub-objects.
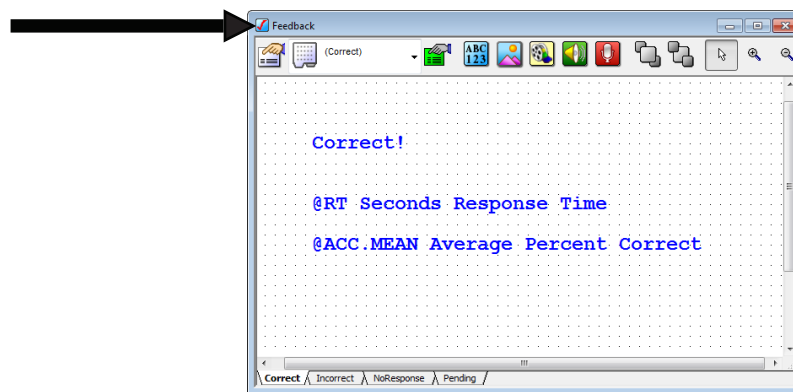
The Feedback object contains default specifications for the following slide states: Correct, Incorrect, NoResponse, and Pending. The sub-objects correspond to the tabs that appear along the bottom of the object. The CorrectResponses sub-object is displayed initially. Select a different tab to display the settings for another sub-object.

No changes are needed to the default settings for the slide states. However, the display duration of feedback needs to be changed from the default of 1500 ms to 2000. Click the Properties button and then the Duration/Input tab to set the properties for the Feedback object. Set the Duration for the Feedback object to 2000 ms.

46
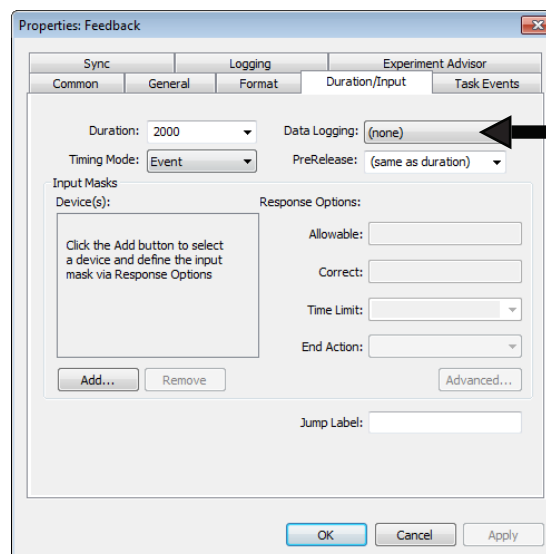
Instruct the Feedback object to display feedback based on the scoring of the Probe object by setting the Input Object Name (General tab in Feedback properties) to Probe. Accept the standard statistics settings (collect ACC stats, No Response ACC Stats, RT Stats, and Incorrect RT Stats) in order to collect reaction time and accuracy values, and log non-responses as "Incorrect".
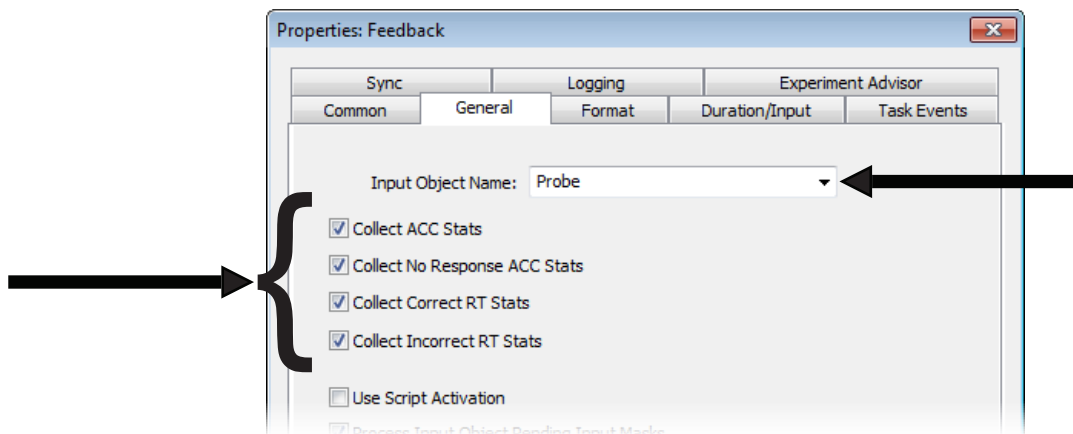
Stage 2, Step 2: Run and Verify the Get Ready, Prime, and Feedback Objects

Save the experiment, generate it, and run it to verify that the new additions to the trial procedure work as expected for all possible participant responses. Remember to test deviant as well as expected cases (e.g., correct, incorrect, and invalid key responses).

3.3 Stage 3: Add All Conditions, Set Number of Trials and Sampling

⚠ **NOTE:** *This section builds on the experiment created during the previous section. A finished version of Stage2-LexicalDecision001.es2 is located in the…\My Experiments\Tutorials\Using E-Studio Stages folder.*

The goal of Stage 3 is to complete the DesignList object. This involves entering the complete stimulus set, and setting the properties for the number of trials and the sampling method.

Stage 3: Add All Conditions, Set Number of Trials and Sampling

1) Add All Conditions
   • Method A – Enter Conditions Directly
   • Method B – Use of Nested Lists
   • Method C – Factor Table Wizard

2) Set the Weights
3) Set the Sampling Mode and Exit Condition
4) Verify

Stage 3, Step 1: Add All Conditions

By adding a Prime Display in Stage 2, we have increased the number of conditions, so the Condition attribute name no longer makes a great deal of sense. Recall that we implemented a simple version of a prime manipulation in Stage 2: prior to viewing the probe stimulus, the participant is presented with a priming display, which is either the string "word" or "nonword". Of interest is whether or not the nature of the Prime type influences the reaction time to the probe stimulus.

Open the DesignList object in order to modify the attributes. Double click the header for the Condition attribute to rename it to "ProbeType."

ID	Weight	Nested	Procedure	ProbeType	Stimulus	CorrectResponse	PrimeType

To rearrange the columns, click on the column header to highlight the column, and then click and drag the header to a new location. A red line will appear as a visual cue for where the column will be placed when the mouse button is released.

Add exemplars (rows) to the DesignList to complete the listing of conditions and stimuli. Use one of the three methods below to complete the DesignList.

Method A – Enter Conditions Directly
⚠️ **NOTE:** *A finished version of Stage3-MethodA-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

Directly entering the list is preferred for a small number of conditions (e.g., less than 10). It involves just adding the rows and typing in the attributes. Click the Add Multiple Levels tool button to add six rows to the DesignList object, for a total of eight rows. The DesignList will be used to organize the full crossing of PrimeType and ProbeType, and we will include four stimuli (two words, two non-words). Type directly in the cells to complete the DesignList as shown in the figure below Notice the following about the revised DesignList: The second exemplar (id #2) has changed from a nonword probe (jop) to a word probe (dog), and the other corresponding attributes (ProbeType, CorrectAnswer) have changed as well. you will also need to enter values in the "Weight" and "Procedure" columns for each new level as well as for the user-defined attributes:

DesignList

Summary

24 Samples (2 cycles x 12 samples/cycle)

1 Cycle equals 12 samples

Random Selection (No Repeat After Reset)

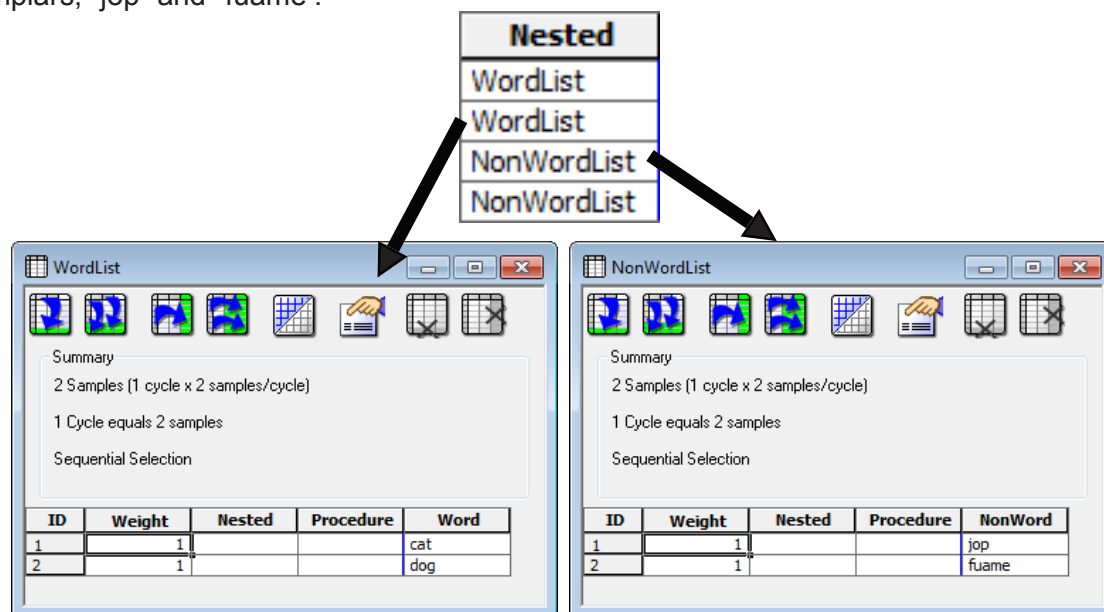| ID	Weight	Procedure	Nested	PrimeType	ProbeType	Stimulus	CorrectResponse
1	1	TrialProc		word	Word	cat	1
2	1	TrialProc		word	Word	dog	1
3	1	TrialProc		nonword	Word	cat	1
4	1	TrialProc		nonword	Word	dog	1
5	2	TrialProc		word	NonWord	jop	2
6	2	TrialProc		word	NonWord	fuame	2
7	2	TrialProc		nonword	NonWord	jop	2
8	2	TrialProc		nonword	NonWord	fuame	2

Method B – Use of Nested Lists
⚠️ **NOTE:** *A finished version of Stage3-MethodB-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

Nested lists provide a method to organize a list of items (e.g., words and non-words) and reference the organized list in the List object used to run the Procedure. One benefit to this method is the ability to set the selection method for each of the List objects independently. As a result, a list that contains a full crossing of all conditions and exemplars does not need to be created. In the case of our lexical decision experiment, you may wish to randomly sample from separate lists of words and non-words, based on the level that was selected for ProbeType on a particular trial. Contrast this with the exemplars which we added in Method A above, in which we paired each probe type word with each level of prime type. Of course, in this simple example, the full crossing of PrimeType and ProbeType with four different probe strings only resulted in 8 unique exemplars. Often, though, a full crossing of the independent variables results in more exemplars than is feasible to present within an experiment session.
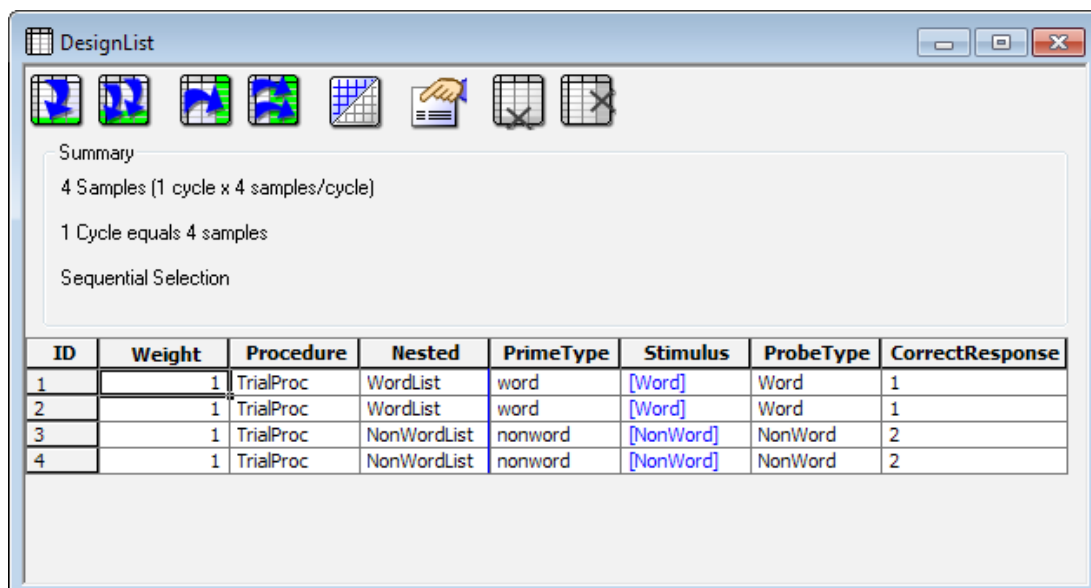
With the use of nested lists to select the probe stimulus, the DesignList object can define all of the trial types in four rows, which represents all combinations of prime type and probe type.

To use nested lists, do the following:
1) In the "Nested" column of the DesignList object, enter "WordList" as the value in row #1 (this same row should have a value of "Word" for ProbeType). Click on "yes" when prompted to create the ListObject. This row represents the Prime type = word, probe type = word condition.
2) Enter "WordList" as the value in the "Nested" column of row #2 (this same row should currently have a value of "NonWord" for ProbeType). This row will represent the Prime type = nonword, probe type = word condition, once all of the edits shown below are completed.
3) Add two more levels to the list to cover the other two combinations of prime and probe type. Enter "NonWordList" as the value in the "Nested" column of rows #3 and 4, and click "yes" when prompted to create the ListObject. (The complete specification of all attributes for these rows is specified below).
4) On the WordList list object, create a single attribute named "Word" and specify two exemplars, one for "cat" and one for "dog". On the NonWordList, create a single attribute named "NonWord" and enter two exemplars, "jop" and "fuame".

Nested
WordList
WordList
NonWordList
NonWordList

WordList

Summary
2 Samples (1 cycle x 2 samples/cycle)
1 Cycle equals 2 samples
Sequential Selection

ID	Weight	Nested	Procedure	Word
1	1			cat
2	1			dog

NonWordList

Summary
2 Samples (1 cycle x 2 samples/cycle)
1 Cycle equals 2 samples
Sequential Selection

ID	Weight	Nested	Procedure	NonWord
1	1			jop
2	1			fuame

Modify the DesignList as shown in the figure below (notice that you need to change the original values in the PrimeType and CorrectResponse columns in row #2, since this row now presents a word stimulus):

DesignList

Summary
4 Samples (1 cycle x 4 samples/cycle)
1 Cycle equals 4 samples
Sequential Selection

ID	Weight	Procedure	Nested	PrimeType	Stimulus	ProbeType	CorrectResponse
1	1	TrialProc	WordList	word	[Word]	Word	1
2	1	TrialProc	WordList	word	[Word]	Word	1
3	1	TrialProc	NonWordList	nonword	[NonWord]	NonWord	2
4	1	TrialProc	NonWordList	nonword	[NonWord]	NonWord	2

This specification has the following effect. On each trial, a row is selected from the DesignList. For illustration, assume the first row (ID = 1) is selected. The Nested column indicates that WordList is a nested list, and the experiment goes to this List to make a selection. The method of selection (e.g., sequential, random) is determined by the properties of the WordList. Let's assume the second row of the WordList is selected (ID = 2 with Word = dog). The variable [Word] is set to "dog," then the experiment returns to the DesignList to set the other attributes. In the DesignList, the Stimulus attribute refers to the Word attribute using bracket notation (i.e., [Word]). Thus, for the Stimulus attribute, the experiment resolves the value of [Word] using the value obtained from the nested list, and the Stimulus attribute is set to "dog." After the resolution of all relevant attribute values, the first trial would present the stimulus of "dog" selected from the nested list (i.e., WordList).

⚠ **NOTE:** *When using nested lists, there may be very few rows in the List object referencing the nested lists (e.g., In this example, the DesignList was reduced to two rows of Word and NonWord). This can produce a random, alternating sequence that the participant might use to predict the sequence. For example, sampling from a two row list object, the participant might notice that if the first stimulus is condition A, then the second must be condition B. It is a good idea to increase the number of conditions, or increase the Weight values of the conditions, so there are at least 4 conditions. This reduces the chances of the participant "figuring out" the sampling method and being able to predict the next stimulus.*

If the design requires that multiple items be picked from the same list, E-Prime 2.0 provides a method for doing this. A colon syntax is used to indicate the number of exemplars to be chosen from the same List during a single trial. Refer to *Colon Syntax – Sampling multiple stimuli per trial using nested lists (Page 62)* for detailed information concerning the colon syntax.
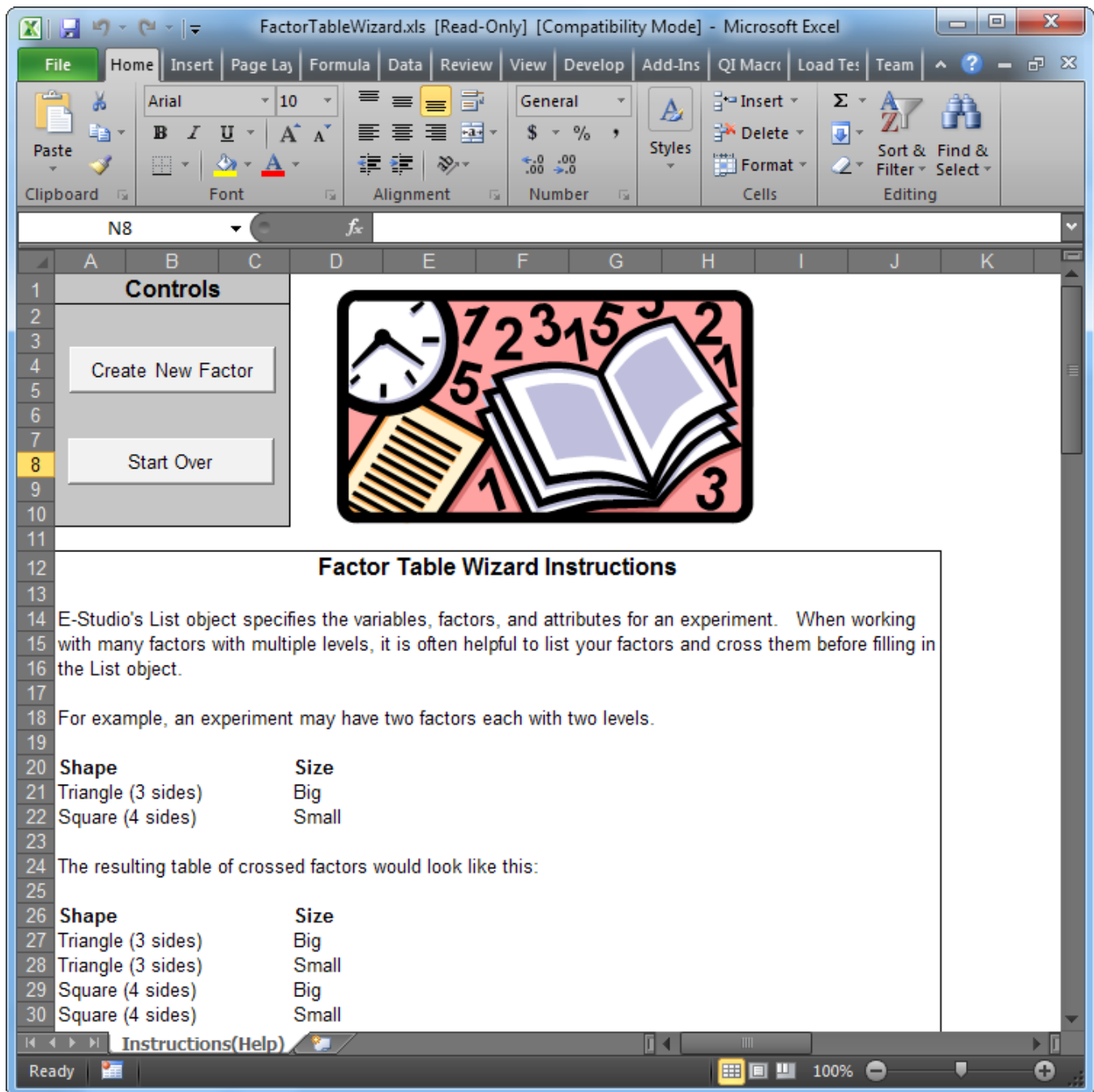
⚠ *Method C – Factor Table Wizard*
NOTE: *A finished version of Stage3-MethodC-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

For complex factorial designs, use the Factor Table Wizard to construct the full crossing of the factors and levels. This is preferred with large designs containing multiple variables. The Factor Table Wizard operates via an Excel macro to first create each individual factors and their corresponding levels, and then create a full crossing of the factors, and finally copy the conditions into E-Studio. Since the Factor Table Wizard is implemented via an Excel macro, you must first ensure that your Excel security settings are not configured to block macro execution.

We strongly recommend against selecting the "Enable all macros" option in Excel 2007. Microsoft documents this option as "leaving you vulnerable to potentially malicious code" (see Microsoft Knowledge Base article # 919195.

When any Excel workbook that contains a macro is opened, including the Factor Table Wizard workbook, the macro will be disabled by default, but Excel will alert you to the fact that a macro is available. Click on the "Options" button, and the following information is displayed:
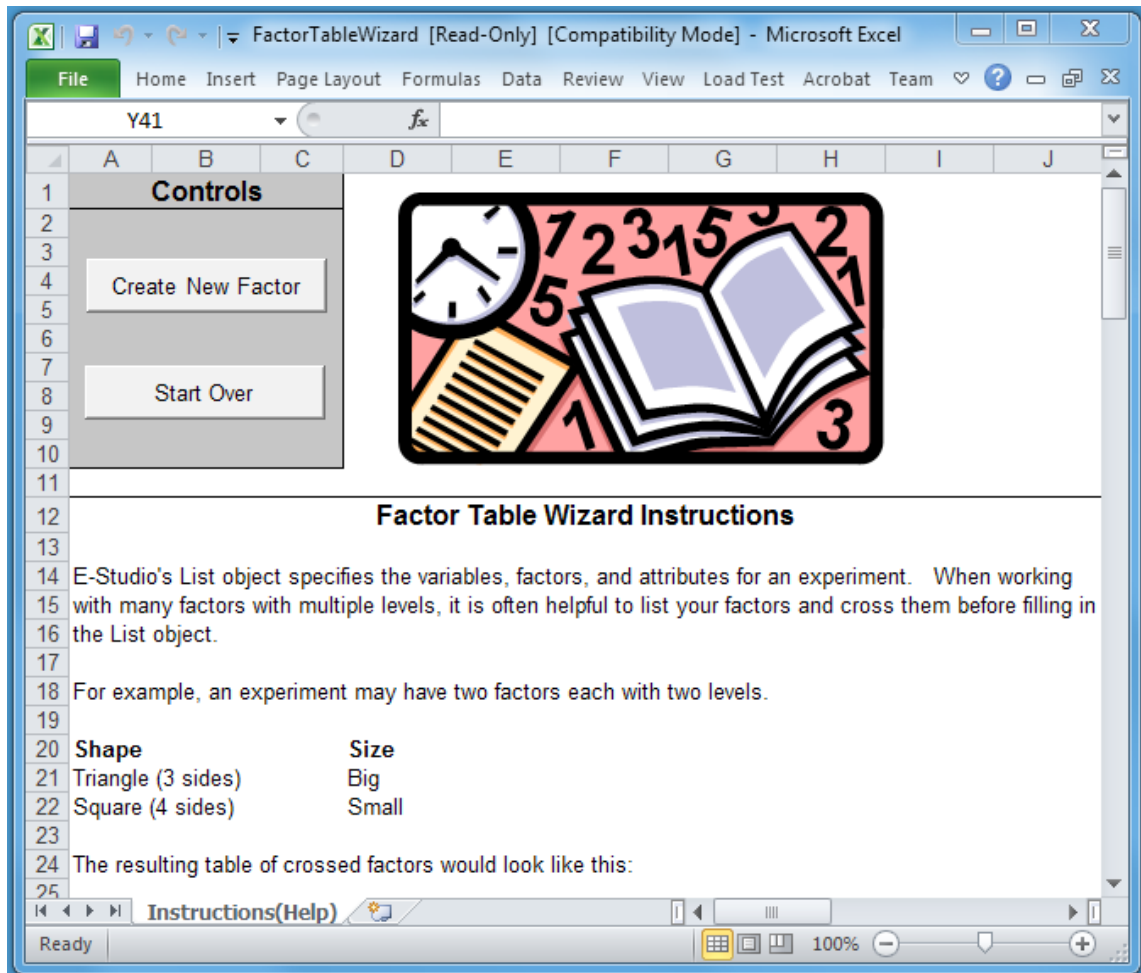
Running the Factor Table Wizard
Once your Excel security settings are configured to enable macros, launch the Factor Table Wizard from the E-Prime 2.0 start menu (All Programs > E-Prime 2.0 > FactorTable Wizard).

The Instructions (Help) worksheet is the first worksheet in the Factor Table Wizard workbook. It contains the control buttons which enable you to define factors, add levels, and ultimately cross the factors.

To create the factors for the Lexical Decision experiment, do the following:

1) Define the PrimeType factor by doing the following;
   a. Click on the Create New Factor control on the Instructions(Help) worksheet. The Factor Table Wizard prompts you for the factor name and the number of levels for the factor. Enter "PrimeType" and "2" respectively, and then click the "Create" button. The Factor Table Wizard adds a new worksheet to the workbook named "PrimeType":
   b. Enter the two levels for PrimeType, "word" and "nonword".

2) Define the ProbeType factor:
   a. Click the "Create New Factor" control (on either the PrimeType or the Instructions worksheet); specify name = Stimulus and number of levels = 4.
   b. Specify the values for the 4 levels for Stimulus in the order listed here: "cat", "dog', "jop", and "fuame".
   c. Click the "Add Column" control on the ProbeType worksheet. Name the new column "ProbeType", but do not specify any values (i.e. accept the default value of "?").
   d. Enter the following four values, in the order listed, for the new "ProbeType" column: "Word", "Word", "NonWord" and "NonWord".
   e. Click the "Add Column" control again on the ProbeType worksheet. Name the new column "CorrectResponse", but do not specify any values (i.e. accept the default value of "?").
   f. Enter the following four values, in the order listed, for the new CorrectResponse column: "1", "1", "2", and "2". When you are finished, the ProbeType worksheet should look like the following:
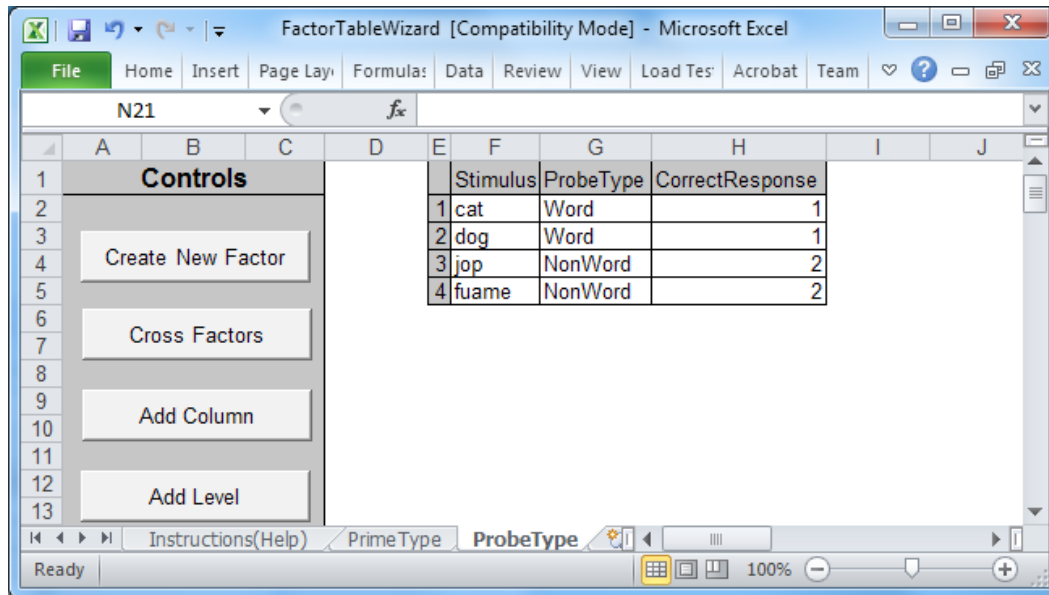
3) Create a new table of the crossed factors by selecting the "Cross Factors" control (from any of the 3 worksheets in the workbook). The Factor Table Wizard creates a new workbook named "FactorTable", with a full crossing of the factors, which results in 8 levels:
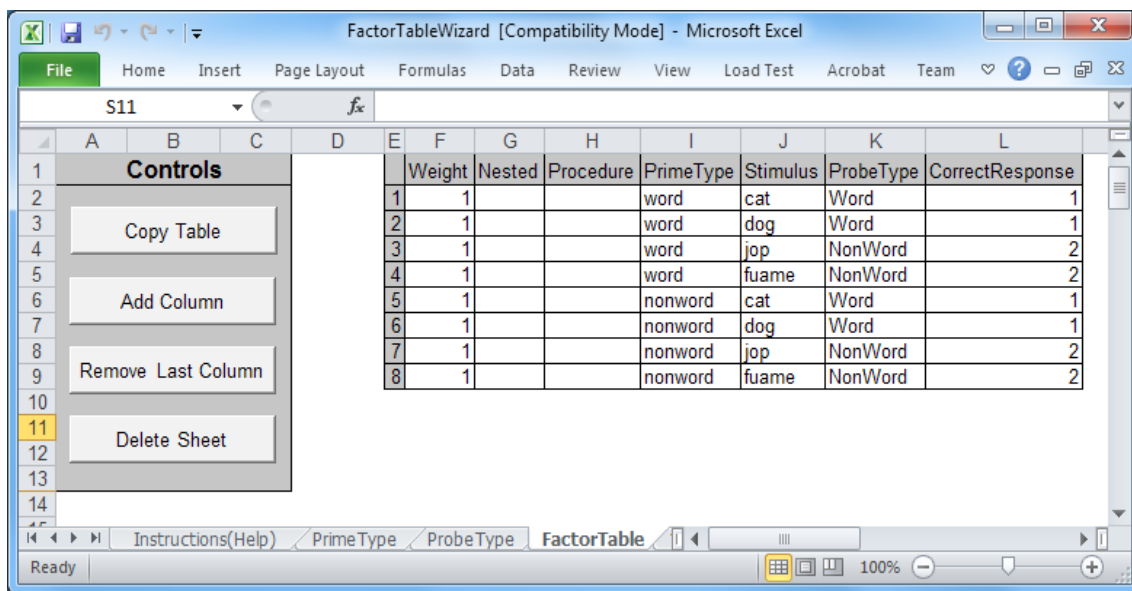
4) Lastly, set the value for each cell in the "Procedure" column to "TrialProc". The Factor Table Wizard "Copy Table" function replaces the contents of the target list object with the contents of the worksheet table. Therefore, the Procedure column needs to be set on the worksheet prior to performing the copy operation. The Excel "fill" feature may be used to quickly set the 8 cells in the Procedure column.

Although it is not necessary for this example, Excel's editing features can be used to specify the factor table levels. For example, the columns can be sorted in order to use the intelligent fill feature within Excel), to reorder levels, or remove certain levels.

Once the list is completed, copy the values in the table by selecting the "Copy Table" button. Then, open the DesignList object in the experiment, place the cursor in the first cell in the PrimeType column (i.e., to the right of the Nested column) and type Ctrl+V to paste the data from the clipboard to the List. Note that only the table values themselves are copied; the column headings are not. However, the column headings on the list object match the order of the columns in the Factor Table Wizard, so no additional editing of the list object is required.

⚠ **NOTE:** *For the remainder of the example, we return to the List entered using Method A. A finished version of Stage3-MethodA-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

Stage 3, Step 2: Set the weights
⚠ **NOTE:** *For the remainder of the example, we return to the List entered using Method A. A finished version of Stage3-MethodA-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

To change the relative frequency of trial types, change the weight of the rows. In this example, this is done by setting the values for the Weight attribute in the DesignList. Set the weights for the rows of ProbeType = "Word" to "1," and the weights for the ProbeType NonWord rows to "2," making the ratio of non-word to word stimuli 2 to 1.

ID	Weight	Procedure	Nested	ProbeType	PrimeType	Stimulus	CorrectResponse
1	1	TrialProc		Word	word	cat	1
2	1	TrialProc		Word	word	dog	1
3	2	TrialProc		Word	nonword	cat	1
4	2	TrialProc		Word	nonword	dog	1
5	1	TrialProc		NonWord	word	jop	2
6	1	TrialProc		NonWord	word	fuame	2
7	2	TrialProc		NonWord	nonword	jop	2
8	2	TrialProc		NonWord	nonword	fuame	2

Stage 3, Step 3: Set the Sampling Mode and Exit Condition
The sampling mode specifies of the order in which levels or conditions are run. Sampling modes include sequential, random (without replacement), random (with replacement), counterbalance, offset, and permutation. The default sampling mode is sequential presentation of items. This is useful for debugging to check each condition. Sequential sampling mode is also useful when setting up practice block conditions, but for most experimental trials, one of the other sampling modes is typically used.

To set the sampling mode, click the Properties button in the DesignList. In this case, we will select random without replacement and select two cycles of samples. On the Selection tab, set the Order to "random." On the Reset/Exit tab, reset the List after all samples have been run (12 exemplars; 4 Word, and 8 NonWord). Keep the Exit List option set to the first radio button, after some number of cycles, but change the value in the text box from 1 to 2. This will result in the List being exited after two cycles (for a total of 24 trials).

Click OK, the DesignList now displays a summary describing the sampling method, the number of samples, and the selection method. The list of 12 exemplars will be sampled randomly, without replacement, until all 12 exemplars have been chosen; all exemplars will then be returned to the pool of eligible exemplars and randomly selected again, one at a time, until all 12 have been chosen again. After the second group of 12 exemplars have been selected and presented, the list will exist.

To select other sampling methods, simply select the method name from the "Order" dropdown list box on the "Selection" tab of the List object. *Chapter 2: Designing Your Experiment, 2.1 Stage 1: Conceptualize the Core Experimental Procedure (Page 12)*, defines the various sampling methods and provides an illustration of how various conditions are selected with each method.

Stage 3, Step 4: Verify
Save the experiment as Stage3-LexicalDecision001.es2, generate, and run to verify that the new additions to the DesignList work as expected. There should now be 24 trials in random order with non-words occurring twice as frequently as words.

3.4 Stage 4: Add Blocks and Block Conditions
⚠ **NOTE:** *For the remainder of the example, we return to the List entered using Method A. A finished version of Stage3-MethodA-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

The goal of this stage is to add a block procedure to run the DesignList. This will enable multiple blocks to be run during the session. In this example, we will run one block of primes with a prime duration of 500 ms and a second block with a prime duration of 100 ms. To do this we will add a List at the block level, declare a new attribute (PrimeDuration) to set the duration of the Prime Display, and move the DesignList to operate in the Block procedure. We will also add two new displays to the session, one to welcome the participant and another to dismiss them. It is good practice to always include these type of displays in the experiment. Lastly, we describe two additional block manipulations that are easily implemented in E-Studio which enable you to implement a wide variety of advanced experimental designs. The steps that are required to implement the new block manipulation are shown in the table below:

| Stage 4: Add Blocks and Block Conditions |
| --- |
| 1) Verify setup of the Block Manipulation
   • Add a Block List Object
   • Add a Block Procedure
   • Add a Block Attribute
   • Move the trial stimuli from the Session level to the Block level |
| 2) Add Block Instructions |
| 3) Modify TrialProc to use PrimeDuration |
| 4) Add Introduction and Goodbye displays |
| 5) Special Notes: Multiple Methods to Divide a Design Between Levels |

Stage 4, Step 1: Verify setup of the Block Manipulation

Add a BlockList Object
We need to add a block level to the experiment. This will involve adding a BlockList (List object) and a BlockProc (Procedure object). Click the List object in the Toolbox, and drag a List to the Structure window as the first event in the SessionProc. Rename the new List object to BlockList.
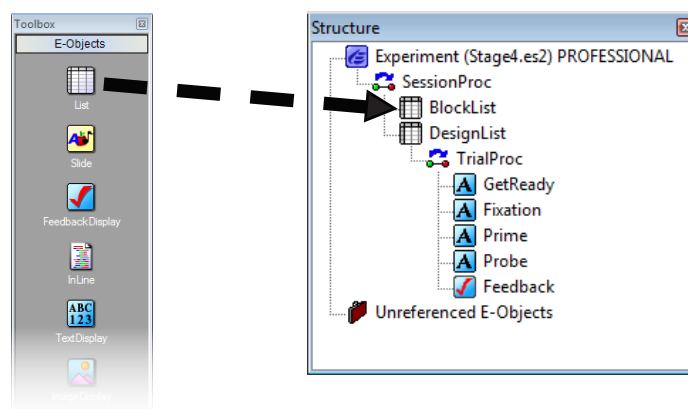
Add a Block Procedure

Double click the BlockList in the Structure window to open it in the Workspace. In the Procedure attribute column, type BlockProc to run the BlockProc from the BlockList. When prompted to create the BlockProc, answer "yes". When prompted to use the new procedure as the default value for newly created levels, answer "yes".
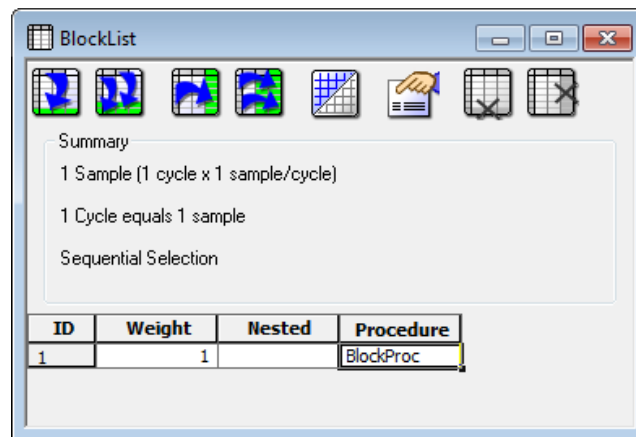

Add a Block Attribute

In the BlockList, add an attribute and rename it PrimeDuration. This attribute will be used to vary the duration of the Prime across blocks. Click the Add Level tool button to add one level. Complete the BlockList to define the levels of the PrimeDuration attribute as 500 and 1000 ms.
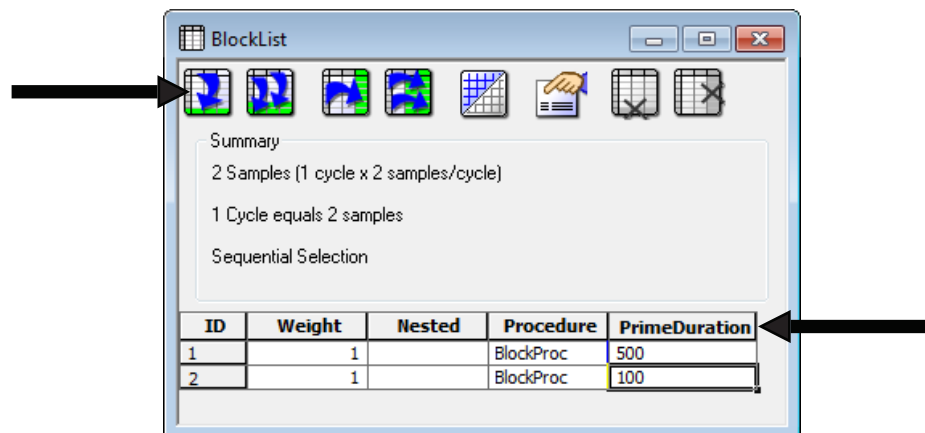
Click the Properties button in the BlockList and set the order of selection to "random" (Selection tab). The summary will appear as follows:
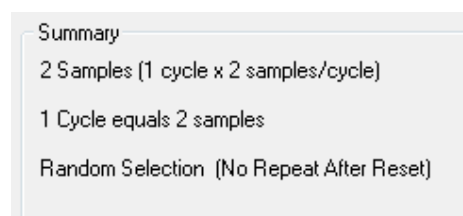
Move the Trial Stimuli from the Session to the Block Level

We need to move the DesignList one level deeper in the experiment. We now want the DesignList to occur in the BlockProc rather than the SessionProc, because the sampling of the trial stimuli as specified on the DesignList object needs to occur at the start of each new block, not once at the start of the experimental session. Specifically, we need to move the DesignList to the BlockProc, and then delete it from the SessionProc. Open the BlockProc in the Workspace. Click the DesignList object in the Structure window and drag it to the BlockProc timeline to run the DesignList from the block Procedure. After adding the DesignList to the block Procedure, delete the DesignList from the SessionProc.
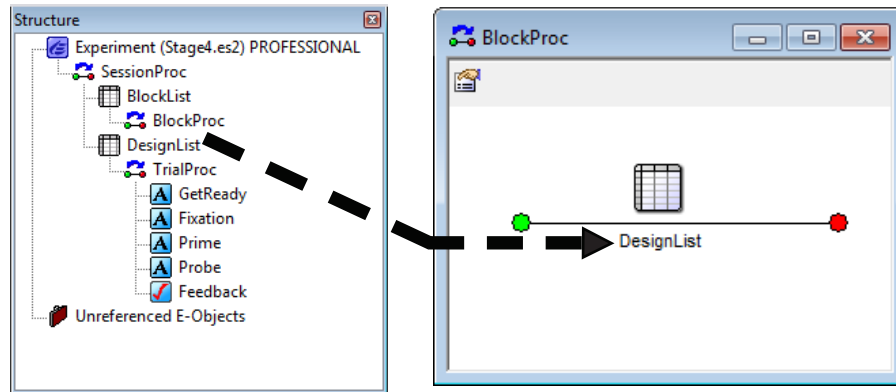
Stage 4, Step 2: Add Block Instructions

In the Toolbox, click the TextDisplay object icon, and drag a TextDisplay object to the BlockProc as the first event in the procedure. Rename the TextDisplay to BlockInstructions. Type the text that is shown in the BlockInstructions TextDisplay figure below as the text for the BlockInstructions object. In the instructions, the [PrimeDuration] attribute reference informs the participant what the delay will be for each block.
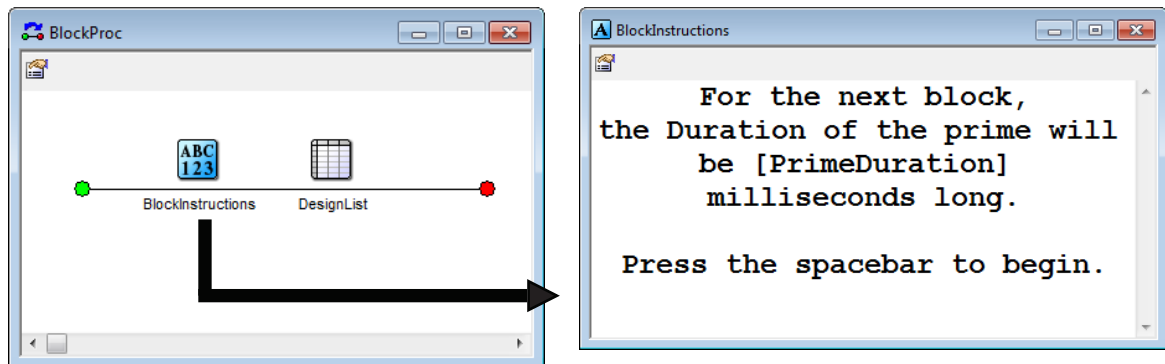

The participant is instructed to press the spacebar to begin the block. To specify the spacebar as an input key, make the following settings on the Duration/Input tab of the BlockInstructions properties 1) Set the Duration equal to "infinite" (i.e., wait until a response is entered), 2) Set the Input Mask to allow keyboard input, and 3) Set the "Allowable" field to the spacebar (i.e., {SPACE}). The End Action should be already set to "Terminate".
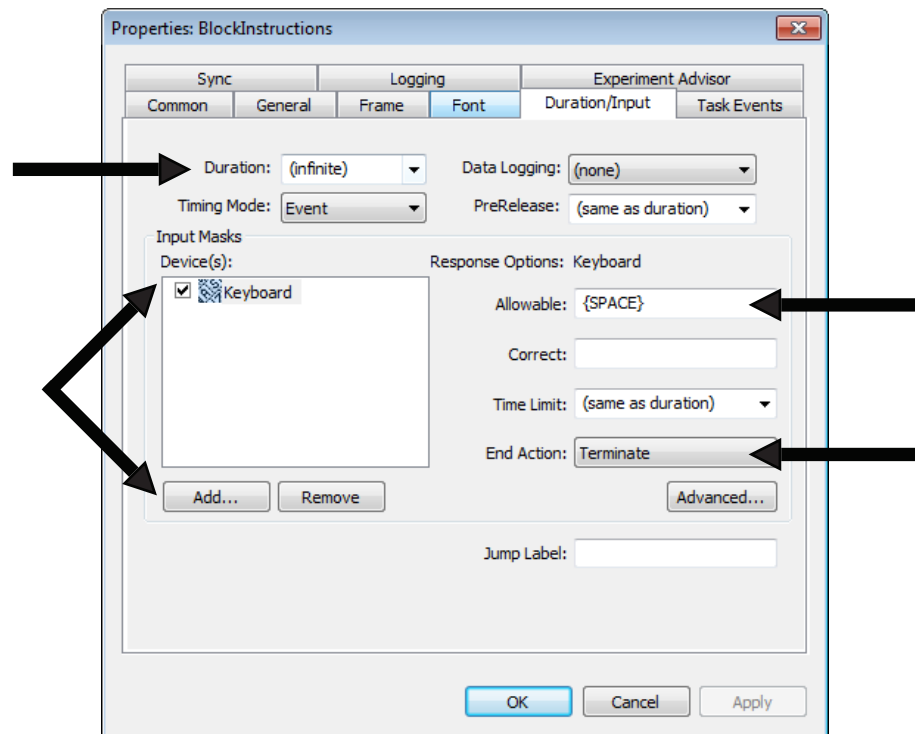
Stage 4, Step 3: Modify TrialProc to use PrimeDuration

To alter the duration of the Prime, set the Duration of the Prime Display to the value of the PrimeDuration attribute defined at the block level. To do so, double click the Prime object in the TrialProc to open it in the Workspace. Click the Property pages button to open the Property pages for the Prime object, and select the Duration/Input tab. Change the value in the Duration field from '1000' to the [PrimeDuration] attribute. This will vary the duration of the Prime Display across blocks based on the block level attribute.
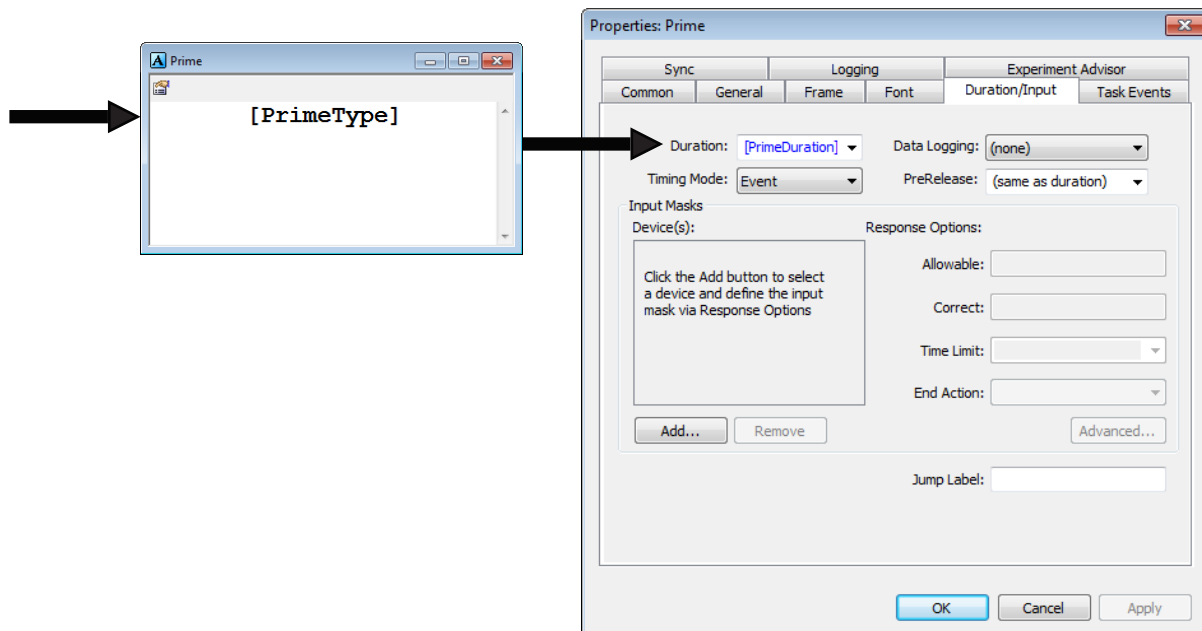
Stage 4, Step 4: Add Introduction and Goodbye to SessionProc

Add a TextDisplay as the first object in the SessionProc. Rename this object to "Introduction," and type a message to welcome the participant as the text for this object to display. Add a TextDisplay as the last object in the SessionProc. Rename this object "Goodbye," and type a dismissal message in the Text field of this object.
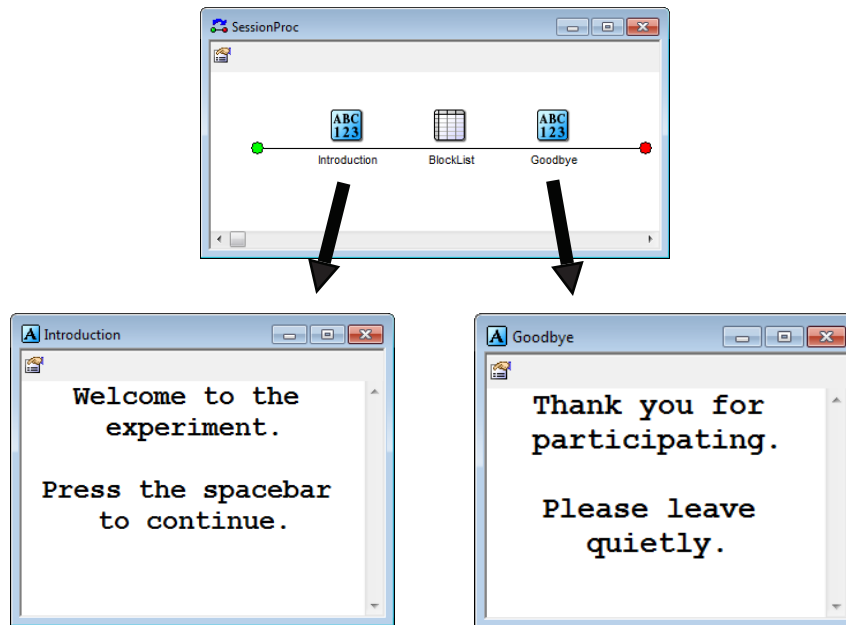
Like the BlockInstructions, the Introduction object instructs the participant to press the spacebar to continue. In the properties for the Introduction object, 1) Set the Duration equal to "infinite" to wait until a response is entered, 2) Set the Input Mask to allow keyboard input, 3) Set the Allowable field as the spacebar (i.e., {SPACE}), and 4) Accept the default value of End Action equal "Terminate" upon response from the participant. In the Properties for the Goodbye object, set the Duration to 5000. No input needs to be enabled for the Goodbye object; it will simply time-out at the end of the duration.

Stage 4, Step 5: Special Notes: Multiple Methods to Divide a Design between Levels

Multiple methods may be used to execute different procedures during different blocks. The Stage 4 manipulations to this point illustrate using a block level attribute to set a parameter of an object executed in the trial procedure (the PrimeDuration). In other words, attributes can be sampled at the block level, and the selected level of a block attribute can be "passed down" to affect aspects of the trial level displays, such as the display duration.

In addition, though, more advanced block manipulations are available which enable more complex designs to be implemented. Two common procedures are described in this section: passing pass a nested list and passing an experimental procedure. These advanced features provide great deal of flexibility, and they are briefly illustrated here.

Use of nested lists passed from the block level

⚠️ **NOTE:** *A finished version of Stage4-NestedBlockList-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

By passing nested list objects from a block List, the trial level Procedure can select stimuli from different sets of items for each block. For example, you could run one block of trials in which all of the words are from the Animal category, and a second block in which all of the words are from the Fruits category. This can be accomplished by using an attribute at the block level (e.g., ListType), which specifies the different lists of stimuli (e.g., AnimalList and FruitList). One of the defined levels of the block attribute is selected at the start of the block; then, this value is used to set which list of stimuli will be sampled

within the current block of trials. This is accomplished by passing the block attribute to the Nested column of the list at the trial level.

Continuing the example above, add a new attribute named the ListType attribute to the BlockList, and enter "AnimalList" and "FruitList" as the values for this attribute. This attribute identifies which list of words will be sampled for all of the trials within a block.
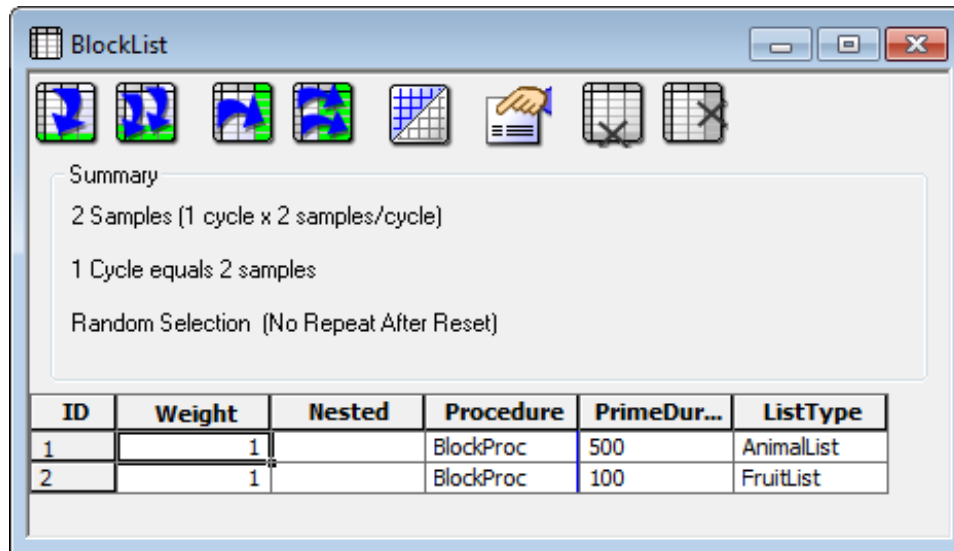

In the DesignList (i.e., the List at the trial level), enter a reference to the block-level ListType attribute (e.g. [ListType]) in the Nested column for each row that is a Word trial (i.e., ProbeType=Word). This allows the Word trial stimuli to be chosen from a single List, either the AnimalList or the FruitList, for all of the trials within a block. The list of stimulus words that is sampled for each block of trials will vary according to the ListType attribute.

The NonWord trial stimuli can also be placed onto a separate list. For the NonWord trials, all stimuli can be chosen from a single list of non-words. While this manipulation is not technically necessary, it provides a more consistent implementation (in that all of the probe strings are contained on separate lists, rather than have some probe strings the words) sampled from separate lists and other probe strings (the nonwords) sampled from the DesignList object. To change the manner in which the nonword strings are sampled, the Nested column for the NonWord trials (i.e., ProbeType=NonWord) needs to refer to a (not yet created) list of nonwords. Since there is only one list of nonwords, the value in the "Nested" list will be the actual List object name, and not a reference to an attribute that resolves to a list object name, as in the case of the Word trials. Enter "NonWordList" in the nested column for each level where ProbeType = NonWord; this should be row ID # 5-8. Answer 'y' when asked if you want to create the NonWordList object.

With this setup, the probe stimulus that is presented on each trial is no longer being sampled directly from the DesignList; it is sampled from one of the three lists of stimuli that are referenced on the "Nested" column on the DesignList. Therefore, the stimulus string that is currently entered on each row of the in the Stimulus attribute column needs to be replaced with the name of an attribute (e.g., ListStim) that is resolved using values from the nested list. Replace the existing strings in the Stimulus column with "[ListStim]", as shown below.
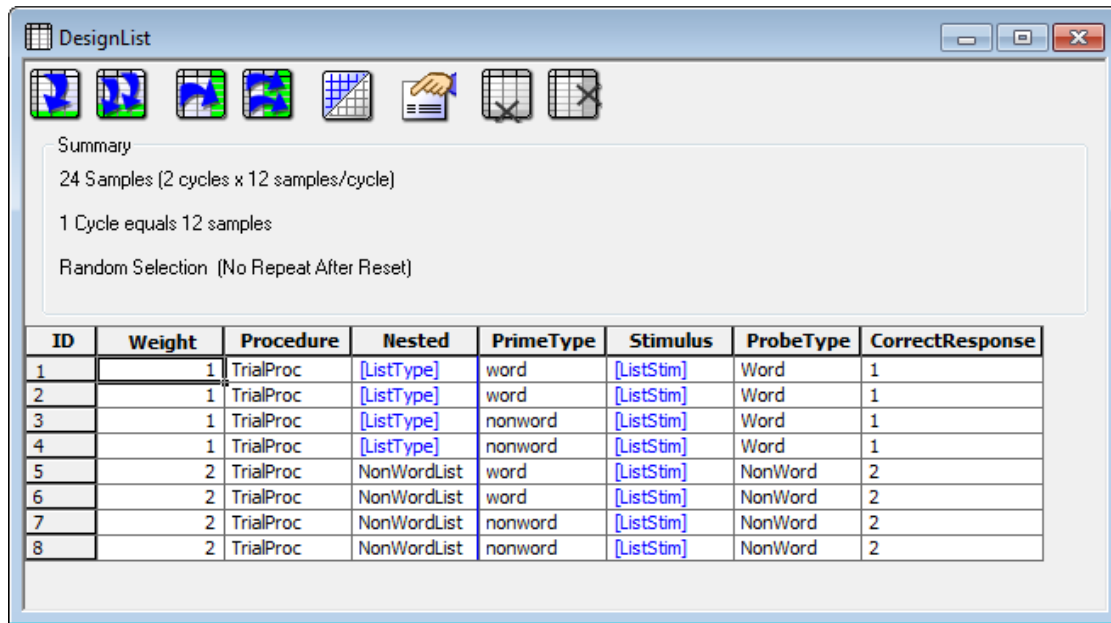
The next step is to create the two new List objects (i.e., AnimalList, FruitList; E-Studio created the NonWordList if you answered yes to the "create object" prompt) and specify the new attribute and levels for the 3 nested lists to which the DesignList now refers (AnimalList, FruitList, NonWordList). Each list would necessarily include the ListStim attribute, which would define the text strings to be used as the stimuli. Create the AnimalList, and FruitList in the Unreferenced E-Objects folder.
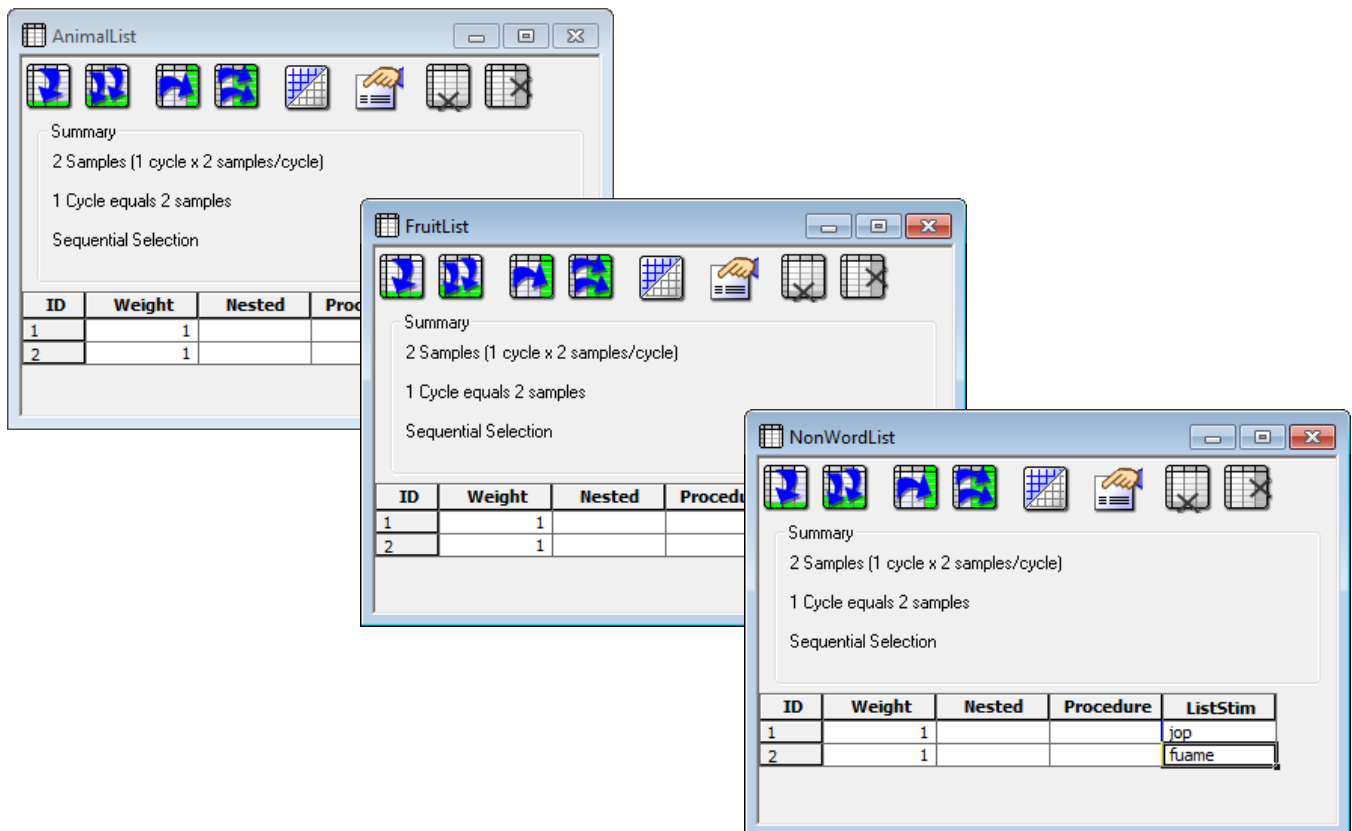
At runtime, the ListType attribute value is resolved to determine which List to use for the block of trials, and the ListStim attribute value is resolved from the List identified by ListType.

It is worth mentioning that the change in the structure of the experiment that results from nesting. Nested lists are placed in the Structure window immediately subordinate to the List calling them. Thus, in the Structure window below, the NonWordList is placed in the structure below the DesignList.
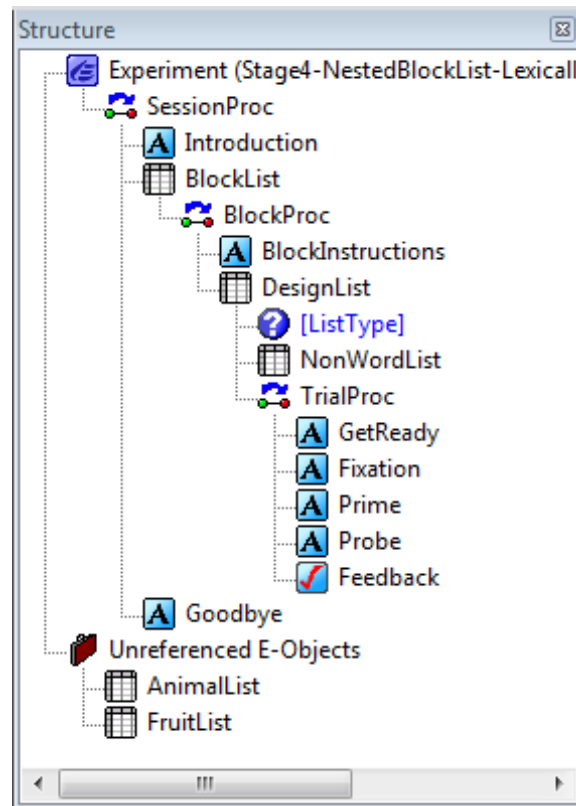
However, because the ListType attribute is being used to vary the selection of words from two List objects, and the value of ListType is resolved only at runtime, E-Studio is unable to place the AnimalList and the FruitList objects in the structure. The ListType attribute is placed in brackets below the DesignList, and is preceded by a to indicate that the nested list is varying. The FruitList and AnimalList objects are placed in the Unreferenced E-Objects folder.

Colon Syntax – Sampling multiple stimuli per trial using nested lists
When a design requires multiple stimuli (rows) to be sampled from a List during a single trial, a colon syntax is used to indicate the number of exemplars to be sampled. The colon syntax places a colon and a number after the attribute name within the bracket notation (e.g., [AttrName:1]). When the colon syntax is not used, a single exemplar is sampled by default (i.e., not using the colon syntax is equivalent to [AttrName:0]). When a colon and a number are placed after the attribute name within the bracket notation, the number used indicates the number of stimuli being sampled during the same trial in addition to the default. For example, the use of [AttrName:2] would result in the sampling of three exemplars from the List during the same trial: the default exemplar plus two additional exemplars.

For the example experiment, perhaps the task might be modified to present two words or non-word strings at the same time instead of just one. If the stimuli are to be sampled from the same List, the colon syntax must be used. By changing the Stimulus attribute values to use the colon syntax (Figure 1 below), two stimuli will be chosen per trial from the List named in the Nested column, and assigned as the value of the Stimulus attribute (i.e., two stimuli will be displayed on the same line). Alternatively, a second attribute could be created and used to display the second exemplar (Figure 2), which would allow for more flexibility in the display (e.g., displaying Stimulus2 on the line below Stimulus).

Stimulus
[ListStim] [ListStim: 1]
[ListStim] [ListStim: 1]
[ListStim] [ListStim: 1]
[ListStim] [ListStim: 1]
[ListStim] [ListStim: 1]
[ListStim] [ListStim: 1]
[ListStim] [ListStim: 1]
[ListStim] [ListStim: 1]

Stimulus	Stimulus2
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]
[ListStim]	[ListStim: 1]

Figure 1. Two stimuli assigned to a single attribute value.

Figure 2. Two stimuli assigned to separate attributes.

Use of block Procedure to pass in the trial Procedure

⚠ **NOTE:** *A finished version of Stage4-ChangeTrialProc-LexicalDecision001.es2 is located in the C:\My Experiments\Tutorials\Using E-Studio Stages folder.*

By passing different values to the Procedure attribute in a block list, you can run different trial Procedures during different blocks. For example, assume we wanted to run a PrimeTrialProc (as was developed as TrialProc in Stage 3) and add a MaskTrialProc, in which the Prime word is presented but masked after 50 ms. The running of two Procedures could be accomplished by calling two different Procedure objects from the DesignList object. However, if the Procedure is to be randomized by block, the BlockList must be used to determine the Procedure run by the DesignList. This can be accomplished by passing the Procedure as an attribute from the BlockList to the DesignList. Open the BlockList and add an attribute called TrialProcName. Enter PrimeTrialProc and MaskTrialProc as the values for this new attribute.

BlockList

Summary

2 Samples (1 cycle x 2 samples/cycle)

1 Cycle equals 2 samples

Random Selection (No Repeat After Reset)

ID	Weight	Nested	Procedure	PrimeDur...	TrialProcName
1	1		BlockProc	500	PrimeTrialProc
2	1		BlockProc	100	MaskTrialProc

Create the new mask trial procedure

Rename TrialProc to PrimeTrialProc. This is the only change needed to the Prime trial procedure. Notice that the renaming of TrialProc to PrimeTrialProc results in the updating of all references to TrialProc (e.g., in the Structure window and in the DesignList).

Next, create the new trial procedure that presents the masking trials as follows:
    1) Create a new procedure object in the Unreferenced objects folder. Name this new procedure "MaskTrialProc".
   2) Open the MaskTrialProc in the workspace.

The MaskTrialProc will present all of the objects that are called in the PrimeTrialProc, as well as presenting a new mask display prior to the Probe Display. We can recycle the PrimeTrialProc objects by re-using them on the new MaskTrialProc. Start with the first object in the procedure, the GetReady TextDisplay.
    3) Click once on the GetReadyDisplay that is shown in the Structure window as part of the PrimeTrialProc, and then drag the object to the MaskTrialProc in the workspace.
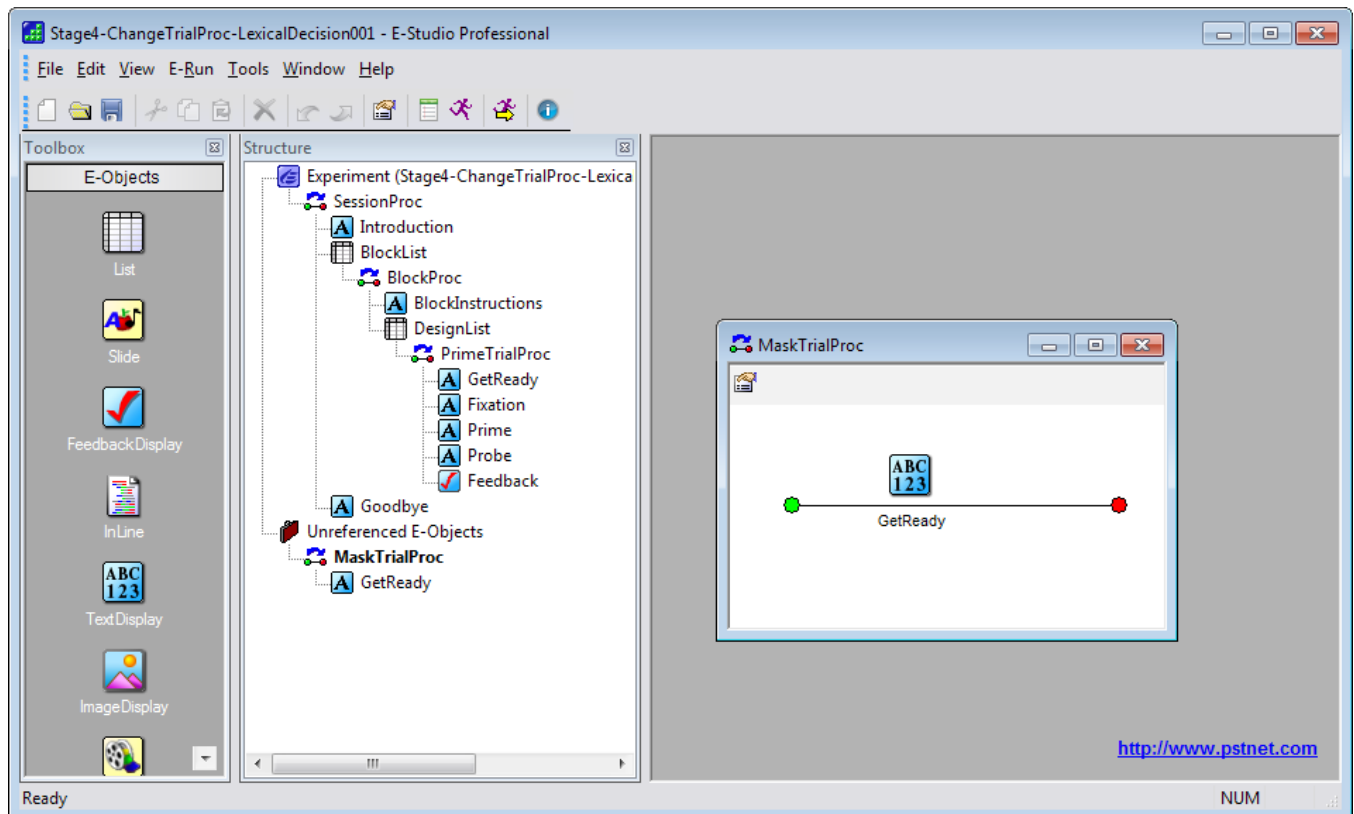
Notice that the GetReady display now appears in both of the trial procedure objects.
    4) Repeat with each of the remaining objects on the PrimeTrialProc. At this point, there is no difference between the PrimeTrialProc and the MaskTrialProc procedures; they each present the same objects in the same order. We still need to add the new masking display to the MaskTrialProc, but this step will be performed later.
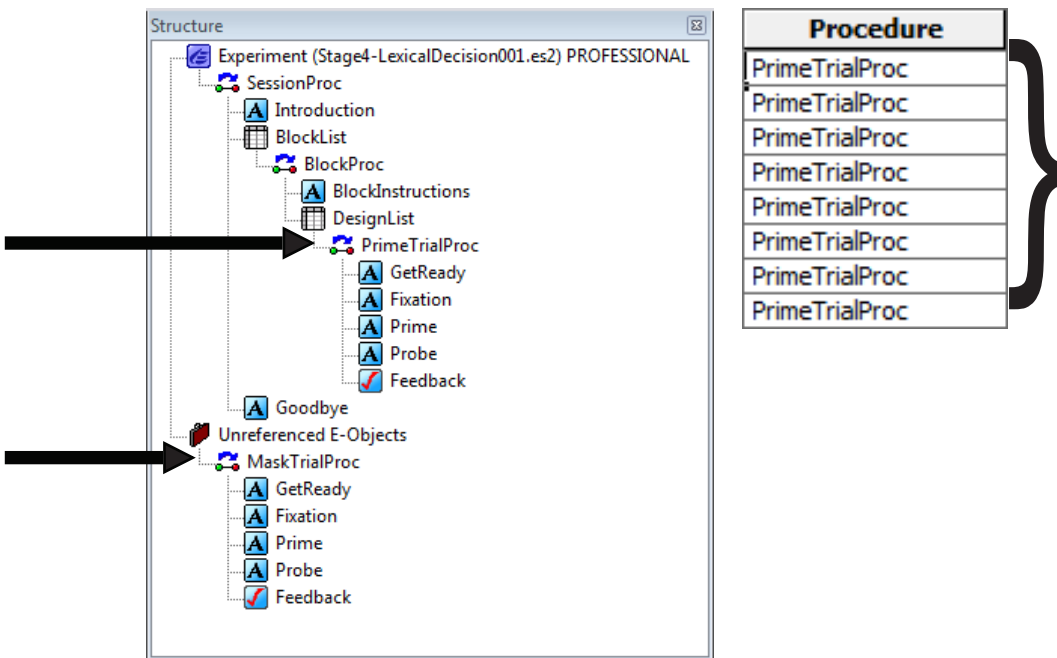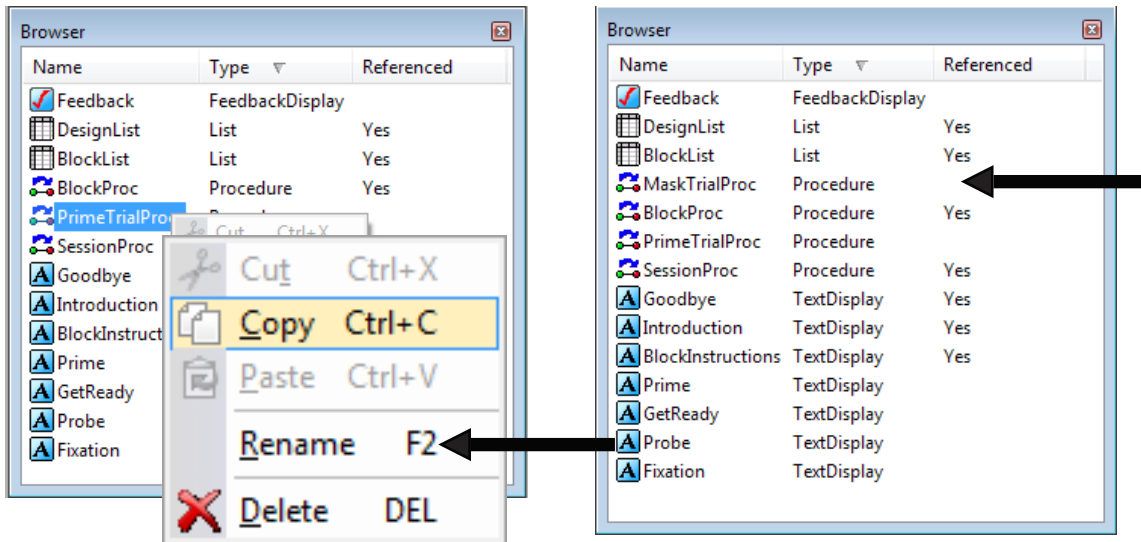
Vary the procedure that the DesignList runs
To vary the procedure run by DesignList, modify the Procedure column in the DesignList to refer to the TrialProcName attribute set by the BlockList.
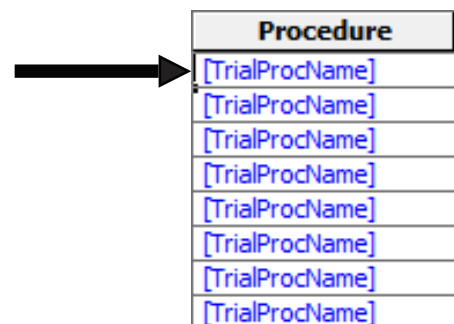
The results of this modification are displayed in the Structure window below. Since the DesignList refers to the [TrialProcName] attribute as the value in the Procedure column, the Structure window shows a "?" as the procedure called by the DesignList, and the MaskTrialProc and PrimeTrialProc reside in the Unreferenced E-Objects folder. At runtime, the value for the TrialProcName attribute is resolved, and the appropriate Procedure is executed.

Complete the edits to the MaskTrialProc

Lastly, notice that in the MaskTrialProc shown above, there is an additional object to present the Mask. To complete the MaskTrialProc as in the view above, drag a TextDisplay object to the MaskTrialProc following the Prime. Rename the new TextDisplay to "Mask", set the Mask Duration property to 50 ms, and the Height property to 25% (i.e., to match the display area settings for the Prime Display).

3.5 Stage 5: Add Practice Block

⚠ **NOTE:** *This section builds on the experiment created during the previous section. A finished version of Stage4-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

The goal of Stage 5 is to add a block of practice trials prior to the experimental trials. The practice block will be run repeatedly until the desired block accuracy of 80% is obtained. This involves adding a PracticeBlockList, a PracticeMode attribute to track the practice mode, some user-entered script to score the accuracy, and repeating the practice block if the accuracy criterion is not met.

There are four steps involved in adding a practice block, as shown in the following table:

Stage 5: Add Practice Block

1) Define the Practice Block
   • Copy the BlockList object
   • Call the new PracticeBlockProc from the SessionProc
   • Edit the PracticeBlockList
   • Recycle the BlockProc

2) Add the Practice Block Variable
3) Define the Practice Block Criterion
4) Test

Stage 5, Step 1: Define the Practice Block

Copy the BlockList object

We want to have a PracticeBlockList very similar to the BlockList. By duplicating the BlockList we have a copy that can be modified without affecting the BlockList. The Browser window provides an easy way to copy objects. In the View menu, select the Browser option to display the Browser window. In the Browser, select the BlockList object and right-click to display a context menu. From the context menu, select Copy to make a copy of the BlockList. Right-click again to re-display the context menu, and paste the copy (which will be named BlockList1 by default) into the Browser. Select BlockList1 and right-click once again to display the context menu. Rename BlockList1 to PracticeBlockList.
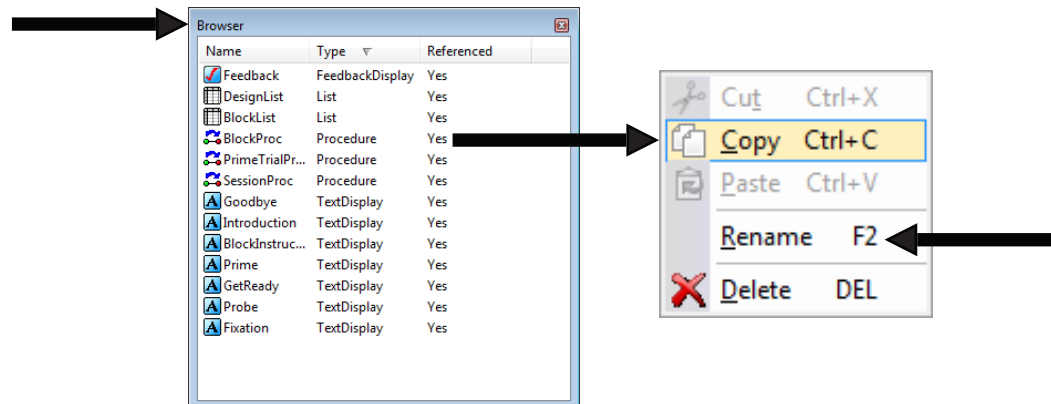
The PracticeBlockList object is created in the Unreferenced E-Objects folder in the Structure window. When E-Studio copies an object, it does two things: makes a copy of the target object (obviously), and the makes a copy of all of the sub-objects on the target object. In this case, the original BlockList object referenced the BlockProc procedure object. When E-Studio copies the list object, it creates a copy of the procedure object as well (BlockProc1). Then, when E-Studio creates the copy of the procedure object, it recognizes that the procedure references a list object, DesignList, and E-Studio makes a copy of the DesignList object (DesignList1). This process continues as E-Studio navigates through all of the sub-objects that are ultimately called by the original BlockList (e.g. the TrialProc and all of it's sub-objects.), making copies of all.

For this particular example, we do not need all of these copied sub-objects, and in fact want to recycle all of the original objects. (There are certainly other situations where you want to work with the copies.) Therefore, the next step is to delete the following objects:

Object Name	Object Type
BlockInstructions1	TextDisplay
BlockProc1	Procedure
DesignList1	List
Feedback1	Feedback
Fixation1	TextDisplay
GetReady1	TextDisplay
Prime1	TextDisplay
PrimeTrialProc1	Procedure
Probe1	TextDisplay

You can quickly and easily delete objects from the Browser window by clicking on the object once to select it, right clicking to bring up a context menu, and selected "DELETE". Answer 'y' when prompted to confirm the deletion.

Call the new PracticeBlockProc from the SessionProc
Open the SessionProc in the Workspace and drag the PracticeBlockList object to the SessionProc following the Introduction.
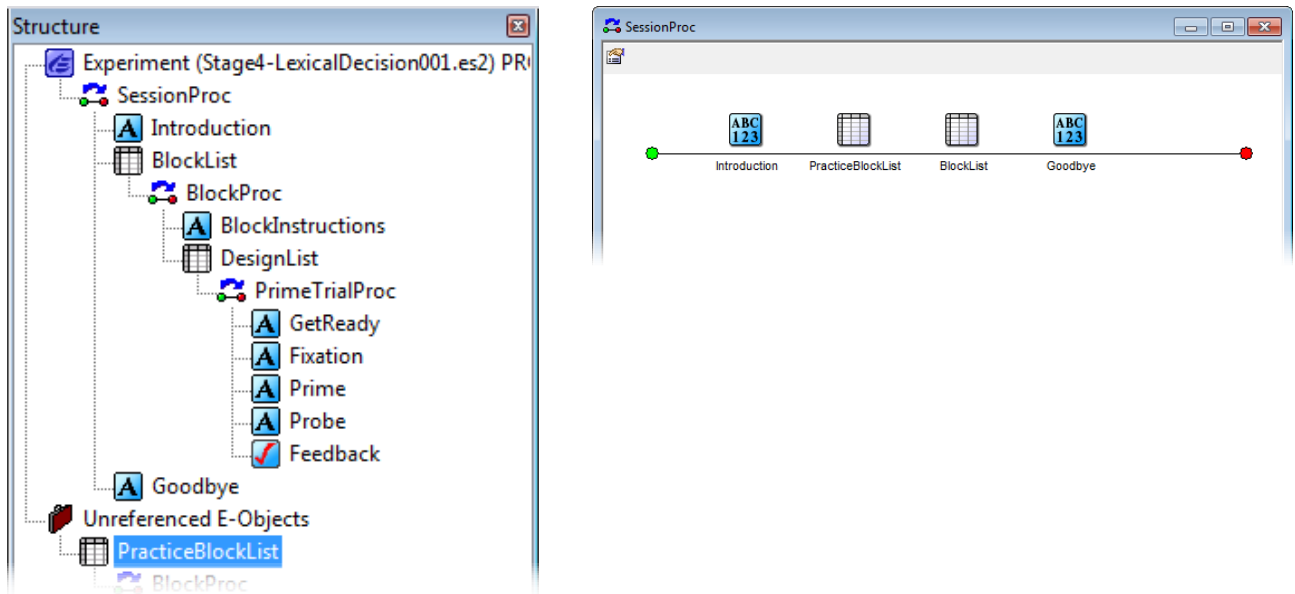
Edit the PracticeBlockList
Remove one level. The PracticeBlockList must be modified to be appropriate for the practice trials. The PracticeBlockList should only present one block of practice trials. (We'll add the manipulation whereby practice is repeated if the minimum accuracy criterion is not met in step 3 below). Therefore, the first of the two levels needs to be deleted. In the PracticeBlockList, select the first row by clicking on the first row in the ID column. Click the Remove Levels tool button to delete the selected level.
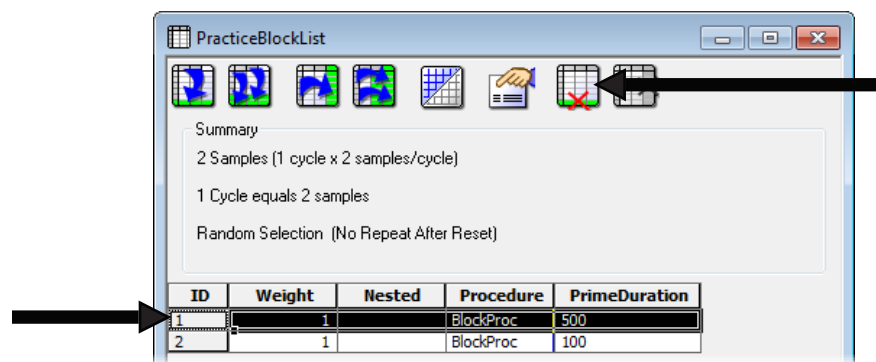

Recycle the BlockProc
When you created the PracticeBlockList object in the Browser, E-Studio also copied the BlockProc procedure to a new object named BlockProc1 and then set the value of the Procedure field on the copied list object to BlockProc1. For our purposes here, we actually want to call the same block procedure as is used in the original BlockList object, and further, the unneeded BlockProc1 object was deleted above. Therefore, the Procedure field on the PracticeBlockList object needs to be edited to specify BlockProc as the Procedure, and not BlockProc1.

Stage 5, Step 2: Add the Practice Block Variable
In order to easily analyze the practice data separately from the main data, add a variable that codes the information as practice or experimental trials.

Open the PracticeBlockList and the BlockList in the Workspace. Add an attribute named PracticeMode to each List object. Set the value for PracticeMode to "practice" for the PracticeBlockList, and to "real" for the BlockList.
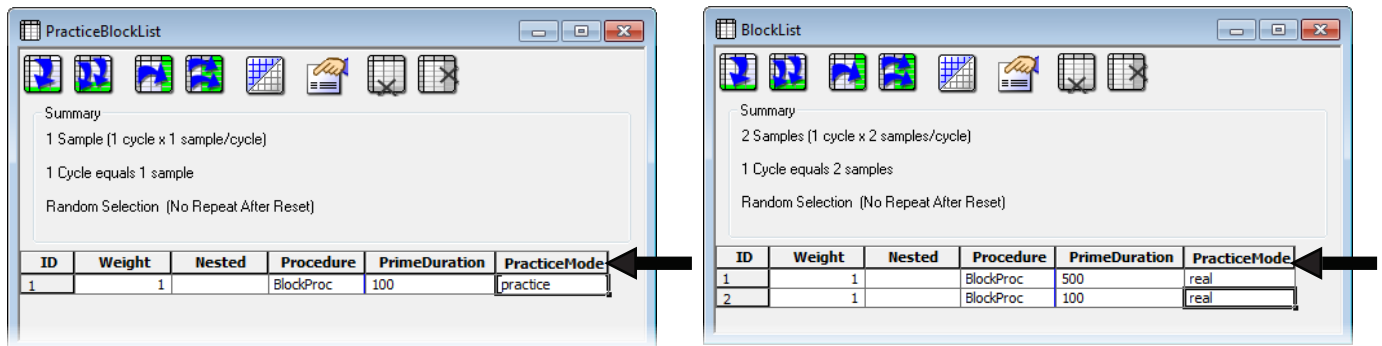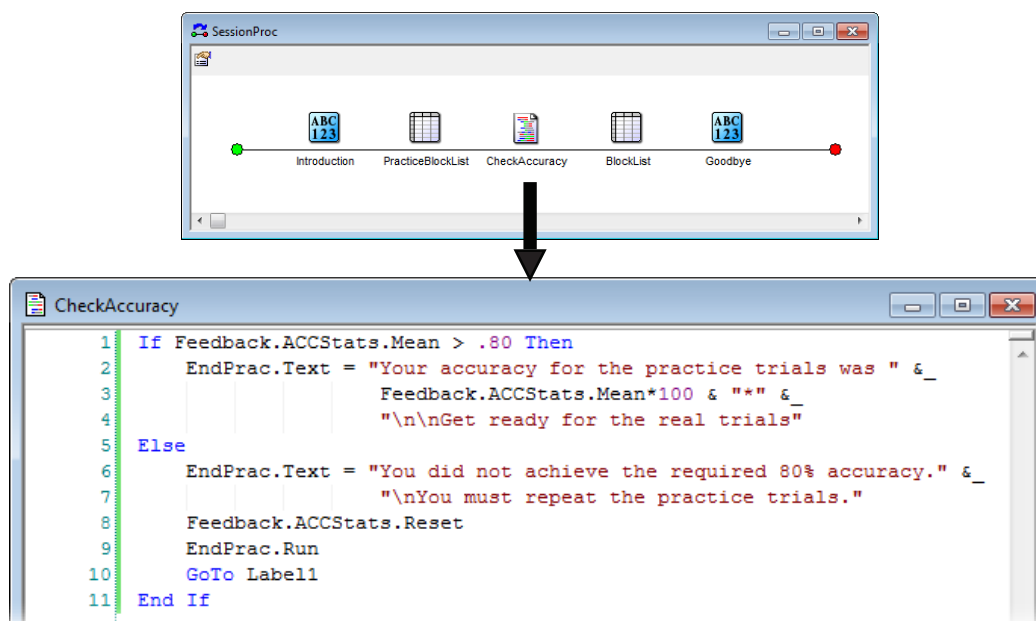
Stage 5, Step 3: Define the Practice Block Criterion

In order to repeat the practice block of trials until the desired accuracy level is achieved, we examine the accuracy after the PracBlockList terminates, and either continue with the real trials or jump back to run the PracBlockList again. We will need to add a few lines of script to the experiment to monitor the accuracy and determine when the accuracy is sufficient to allow termination of the practice block. How to write E-Basic script is detailed in Chapter 5: Using E-Basic. Here we will just provide an example of how to write some script for checking the accuracy during a block of trials.

Add InLine to check accuracy

To enter the code script to check accuracy, add an InLine object to the SessionProc following the PracticeBlockList. Rename this InLine object CheckAccuracy, and double click it to open it in the Workspace. Insert the text below into the CheckAccuracy object by typing directly in the object.



```
1   If Feedback.ACCStats.Mean > .80 Then
2       EndPrac.Text = "Your accuracy for the practice trials was " &_
3                      Feedback.ACCStats.Mean*100 & "*" &_
4                      "\n\nGet ready for the real trials"
5   Else
6       EndPrac.Text = "You did not achieve the required 80% accuracy." &_
7                      "\nYou must repeat the practice trials."
8       Feedback.ACCStats.Reset
9       EndPrac.Run
10      GoTo Label1
11  End If
```

The script above examines the value of Feedback.ACCStats.Mean (calculated automatically by the Feedback object) to determine if the participant achieved 80% accuracy. If this value is greater than 80%, the practice trials terminate and the experiment continues with the real trials (i.e., the next object in the SessionProc is the BlockList). If the participant did not achieve greater than 80% accuracy[2] for the practice trials, the practice trials are repeated using the Goto command to jump to Label1. Using this method, the practice trials will continue until the Criterion for continuation is met. The Feedback.ACCStats.Reset command is used to reset the Feedback accuracy statistics to zero before rerunning the block. The InLine is also setting the text to display using an object called EndPrac. In the next few steps, both the Label1 and EndPrac objects will be created.

[2] Note that, with 24 trials, the participant cannot perform at 80%; they can either get 19/24 correct, or 79.2%, or 20/24 correct, or 83.3%. However, the greater than or equal to operator is supported in E-Basic (along with less than or equal to, equal to, etc.); see the E-Basic help in E-Studio, comparison operators, for details.

Place Label1 prior to the PracticeBlockList
Click and drag the Label object icon to create a Label on the SessionProc just prior to the PracBlockList. Label1 will mark the location in the SessionProc to which the Goto command refers. If the required accuracy level is not achieved during the practice trials, the execution of the experiment will jump back to Label1 and rerun the PracticeBlockList.
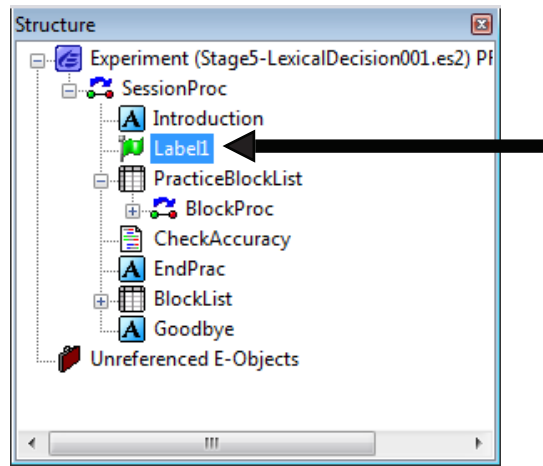
Add the EndPrac object to the SessionProc
It is important to inform the participant whether they have met the accuracy criterion. We do this by adding the EndPrac object to display the text determined by the preceding CheckAccuracy Inline object.

Add a TextDisplay object to the SessionProc following the CheckAccuracy object, and rename it EndPrac. The text for this object to display will be assigned at runtime via the CheckAccuracy InLine object, so no text need be entered for this object. Set the Duration property for the EndPrac object to 3000 ms. The EndPrac object is not necessary for the process of repeating the practice trials, but helps to make the experiment execution more understandable to the participant.
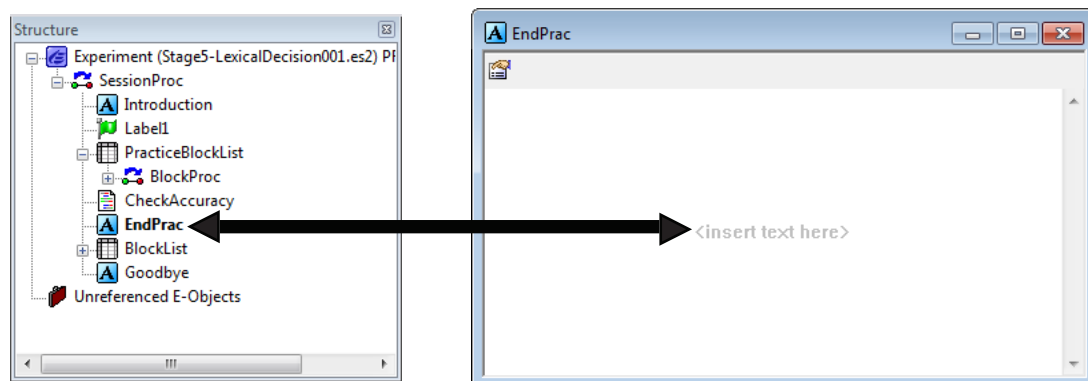
Stage 5, Step 4: Test
Save the experiment as Stage5-LexicalDecision001.es2, generate, and run to verify that the practice block. There should now be a practice block of 24 trials in random order. If less than 80% of the practice trials are answered correctly, then the practice block repeats. Examine the data file, and confirm that you can distinguish the practice block from the experimental blocks via the new block independent variable PracticeMode.

3.6 Stage 6: Special Functions – Setting Timing Modes, and Graceful Abort

⚠️ **NOTE:** *This section builds on the experiment created during Stage 4. A finished version of Stage4-LexicalDecision001.es2 is located in the …My Experiments\Tutorials\Using E-Studio Stages folder.*

At this point there should basically be a functioning experiment. However, additional checks need to be made to verify the timing precision and perhaps to structure the experiment to allow experimenter termination of the experiment.

| Stage 6: Special Functions – Setting Timing Modes, and Graceful Abort |
| --- |
| 1) Checking timing modes |
| 2) Providing early graceful abort of an experiment |

Stage 6, Step 1: Checking timing modes

Chapter 4: Critical Timing in this manual details methods to specify and check timing. There are two primary timing modes that each object can use (more modes are detailed in *Chapter 4: Critical Timing*).

Event mode – maintain the duration for start to end of duration to be as specified. If there are delays (e.g., due to waiting for the screen refresh), lengthen the inter-stimulus interval to compensate for any delays. **Cumulative mode** – maintain the duration from the start of one stimulus to the start of the next stimulus to be the specified duration. If there are any delays (e.g., waiting for the screen refresh), shorten the inter-stimulus interval so the second stimulus occurs at the requested time in relation to the intended display of the first stimulus. This maintains the interstimulus interval.

In this experiment, the timing mode specified on the Duration/Input tab for all stimulus display objects should use Event mode with a the PreRelease set to (same as duration). The Duration tab from the Prime object is displayed below. Note the Data Logging (Time Audit Only), Timing Mode (Event), and PreRelease (0 ms) fields. These fields must be checked for each of the critical objects in the trial procedure.
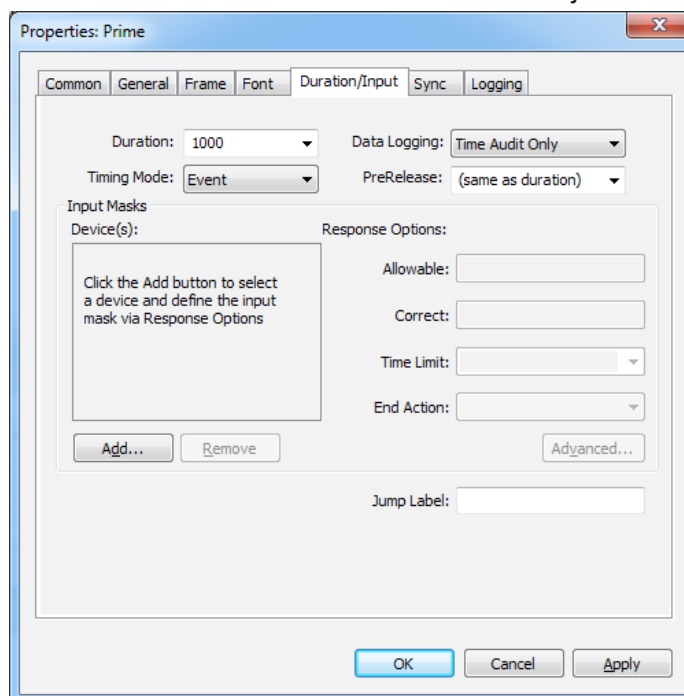
The "Time Audit Only" option in the Data Logging field logs the duration information for later analysis (i.e., OnsetTime, OnsetDelay, DurationError). The Time Audit Only option should be used on critical displays without responses, such as the Fixation and Prime Displays. Objects collecting a response (e.g., Probe) should set Data Logging to "Standard" in order to collect the response data in addition to the duration related data.

Stage 6, Step 2: Providing early graceful abort of an experiment

There is one elaboration to consider adding to the existing experiment to allow for a more graceful, early termination of the experiment. When running an experiment, one normally runs from the start to the finish of the experiment with the automatic selection of trials. However, E-Prime 2.0 permits an experiment to be aborted prior to the normal termination of trials by pressing the Ctrl+Alt+Shift keys simultaneously. This method is fine for debugging and aborts the experiment immediately. However, when the Ctrl+Alt+Shift method is used to abort an experiment, no data file is created. Therefore, any data for previously presented trials and blocks is not saved. This should not create problems when the escape sequence is used as intended, e.g. as a debugging tool during experiment development.

E-Prime 2.0 provides another method for putting in script to check to see if a different, special key sequence has been entered. If so, a List object can be terminated at the end of a trial or block, and all of the accumulated data collected to the current point of the experiment can be saved. This alternative method uses a special function, GetUserBreakState, which returns a "1" if the Ctrl+Shift keys have been set, and "0" if not. After the user selects Ctrl+Shift, then the value of GetUserBreakState is set to 1 until the experiment explicitly resets the state to zero with the following script:

SetUserBreakState 0

To check if the user has pressed the Ctrl+Shift combination, we add an Inline object at the end of the TrialProc and check the state of GetUserBreakState. If it is "1", meaning the user has pressed the abort experiment early key sequence, then the DesignList that is running and driving the current set of trials is terminated using the DesignList.Terminate ```command. This results in the currently block of trials being completed. Experiment control moves to the parent object that was called the DesignList, the BlockList object. The UserBreakState is then reset to zero to run the next block.
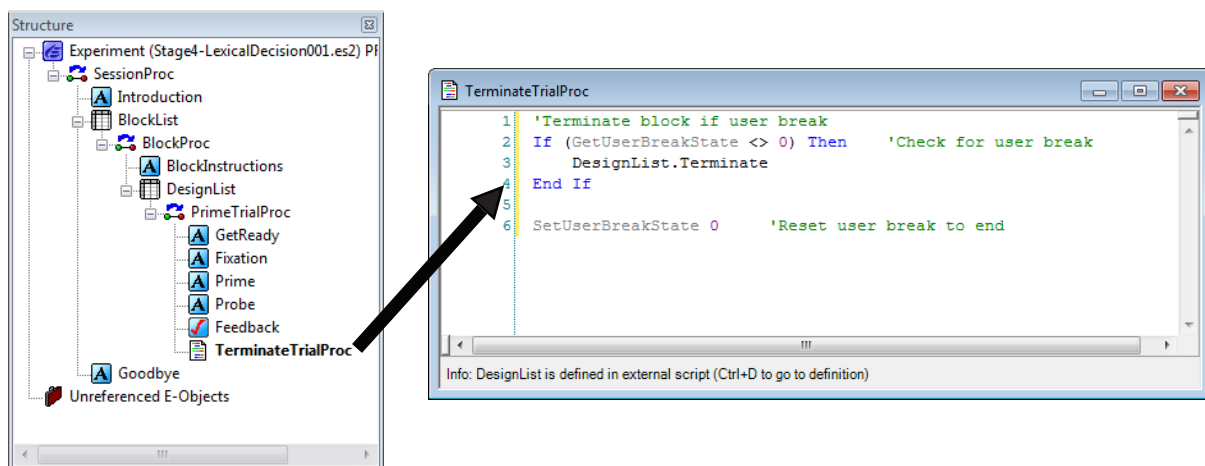
To terminate the BlockList, the same procedure would be used with a BlockList.Terminate command. If you are using E-Prime 2.0 Professional see the KB 2634 - FEATURE: Conditional Exit (Graceful Abort).

3.7 Stage 7: Testing the Experiment

⚠️**NOTE:** *This section builds on the experiment created during the previous section. A finished version of Stage6-LexicalDecision001.es2 is located in the ...My Experiments\Tutorials\Using E-Studio Stages folder.*

Stage 7: Testing the Experiment
1) Run Experiment to Verify Correct and Error Responses
2) Checking Scoring and Data Collection
3) Checking Timing Accuracy

4) Running Pilot Participants
  • Verifying the understanding of instructions
  • Checking the data

Once the experiment is "running" without error, a thorough process of testing must take place to reduce the threat of problems during data collection that may lead to incomplete or lost data.

Stage 7, Step 1: Run experiment to verify correct and error responses

Participants cannot be expected to respond correctly on every trial, or to respond correctly on any trial for that matter. The experiment should account for cases in which the participant responds correctly, responds incorrectly, presses keys that are not valid response keys, or fails to respond at all.

Stage 7, Step 2: Checking scoring and data collection

During the process of testing correct and error responses, the scoring process should be verified as well. The correct feedback should appear for every possible scenario, including correct responses, incorrect responses, and non-responses. It is a good practice to run through a block of trials, keeping track with a paper list of what happened during the run, and to verify that the paper list matches with the data in E-DataAid. Most likely, it is not necessary to run a large number of trials in order to be convinced that the data is being logged accurately. Five to ten stimulus-response trials may suffice.

• Examine the data file in E-DataAid to verify that the data is logging correctly at all levels of the experiment.

• At the lowest level, the responses and reaction times should be logged and scored correctly, and should be within the expected range of values.

• Verify that the correct number of trials and blocks are being run, and that the session level information is logging correctly (i.e., subject number, session number, etc.).

• Determine that the data file reflects the correct number of observations for each cell in the design. For example, if a 2x3 design is running 5 repetitions of each cell, the data file should display 30 trials including 5 trials for each of the 6 conditions.

• Finally, and most importantly, double check that all of the measures necessary for the analysis are being logged.

Because E-DataAid offers filtering capabilities, it is recommended that the user err on the side of logging too much information rather than failing to log some crucial variable.

Stage 7, Step 3: Checking timing accuracy

The logging options within E-Prime 2.0 allow for detailed monitoring of the timing of the events within an experiment. It is important to log, examine and monitor the timing information for experimental events (see *Chapter 4: Critical Timing*). On the Logging tab for each object, the Time Audit and Time Audit (Extended) variables provide the means to verify the launch-time, display-time, finish-time, and possible delay or error associated with each object.

Stage 7, Step 4: Running pilot participants

Usually, the person programming the experiment is too familiar with the instructions or the task to be able to pick up errors or inconsistencies in the experiment. It is important to run pilot participants to iron out the wrinkles in the experiment prior to actual data collection.

Verifying the understanding of instructions

For the participant, the instructions may not be as clear as they were for the experimenter, who certainly has more information concerning the purpose of the experiment. During the running of pilot participants, patterns of responding with inappropriate keys, low accuracy levels, or long reaction times may be indicative that the participants did not understand the task. Or, perhaps not enough instruction was supplied (e.g., often experiments are run without written instructions, and the experimenter verbally communicates the task to the participant).

It is generally a good idea to present participants with illustrations of the sequence of displays they will see on the computer and describe the task verbally before running it on the computer. Participants seem to be more likely to ask questions when presented with a scenario in paper form. Put test questions into the instructions (e.g., "Now if you saw this sequence of stimuli, how would you respond?").

Poor instructions result in irritated participants and inaccurate responses. Pilot test the instructions, being sure to run someone who does not have experience with the experiment. Clear instructions will not only add to the consistency of the experiment, but will serve as a record of the method used within the experiment when the experiment is passed on to a colleague or a student.

Checking the data

After each pilot participant is run, the individual data files should be reviewed for accuracy prior to merging them into a master data file. Monitoring of individual data files will help to eliminate potential conflicts during merge operations.

3.8 Stage 8: Running the Experiment

Once experiment development is completed and you have thoroughly tested the experiment and examined the data file to determine that the necessary information is logging appropriately, data collection is ready to begin. With a participant-ready experiment, E-Studio is no longer needed to launch the E-Run application.

Stage 8: Running the Experiment
1) Running Participants
2) Running on Multiple Machines

Stage 8, Step 1: Running Participants

During experiment development, the E-Basic Script (.ebs2) file is generated from within E-Studio via the Generate button, and then launched from within E-Studio by selecting the Run button. Once the experiment development and testing is complete, however, the E-Basic script file does not change and therefore does not need to be re-generated. There are three options for running a generated E-Basic Script file:

• Either double- click on the .ebs2 file name or right—click on the .ebs2 file name and select "Run ". The familiar E-Run prompt for startup information appears, unless the Experiment Object properties are configured to assign default values to all startup variables.

• From the Start menu, choose Programs-E-Prime 2.0 to display the E-Prime 2.0 menu, and then select E-Run.
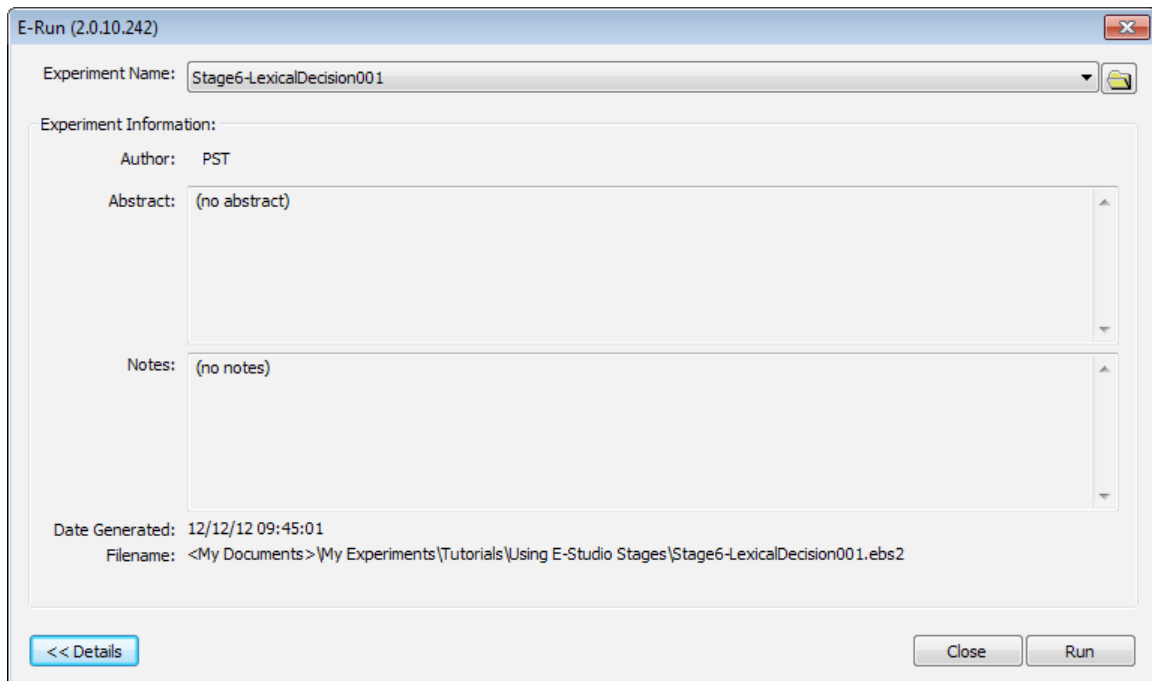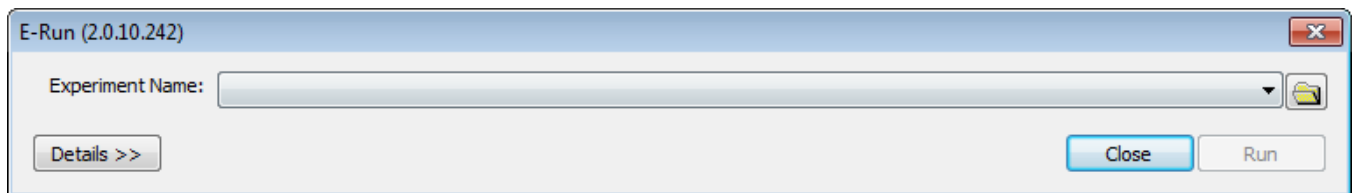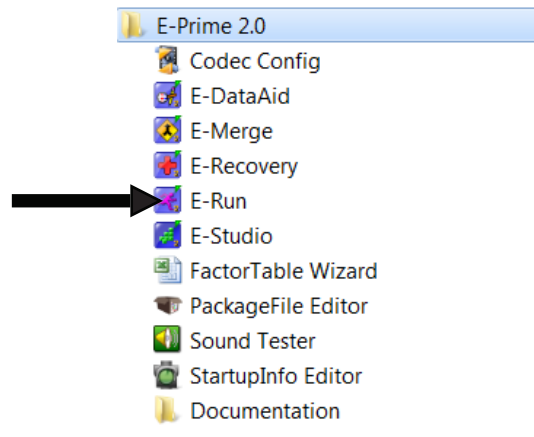
To run a pregenerated .ebs2 file in E-Run, enter the desired Experiment Name in the Experiment Name field. To display a list of recently run experiments, click within the Experiment Name field. To navigate to a different folder, click on the folder icon.

After the run is complete, the E-Run interface returns. You may run the same .ebs2 file again by clicking on the Run button, choose a different .ebs2 file to run by clicking on the folder icon and selecting a new .ebs2 file, or exit the R-Run application by clicking on the Close button.

Stage 8, Step 2: Running on Multiple Machines

E-Prime 2.0 must be installed locally in some manner on every machine on which its use is intended. However, the entire E-Prime 2.0 system need not be installed on machines to be used only for data collection. The E-Prime 2.0 installation includes a Subject Station installation that is approximately half the size of the full installation. To install the runtime application and only those components necessary

for running pre-generated .ebs2 files, insert the E-Prime 2.0 CD into the CD-ROM drive to launch the installation program. During installation, choose the Subject Station option. Repeat the Subject Station installation on all data collection machines. See *Chapter 1: Introduction* for details.

It is always a good idea to run an experiment to test that the installation was completed successfully prior to attempting to run actual participants. The BasicRT.ebs2 file is included with the Subject Station installation for this purpose. Refer to the previous section for information concerning running an .ebs2 file from E-Run. To run any other .ebs2 file using a Subject Station machine, simply copy the .ebs2 file (and any other files required by the experiment) to that machine, load the experiment into E-Run, and click the Run button.

E-Prime 2.0 collects only single participant data files, which include the .edat2 extension. Thus, for analysis, the single participant data files must be merged into a master file. The E-Merge application is used for this purpose (refer to Stage 9, this chapter). If data is being collected on multiple machines, the .edat2 files must be copied or moved to a common location for merging, unless the data collection machines are networked.

3.9 Stage 9: Basic Data Analysis

E-Prime 2.0 collects only single participant data files. For group analysis, the single participant data files must be merged into a master file using the E-Merge application. Detailed instructions for analyses can be found in Chapter 6: Data Handling. Here we provide a brief overview.

Stage 9: Basic Data Analysis
1) Merging Data
2) Checking Data Condition Accuracy and Data Quality
3) Data Editing • Tracking modifications
4) Analysis and Export of Data • Exporting raw data • Exporting a subset of the data • Creating and exporting tables

Stage 9, Step 1: Merging Data

E-Merge may be launched via the Start button. From the Start menu, choose Prime 2.0 to display the E-Prime 2.0 menu, and then, select E-Merge.
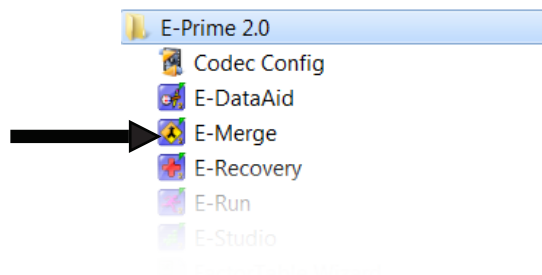

Alternatively, E-Merge may be launched via the Tools menu in E-Studio. E-Merge opens to display a Quick Reference dialog for guidance through the steps involved in merging files.
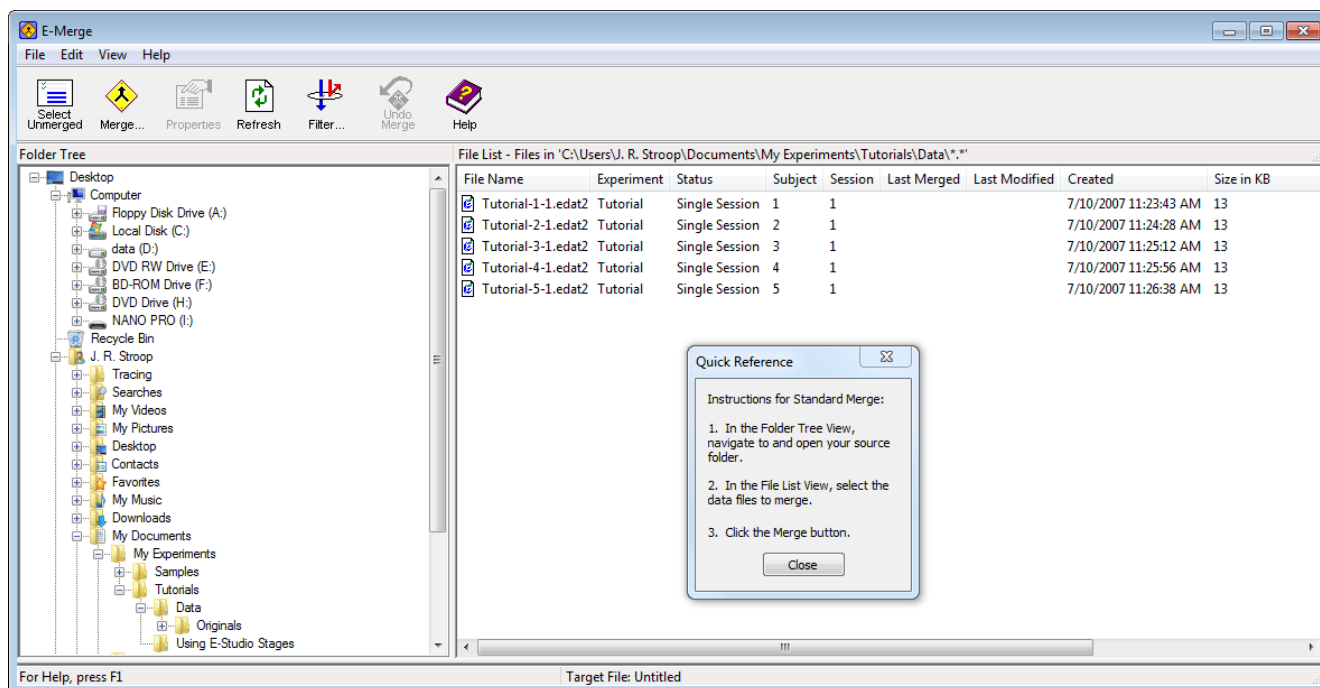
To merge individual data files, navigate to the folder containing the data files using the Folder Tree, select the files to be merged in the File List view, and click the Merge button. Refer to the *E-Prime Getting Started Guide* and Chapter 6: Data Handling for more detailed information.

In many circumstances, data files may be saved in different folders. For example, data files collected by different research assistants may be saved in separate sub-folders within the same folder, or on a shared drive. The E-Merge application provides a specialized merge procedure to facilitate the process of merging files arranged in this way. The Recursive Merge operation merges files from a selected folder and all of that folder's sub-folders. When a merge operation is launched, the user must specify whether the operation will be a Standard Merge or a Recursive Merge operation.
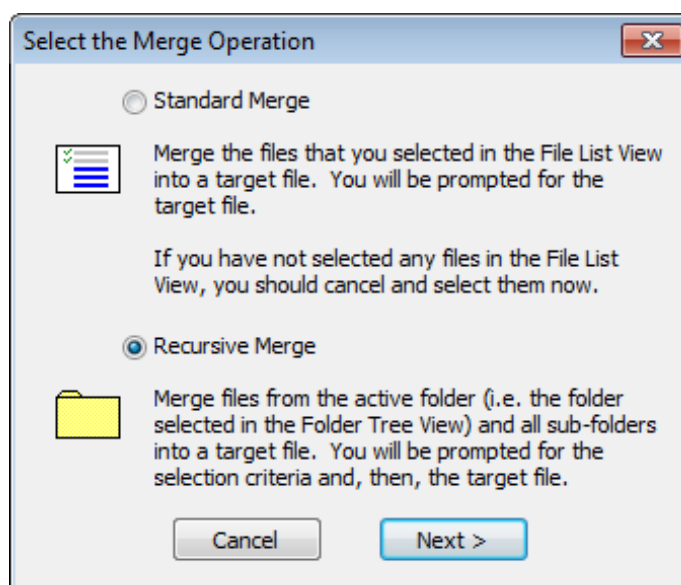
Data files may be collected on multiple machines, and then merged for analysis. When collecting data on multiple machines, take care to assign unique participant numbers. For example, when collecting data on four separate machines, a scheme could be used in which all participants run on the first machine are numbered 101-199, all participants run on the second machine are numbered 201-299, etc. For merge operations, it is important to keep experiment names and participant numbers unique, or conflicts will arise during a merge operation. There are two methods of merging the data for analysis.

Method 1: Copy all of the data from the separate machines to a single folder on one machine. Copying may be accomplished through file transfer, using read/write CDs, memory sticks or other portable devices, or e-mailing attached files to be saved into the common folder The relevant files are those with the .edat2 extension.

Method 2: Have a shared disk space accessible by all of the machines, and save the data file to that shared space . Each station could record its data in a separate area on the shared disk, and E-Merge's Recursive Merge feature could be used to aid in the process of merging these files from different folders. For example, a two-computer, two-experiment shared data folder might look like the folder tree below.
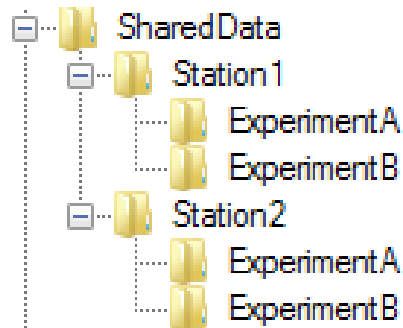

⚠ **NOTE:** *Running an experiment on a network has significant performance implications and is likely to result in timing errors. Please review the information in Chapter 4: Critical Timing carefully prior to setting up data collection in this manner.*

## Stage 9, Step 2: Checking Data Condition Accuracy and Data Quality

Once the data has been merged to a master file, the file should be checked to verify the data it contains and that the values are within expected ranges. Rather than a checking of the logging of variables (i.e., this would have been done prior to data collection), the checking at this point is more in reference to the completeness and appropriateness of the data, the experimenter's assignment of participant numbers, and assignment of participants to conditions. Particularly, if several assistants are collecting data for one or more experiments, it is easy to mistakenly assign the same subject number to several participants, or to fail to assign participants to a particular condition. E-DataAid is the application within E-Prime 2.0, which allows the examination and modification of data files.

Stage 9, Step 3: Data Editing

Data may be examined and edited using the E-DataAid application. To launch E-DataAid, choose E-Prime 2.0 from the Start menu, and then select E-DataAid.
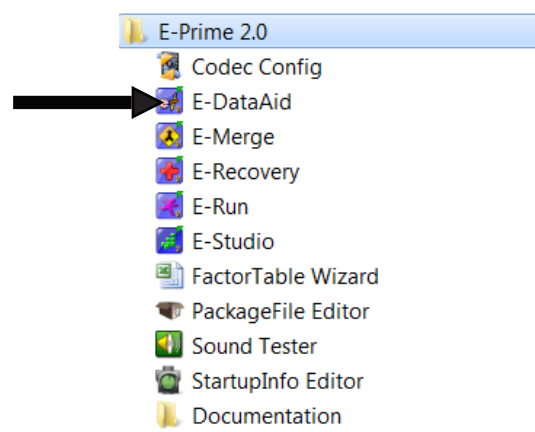

E-DataAid may also be launched from the Tools menu within E-Studio. The E-DataAid application opens without loading a particular data file. A data file must be opened within E-DataAid using the Open command in the File menu, or by clicking the Open tool button.
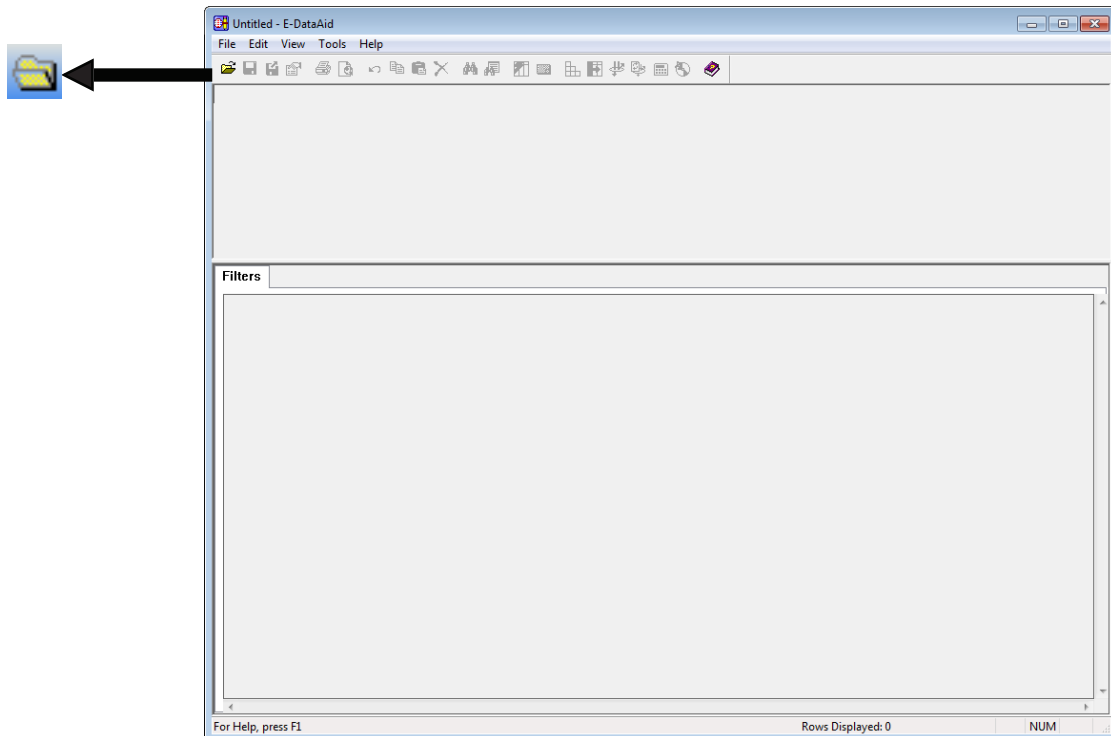
E-DataAid works much like Excel, allowing the addition of columns (i.e., variables) and modification of cell values. For example, to edit the value in a cell, simply place the cursor in the appropriate cell, delete the current value, and type the new value. To select a column, click the column header, and to move that column, click and drag the header to the new location. Specific tool buttons, commands, and filtering abilities are included in E-DataAid to organize, rearrange, or reduce the amount of data displayed.

Tracking modifications

E-DataAid includes additional features specifically designed to track modifications made to an E-Prime 2.0 data file (i.e., .edat2, .emrg2). For example, all edits made to a data file (e.g., modification of cell values, added variables are displayed in red, indicating a modification to the file. In addition, an annotation of each modification is written to an Annotations record which is saved with the data file. The example Annotations record below (from a merged data file) indicates the single participant data files from which the data originated, and notes that the subject number was modified for one of the participants (specifically, Participant 1 was changed to Participant 10).
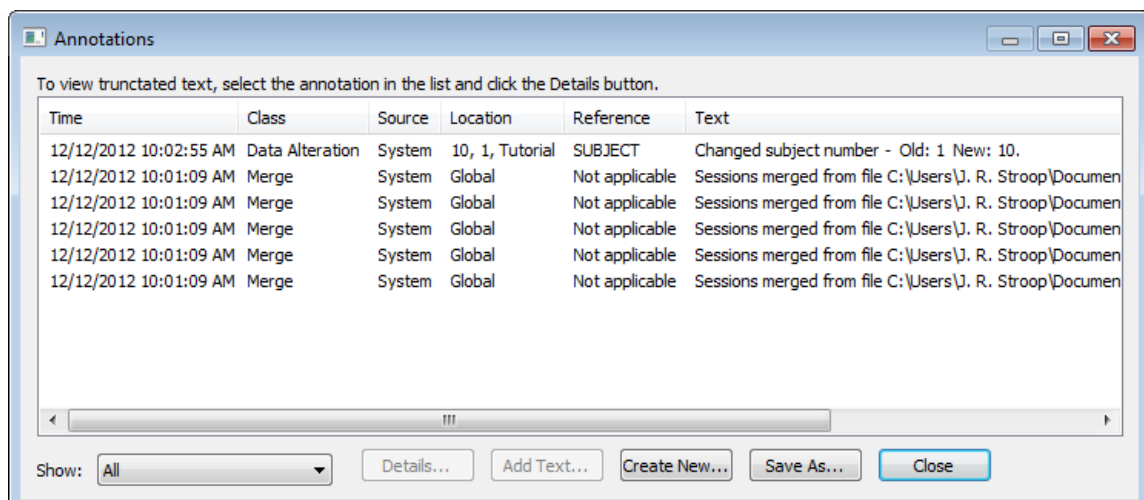

The Annotations record serves as a history of merges and modifications to allow the user to track not only the source files contributing to the master data file, but also any changes that were made to the data.

Stage 9, Step 4: Analysis and Export of Data

E-DataAid also allows the formatting and exporting of raw data, a subset of the data, and data tables (e.g., Mean RT x Condition).

Exporting raw data

To export the raw data, use the Export command in the File menu, or click the Export tool button. The Export command allows the output format to be specified for a particular statistical package.
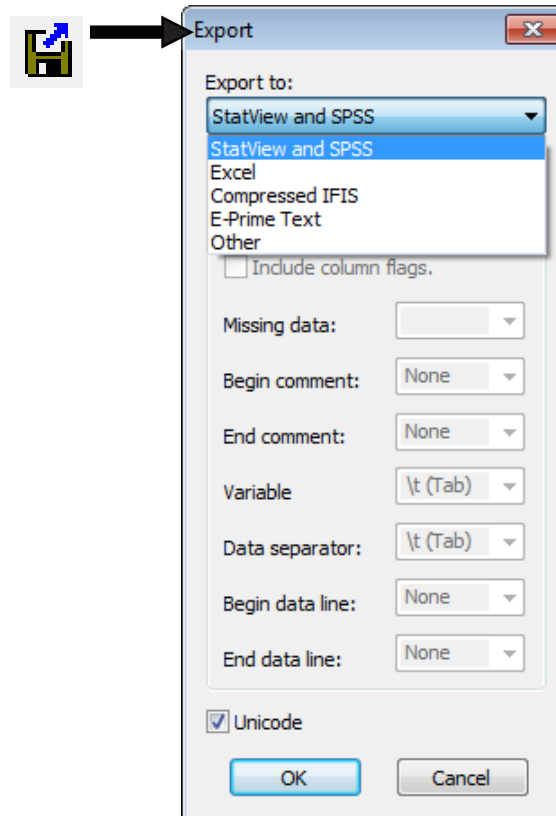

Formatting for several common software packages (e.g., StatView, SPSS, Excel, etc.) is offered as an export option. The "Other" formatting option allows the user to manually specify the format of the data during export so that the output file may be imported into a package that is not specifically listed in the export options.
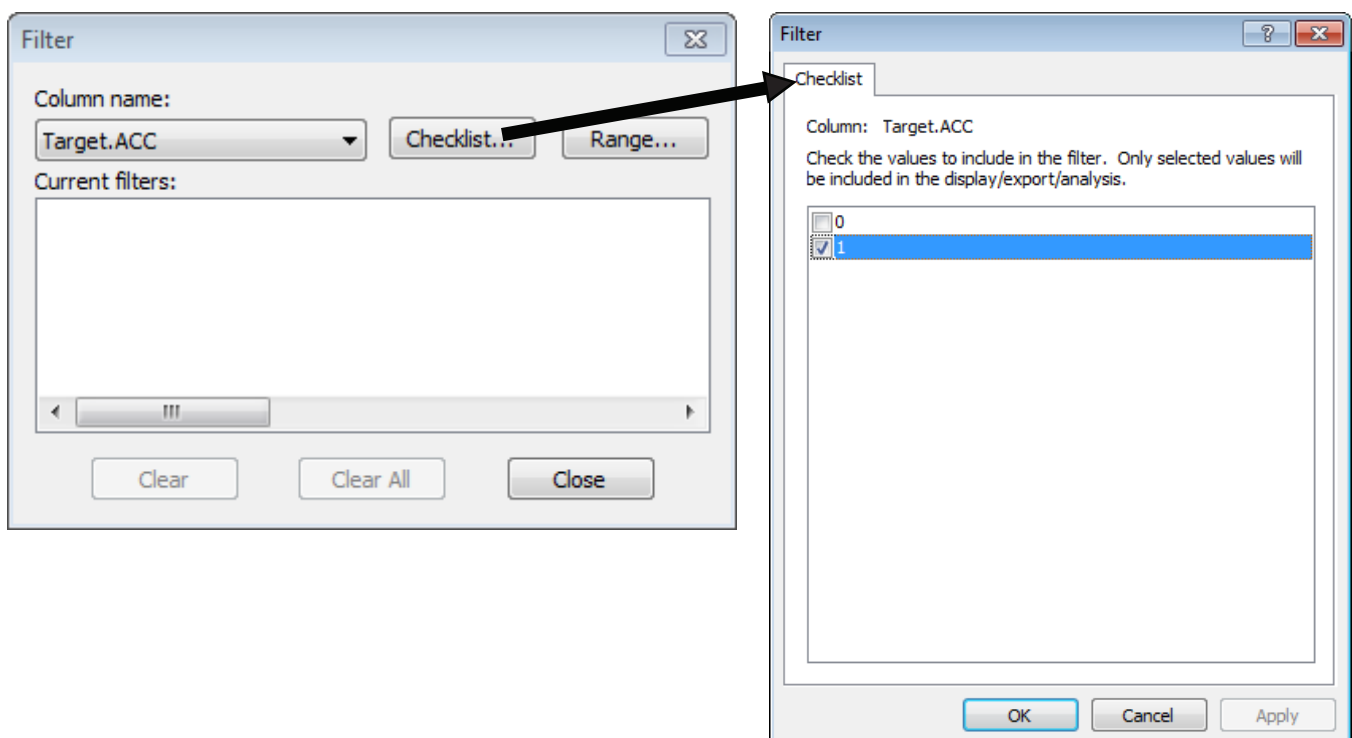
Exporting a subset of the data

During an export operation, all of the data displayed in the spreadsheet is exported. Therefore, to export only a subset of the data, the display must first be reorganized using the Arrange Columns and/or Filter commands in the Tools menu prior to the export operation.
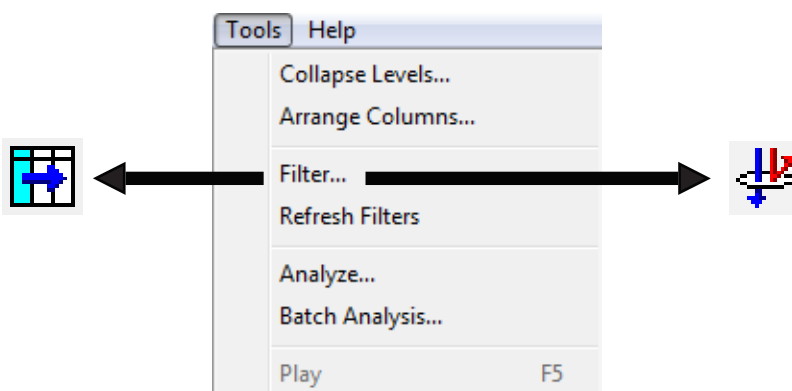
The Arrange Columns command allows the user to choose whether to hide or display specific columns in the spreadsheet, and to organize the order of the displayed columns.
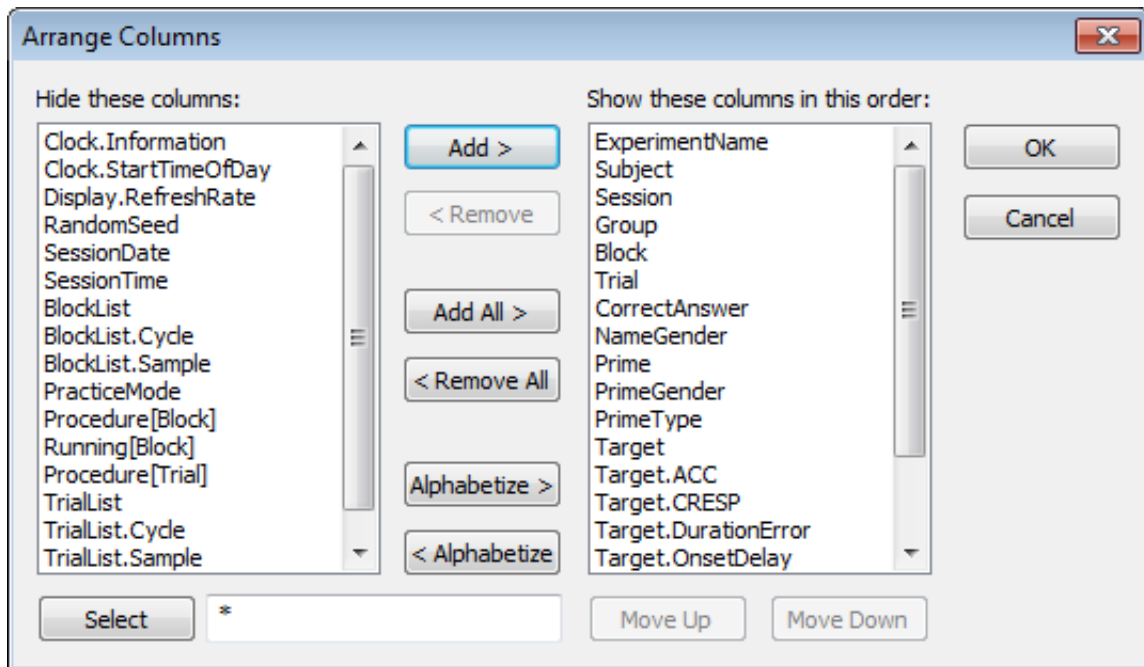
The Filter command allows the user to limit the display to certain values within a specific variable (e.g., only correct responses). To set a filter, choose the variable (i.e., column) name and the type of filter to apply (checklist or a range of values). Then select the specific values to include (i.e., apply a Checklist filter) or the boundaries for the range of values to include (i.e., apply a Range filter). Filters are inclusive, indicating the values that will be included in the display. The example below filters the data to include only correct answers (i.e., Target.ACC=1).

Once the columns are arranged and/or filters are applied, the export operation may proceed as usual, with only the displayed subset of the complete data file being exported, refer *Exporting raw data (Page 80).*

Creating and exporting tables

A summary table of values (e.g., MeanRT x Condition) may be exported using the Analyze feature. The Analyze command may be invoked via the Tools menu, or by clicking the Analyze tool button.
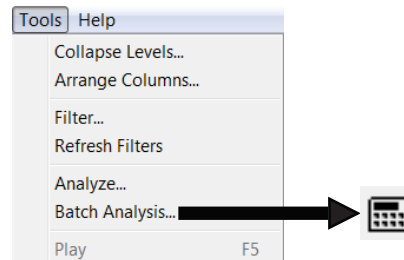
The Analyze feature allows the user to specify the summary analysis to perform. An analysis is defined by dragging the appropriate factors from the Variables list to fields (i.e., Rows, Columns, or Data) to specify them as row, column, or data factors. Once the analysis is specified, click the Run button to generate the table, and (from the resulting Table dialog) the Export button to perform the export operation. With the Save Analysis and Load Analysis commands, the Analyze feature allows analysis specifications to be saved so that they may be reloaded and run again. Analyses are saved with the .anl extension.

In addition to an export, the table may be copied directly into Excel, or to the clipboard for saving in an alternative software package. The data table above, when copied to Microsoft Excel, would appear as the following:

3.10 Stage 10: Archiving Experiments and Results

Stage 10: Archiving Experiments and Results
1) What Files to Store
2) Saving Data in a Spreadsheet and .edat2 formats
3) Saving Results of Analyses

Stage 10, Step 1: What Files to Store

For the purposes of replication, collaboration, and verification, it is necessary to keep thorough records of experiment files and methods. The files listed below should be archived in order to investigate, share, or replicate an experiment:

File Name	Description
ExperimentName.es2	Experiment specification file
ExperimentName.ebs2*	Generated E-Basic script file
Files called by the .es2 file (.BMP, .WAV, .TXT, etc.)	Images, audio, stimulus sets, etc.
ExperimentName.edat2	Single participant data file
ExperimentName..emrg2	Merged data file
ExperimentName.anl	Analysis specification

The .ebs2 file need not be archived as it can be regenerated via the .es2 file. However, if collaborating with a lab owning only a Run-Time license, access to the .ebs2 file without having to regenerate would be convenient. Refer to 1.6 Hardware Key and Licenses (Page 8), for information concerning licensing.

In order to minimize confusion when passing an experiment from one experimenter to another, or when returning to a previously run study, it is important that the experiment specification (.es2) file be accompanied by a thorough explanation of the experiment specifications (a description of the design, the factors and levels, etc.). This could be accomplished using any text editor to create a summary. However, the Notes field on the Common tab available via the Property pages for each object is an ideal location for such information. For example, the Notes property for a List object could be used to record a description of each of the attributes defined within the List, as well as the levels for those attributes.

Stage 10, Step 2: Saving Data in a Spreadsheet and .edat2 formats

Researchers are expected to maintain records of their data for a time after collection (typically 5 years after publication). It is recommended that the single participant data files (.edat2) be saved as well as any merged data files (.emrg2). In addition, it is recommended that the raw data be saved in ASCII format by exporting the data as E-Prime Text, as was described in *3.9 Stage 9: Basic Data Analysis (Page 76)*.

⚠ **NOTE:** *When saving the raw data, it is important to remove any filters, and re-display any hidden columns prior to the export, so that the entire data file is saved. Use the Restore Spreadsheet command for this purpose.*

Stage 10, Step 3: Saving Results of Analyses

In order to regenerate any and all analyses, it is recommended that the researcher maintain a copy of any plots and tables generated from the data. If any analyses were performed within E-DataAid, a record of the analysis should be saved as an .ANL file (refer to *Creating and exporting tables (Page 82)*.

3.11 Stage 11: Research Program Development

Note that some users may include copyright notices in their experiments. Be respectful of the constraints authors place on sharing script, and always give credit for script used.

Stage 9: Basic Data Analysis
1) Modifying Experiments

2) Sending Experiments to Colleagues
  • Licenses
  • Hints for distributing experiments
  • Include clear and thorough instructions for running the experiment
  • Include design notes
  • Determine the type of license the recipient owns

3) Developing Functional Libraries

Stage 11, Step 1: Modifying Experiments

When modifying experiments, it is important to maintain consistency as much as possible. This will aid not only in the understanding of the differences between versions of an experiment, but also in the merging of data files collected by different versions. Consistency should be maintained not only in the naming of the experiments themselves, but also in the naming of factors and attributes within the experiments. For example, it would be easiest to view the differences between files if objects performing similar duties were similarly named (e.g., the object presenting the stimulus should be consistently named Stimulus). Procedures running tasks at specific levels of the experiment are more easily understood if named to reflect the level (e.g., TrialProc, BlockProc).

When making modifications to an existing experiment, it is recommended that new conditions be added rather than deleting old conditions, making the new experiment a subset of the modified experiment. Detailed records should be kept to outline the differences between versions of an experiment, refer to *Stage 10, Step 1: What Files to Store (Page 83)*. Another useful practice is to name versions of the same experiment using a consistent experiment name and a version number (e.g., ExpName-001.es2).

With the intention of merging data files, care should be taken to maintain consistency in the naming of variables (i.e., attributes and levels). For example, if an attribute is named "Stimulus" in the first version of an experiment, and "Stim" in the second version, the merging of the data files obtained by the two versions would result in two columns of data for what is essentially a single attribute. The values for half of each column would be set to NULL. The user would necessarily have to copy and paste information in order to relocate the appropriate information.

Consistency in naming of levels and attributes will avoid potential errors during the merging of data files. E-Merge will report problems if the same name is used to refer to an attribute in one data file and a level in another. Likewise, errors will be generated during a merge operation if a single name is used to refer to different levels in different data files (e.g., level 2 is called "Trial" in one data file and "SubTrial" in another). With the exception of the same name being used for both a variable and a level, the problems are not impossible to overcome, but are inconvenient and would require some manual editing. Refer to the Chapter 6: Data Handling for more detailed information concerning conflicts during merge operations.

Take care not to change the meaning of a variable across experiments that are to be combined for analysis (e.g., do not use Probe to refer to animals in one experiment, and fruits in another). If need be, add a new attribute or level, but avoid changing the experiment without modifying the logged variables.

Stage 11, Step 2: Sending Experiments to Colleagues

An E-Prime 2.0 license owner is free to distribute any files created by using the system. Files created by users include Experiment Specification files (.es2), E-Basic Script files (.ebs2), data files (.edat2, .emrg2), and analysis files (.ANL). However, according to the license agreement, users are not permitted to distribute any part of the E-Prime 2.0 system, including the runtime application. Refer to *Chapter 1: Introduction* (*1.4 Installation, Registration, and Validation Instructions (Page 6)*) and the information below for definitions of licenses, permissible actions according to the type of license, and files used by each type of license.

Licenses

The full E-Prime 2.0 license permits use of each of the applications available within the E-Prime 2.0 suite of applications. The full license permits the development of new experiments (E-Studio), collection of data (E-Run), merging of data files (E-Merge), and data editing and export (E-DataAid).

The full license includes installation options to permit the installation of the full system, installation of only the runtime components for data collection, and a custom installation option to install only a portion of the full system. E-Prime 2.0 must be installed in some manner on any machine on which E-Prime 2.0 is to be used. For development, the full installation is required. For data collection, the Run-Time Only Installation is necessary. The custom installation is most useful for adding components to a previous installation. Refer to *Chapter 1: Introduction* for a complete description of installation options.

Hints for distributing experiments

The following is a list of suggestions and considerations when sharing or distributing experiments:

Include clear and thorough instructions for running the experiment

Many times the instructions for tasks or response keys are not indicated in the actual experiment instructions. To facilitate the running of an experiment by someone other than the programmer, it is suggested that thorough instructions concerning the task and the response keys be included as a display at the beginning of the experiment.

Include design notes

It may not be obvious to the person receiving an experiment why certain values were used, why certain procedures were created, or what certain variables were recording. Notes field on the Common tab of each object's property page is the ideal place for adding comments concerning the parameters, variables, and organization of the experiment. If the recipient does not own a license including E-Studio, access to the Notes is not possible. Therefore, a text file would be more appropriate.

For demonstration purposes, include a shortened version of the experiment in addition to the full version.

It is often convenient for the person receiving an experiment from someone else to be able to get a preview of the experiment and the data being collected. A shortened version will allow the recipient to quickly view the task, and to collect a small data file for examination rather than working through hundreds of trials.

Determine the type of license the recipient owns
Refer to *Chapter 1: Introduction* concerning the types of licenses and installation options available for E-Prime 2.0. Determining which license the recipient owns will establish which files are necessary to run and modify an experiment, or to examine data files.

Stage 11, Step 3: Developing Functional Libraries

It is likely that many experiments and procedures will be used repeatedly. Therefore, it is beneficial to develop a library of experiments or subroutines that may be modified, inserted, or used as patterns for other experiments. When an experiment has been completed and tested, it is suggested that a copy of the experiment be placed into the library. This copy serves as a record of the experiment, a clean copy that may be retrieved if modifications are made to the original file, or as the basis for a different version of the same experiment.

Another suggestion is to create a package file for procedures used repetitively (e.g., criterion-based exit of a procedure, contingent branching). Package files from a library may be inserted into new experiments in order to avoid wasted time regenerating existing script. Examples of package files might be the N-Back memory task item selection, psychophysical threshold procedures, or moving window text presentation. Refer to the PackageCall object documentation in the *E-Prime 2.0 New Features/ Reference Guide* for further information.

Chapter 4: Critical Timing

Key Points
No program running on a common desktop operating system can deliver accurate measurements at every millisecond.

E-Prime 2.0 provides custom solutions to obtain the most accurate stimulus presentation and response collection times from your PC. These solutions address the challenges that derive from restrictions in the following areas:
   • Operating system
   • Stimulus preparation needs
   • Stimulus presentation limitations
   • Response collection limitations

E-Prime 2.0 monitors its own performance, and flags potentially problematic trials for closer inspection and exclusion from data analysis.

4.1 Introduction

4.1.1 What are the challenges?

There has been a long history of attempts to achieve accurate behavioral timing via computerized experimental research methods (Schneider & Shultz, 1973; Chute, 1986; Schneider, 1988; Schneider, 1989; Segalowtz & Graves, 1990; Schneider, Zuccolotto, & Tirone, 1993; Cohen et. al., 1994; Chute & Westall, 1996). As the complexity of computer hardware and operating systems has increased, it has become more difficult to program and assess the precision of behavioral experiments. As computer operating systems have developed more capacity to perform multiple tasks (e.g., network communications and resource sharing, disk caching, concurrent processing) and more virtual tasks (e.g., virtual memory management, interrupt reflection, low-level hardware commands being routed through layers of virtual device drivers rather than direct hardware access), more effort must be expended to achieve and verify accurate timing.

Many researchers say "I want millisecond precision timing for my experiments." It is helpful to take a moment and write down an operational definition of what that means. If you were to look at two software packages that both claimed millisecond precision, what quantifiable evidence would you want before you would be comfortable reporting scientific data with them? If you interpret millisecond precision to mean that no individual time measurement is ever off by more than a millisecond, then you cannot use modern personal computers running common desktop operating systems to conduct the research.

⚠ **NOTE:** *The most serious problem for computerized psychological research is that the computer requires substantial time to perform actions and may halt the experiment without notice, thereby grossly distorting timing.*

You have probably experienced times when your word processor occasionally pauses for a short time and then resumes. This same process can happen during an experiment.

Let's take a simple example: assume you intend to display seventy-two bitmap images, each for 200 ms (e.g., a rotating checkerboard). If you program this example in a standard programming language such as C or Java and check the timing using a stopwatch, you will likely find that the experiment takes longer than the expected 14,400 ms to run. When we implemented this experiment in E-Prime 2.0 without taking full and proper advantage of E-Prime 2.0's timing features, we also obtained undesirable results. As we'll explore in detail below, timing results from this experiment also show the following characteristics:

• The median display time is greater than the specified display time of 200 ms
• Periodically, there are large spikes in the display time for an individual image
• These large spikes occur intermittently over multiple experimental runs

In the remainder of this chapter, we explain how the operating system and stimulus presentation hardware contribute to these undesirable results, and how E-Prime 2.0 equips researchers to both effectively work with and around the functional limitations to obtain the most accurate timing possible and also to confirm the critical timing aspects of their experiment.

4.1.2 Defining Millisecond Accuracy

Operationally, E-Prime 2.0 defines the interpretation of millisecond precision as the ability to report any reaction time or timing duration such that:

1) A measured and reported timing precision standard deviation is less than half a millisecond.
2) Recordings are within a millisecond from the time they are available to computer hardware. When errors occur, they should be detectable, should not increase measurement variance by more than 1 ms. The investigator should have a method available to identify and filter out timing data that is in error.
3) Screen refreshes (displays of the full screen, described in detail below) can be tracked without misses 99.9% of the time, and misses can be detected and reported.

With such accuracy, researchers can be confident that they are reporting valid results of human behavior, and that those results will replicate between laboratories.

4.2 Timing Challenges Detailed

4.2.1 Operating System Challenges

Key Points
The operating system is in charge.
Multi-core systems alleviate many of the obstacles to critical timing that apply to single processor systems.

At times, operating systems take away processing time from other programs. Even if the other program is reading a microsecond precision clock, the precision is only as good as the accuracy with which the computer processor is actually free to read the clock. For example, assume a software application attempted to continuously read a hardware crystal clock that updated exactly every millisecond. If the application read the clock for a period of 10 seconds (10,000 milliseconds), then you would expect that the application would observe the value read from the clock to change exactly 10,000 times and the difference in time reported between sequential reads to be either 0 or 1 millisecond. However, if such a test application is run on a modern operating system, the application will often observe that, occasionally, the difference between sequential reads of the clock is significantly greater than 1 millisecond. Assuming that the hardware crystal clock itself is independent and runs continuously, these results can only be explained by the application having its reading of the clock paused at times during execution. When the operating system suspends the software application that is reading the clock, the crystal clock itself continues to run, and these individual clock ticks are "missed" by the suspended software application. We refer to the percentage of times for which these individual reads of the clock tick are missed by the software as the "miss tick rate". The maximum amount of time that elapses between two consecutive reads of the clock is referred to as the "maximum miss tick duration."

Modern multi-tasking operating systems (e.g., Windows, Mac OS, Unix) will not allow exclusive and continuous execution of any process, because the operating system must take cycles to perform critical- functions (e.g., virtual memory management). No program running on a common desktop operating system can deliver accurate measurements at every millisecond. In other words, you cannot obtain a 0% miss tick rate. For example, Windows will indiscriminately remove parts of a program from memory to test to see if a program really needs them in order to reduce the applications working memory set. This operation and others like it will produce random timing delays[1].

---

[1] The Windows NT documentation on the virtual-memory manager states: *The working-set manager (part of operating system) periodically tests the quota by stealing valid pages of memory from a process… This process is performed indiscriminately to all processes in the system*. Kath, R. (1992, December 12) The Virtual-Memory Manager in Windows NT, *Microsoft Developer Network Technology Group*, 12.

Let's consider a different example: recall the rotating checkerboard program that was mentioned in the introduction to this chapter. We implemented this experiment in E-Prime 2.0 but without properly using E-Prime 2.0's timing features. Figure 1 below illustrates the measured times from two such runs. The thick line at 200 ms shows the intended time. Session 1 (solid line) and Session 2 (dashed line) have a median display time of 236 ms with irregular spikes as high as 374 ms. Further, the displays seem to be taking about 240 ms, with the longest display taking over 370 ms. In addition to the mean timing error, when running the program multiple times, these long pauses happen at different times
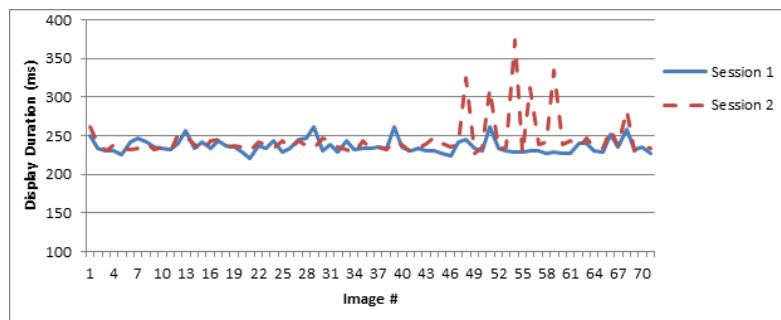

Figure 1 Actual display durations for displays run on 2 sessions relative to the intended (200 ms) duration.

What can account for these results? The task specifications are clear, present each display for 200 ms, but clearly this is not occurring. Instead, the results illustrate three glaring problems. First, the median duration is about 36 ms longer than was intended. Second, there are severe spikes indicating long unintended pauses in the experiment. Third, the spikes happen at random times.

The first problem, that the median duration is longer than the specified duration time, is most strongly influenced by the time required for stimulus preparation, is detailed in the next section.

The second and third faults in the Figure 1 timing data are related to the spikes in timing durations. Why are some delays very long, and why do they occur at random intervals, changing even if run under exactly the same conditions? The spikes are due to the Windows operating system taking control of the machine and halting the program to perform various administrative and/or "clean-up" activities. The operating system will take control and halt the program without any indication (other than that it is temporarily non-responsive). The program pauses for a period ranging from a few milliseconds to hundreds of milliseconds while the operating system performs actions such as managing virtual memory (i.e., portions of the computer's memory are stored or reloaded from disk to provide a larger effective address space).

These actions can occur at nearly any time[2]. The action appears random because when the operating system takes an action is dependent on all of the other events that have occurred in the operating system since it was powered up, and perhaps even since it was installed. For example, if you read in a large word processing document, most of the "real" memory that you have is used up. You now start the experimental program, and the program begins reading in files for the images. Perhaps there is space in memory for the first fourteen images, and they come in fast (100 ms) as the operating system still has room in memory to store them. On the fifteenth image, however, the operating system has no new space in memory to store the image. The operating system then decides to save some of the word processing document to disk to make room. Because it was the operating system's management routine that decided to make more space, this takes a higher priority than the program, and as a result the program is halted.

[2] A request from your program or any other program can trigger operating system actions. With mapping of virtual memory to disk, your program and any files you read may not even be in memory. As you need parts of your program, they are loaded in (e.g., you present a stimulus from a different condition). You may see more delays in the first block of your experiment, as the operating system loads parts of your program as they are needed and continues to clean up from the previous program that was running.

The process of the operating system procuring room and scanning the system to perform various housekeeping operations can take tens to hundreds of milliseconds. During this time, the program is paused. The program will once again read images fine for a while, and then may pause again. In the example shown in Figure 1, a spike occurs in the display duration on Image 54, Session 2.

It is very important to remember that the operating system is in control of your computer, not you. You do not issue commands to the operating system; you only issue requests. You may ask the operating system to do things, but it will, at times, delay your task to perform what it views as critical functions. The operating system typically pauses an executing program dozens of times a second without recording the behind-the-scene timing changes.

A note about multi-core and multi-processor systems
Our testing has shown that many of the timing problems that arise from the operating system attending to other programs are significantly reduced when running on a processor with multiple cores. For example, Appendix A reports a 0% missed clock read rates when we tested E-Prime 2.0 on two different multi-core CPU systems, but a missed tick rate of approximately 10% when running the same test on a single processor machine. Nevertheless, operating system-related timing errors cannot be completely eliminated, and the E-Prime 2.0 solutions described in this chapter should still be applied to multi-processor systems.

4.2.2 Stimulus Preparation Challenges

Key Points
Most stimuli, whether visual or auditory, require setup time before they can be presented.
A sound and movie files are processed by a variety of drivers and/or APIs. Multiple configurations can work with the same found file, but many configurations introduce delays.

We'll continue now with the rotating checkerboard example. The first timing issue, median display times in excess of the intended display duration of 200 ms, in large part reflects the time that is required to prepare the stimulus for display. Recall that the images stored on the disk are to be displayed for 200 ms each. In order to present these files, the following process must occur:

• The image data is read by the computer from the disk
• The image is prepared for presentation
• The image is copied to the display screen
• The computer waits for the stated display time (200 ms)

This cycle is then repeated for the next image presentation. The time to read the image from disk, prepare it for presentation and copy it to the screen required a median time of 36 ms. Thus, the median total time for the image display is 36 (setup time), plus the specified display duration of 200 ms, for a total of 236 ms. This is an illustration of the problem: It is not possible for the computer to put up a display without preparation, which accounts for the "setup time".

Similar issues occur with other types of stimuli. Any graphic file, regardless of format, requires time to load. Graphic files that are in compressed formats (e.g. .gif and .jpeg) must be interpreted ("un-compressed"). Similarly, visual stimuli that are created with E-Basic's graphic's commands (e.g. circle, line), require setup time as well.

Lastly, sound and movie files also require preparation time. Clearly, as is the case with image files, sound and movie files must be read by the computer from the disk. Sound and movie files are also compressed when they are initially stored   It is not feasible to store and/or transmit audio and movie

files in an uncompressed format. For example, uncompressed stereo audio files that are CD quality, meaning 44 KHz sampling rate, 16 bits per sample, requires about 150 KB of hard disk space[3] for each second of sound.

The compression and decompression process for audio and video are more complex than for static image files. There are hundreds of codecs, and they can be packaged in different ways. For example, "containers" are often used to store both audio and video streams for video files. AVI is an example of a popular container. Containers required an extra processing step, because the AVI must first split the incoming signal to be presented into the audio and then video components, and then hand these components of to a codec that knows how to process them.

4.2.3 Stimulus Presentation Challenges

Key Points
The hardware that presents stimuli, whether visual (video card, monitor) or auditory (sound card and its related drivers), restricts the rate at which stimuli can be presented.

Visual Stimulus Limitations:
    • Displays are accomplished via refresh cycles.
    • Refresh cycles are rarely exact multiples of the intended display duration.

Auditory Stimulus Limitations: • Sound Latency, the delay between when a sound card receives the instruction to play a sound file and the sound actually begins, is present in many different sound cards and the method in which the sound is told to play.

Visual Stimuli

⚠ **NOTE:** *The hardware that comprises the computer display, along with the process by which monitors display information, determines the display durations that are possible.*

Therefore, in order to understand the limits on display durations, we need to examine some of the details of the hardware and how it works. Computer display devices, both CRT and LCD displays, paint every dot or "pixel" of a display sequentially. The display begins to be drawn at the top left, and pixels are then painted horizontally across the row, move down to the second line and then continue horizontally. The process is repeated for each row of pixels until the bottom of the screen is reached. This process of painting all the pixels sequentially for a full screen is called the refresh cycle, and typically takes 10-18 ms (the "refresh duration").

The start of a refresh cycle is referred to as the vertical blank event. On a CRT monitor, this is when the electronic gun on the monitor moves from the bottom right of the display to the top left of the screen to restart the refresh. The computer can sense the vertical blank event in order to synchronize the computer events to what is occurring on the display. The display card, the monitor, and the screen settings (e.g., resolution and color depth) all determine the refresh cycle or refresh rate.

LCD monitors also have a vertical blank event. However, in contrast to the CRT monitor, the vertical blank event does not refer to a physical event, because an LCD monitor does not have an electron gun. LCD monitors do, however, have a delay after the full screen has been "painted", during which information about the next screen to be displayed is transferred from the video buffers of the hardware card itself and then through the analog or digital connection to the LCD. This data transfer time imposes a limit on how quickly the display can be updated, similar to the limitation imposed by the vertical blank event on a CRT. Largely for purposes of backwards compatibility with CRT based systems, this period of data transfer from memory buffers to the video card is still referred to as the vertical blank event.

---

[3] Source: Microsoft (n.d.). *Compressing sound files*. Retrieved from http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/compressing_sound_files.mspx?mfr=true.

A typical display resolution includes 1024 pixels horizontally and 768 pixels vertically. Figure 2 shows time progression of the display of each pixel row in a 1024x768 display when a 60 Hz refresh rate is in effect.
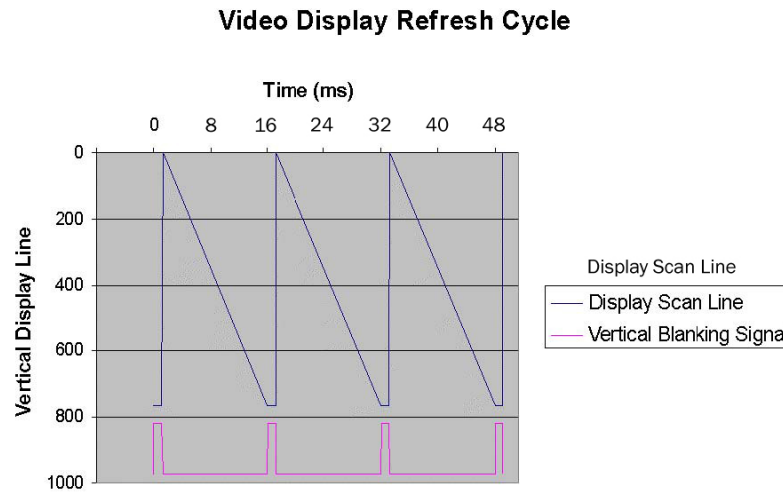
Video Display Refresh Cycle

Figure 2. Video display refresh cycle for a 1024x768 display with a refresh rate of 60Hz (16.666 ms refresh duration). The top line of the display is line 0, and the bottom line is 767. Each pixel is displayed once every 16 ms.

To calculate the refresh duration in milliseconds given the refresh rate specified in Hz, use the following equation:

Refresh Duration (ms) = 1000 / Refresh Rate (Hz)
Example: Refresh Duration = 1000 / 60
Refresh Duration = 16.666 ms

To calculate the refresh rate in Hz given the refresh duration specified in milliseconds, use the following equation:

Refresh Rate (Hz) = 1000 / Refresh Duration (ms)
Example: Refresh Duration = 1000 / 16.666
Refresh Duration = 60Hz

On a CRT monitor, during the refresh cycle the video display hardware activates each pixel for a short period of time (about 3 ms per refresh cycle). Figure 3 shows the typical activation the eye sees for a pixel in the middle of the display. The example presents a fixation, probe, mask sequence of "+", "CAT", and "***" respectively. Each write of the data to the video hardware is assumed to occur during a vertical blanking interval (e.g., screen writes performed at offsets of 0, 16 and 32 ms). The pixels for the middle of the display (line 384) would be refreshed and displayed at the approximate offsets of 8, 24, and 40 ms.
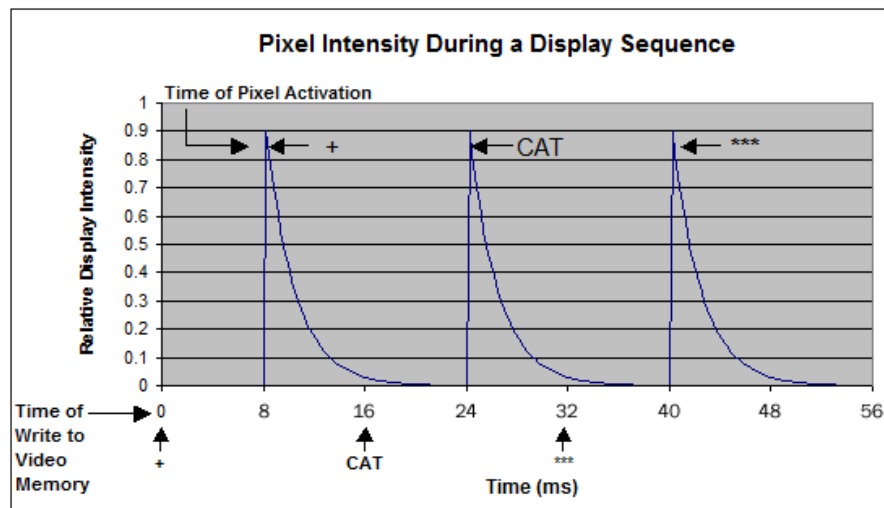
*Figure 3. Pixel activation of center of display for "+" presented at 0 ms, "CAT" presented at 16 ms, and "***" presented at 32 ms. The participant actually first sees "+" at 8 ms, "CAT" at 24 ms, and "***" at 40 ms*

The arrows at the bottom of the figure illustrate when the text is written to the video display card. The exponential decay functions show what is actually displayed to the participant's eyes when the experiment is presented using a CRT monitor. The delay in the activation is due to the vertical refresh cycle. The intensity change is due to the phosphor decay of the CRT monitor after the pixel was activated once per refresh cycle. Our testing shows that the phosphor decay is approximately 1.2 ms, and further that it is fairly consistent across different monitors. Appendix D explores in more detail the psychophysics of how the computer generated displays are actually perceived.

LCD monitors
LCD monitors do not experience the phosphor decay effect because of the technology they use to maintain the image. As noted above, LCD monitors are still participant to the concept of a refresh cycle to update image data on screen, but once updated the image is does not decay. Instead, it is rather continuously illuminated via florescent light or similar technology.

However, LCD monitors introduce a new timing issue, display latency. Display latency is the delay between when the monitor receives the instructions to present information on the screen and when that information becomes visible. LCD manufactures typically report some type of rise- and fall-time, which is intended to reflect how long a pixel takes to switch from black-to-white and white-to-black. The display latency issue includes this rise and fall time, along with the time required for the image to work its way through the LCD circuitry.

When LCDs first became available, Psychology Software Tools recommended that they not be utilized, and that CRTs were the preferred monitor type, for the following reasons:

1) The phosphor activation and decay on a CRT monitor are well-defined and widely understood, while the rise and fall times reported for LCD monitors were ill-defined and difficult to compare[4]. Vendors typically reported latencies for switching a pixel from black to white, which will show the fastest rise time. More recently, vendors have begun reporting gray-to gray specifications. However, there is no industry standard for this metric either, and the gray settings can be manipulated easily to suggest better results.

---

[4] For example, see the following article: PC World. (2005, July 22). LCD Specs: Not So Swift. Retrieved from http://www. pcworld.com/article/121906/article.html.

2. CRTs outperformed early LCD monitors. CRT's phosphor activation times are relatively constant and less than 2 ms, whereas the display latencies for early LCDs were often over 10 ms. However, our recent testing efforts have shown certain LCD monitors with acceptable display latencies. In general, we recommend that display latencies be less than on-half of the refresh rate. See the KB 3133 - INFO: Monitor Recommendations and Timing Information for our latest test results.

In conclusion, our original recommendation against using LCD monitors no longer holds, albeit with the following cautions. First, you should always check the manufacturer's specifications. Second, we continue to recommend that you test your LCD monitor with a third party test kit, such as the Black Box Toolkit, to determine if the monitor's display latency is within your experimental requirements. While PST's testing has identified LCD monitors that are capable of providing acceptable performance, we cannot provide a blanket endorsement of LCD monitors, either now or in the future. We continue to recommend testing in part because of potential technological changes that might be coming in LCD technology, such as painting the display in quadrants rather than line by line and various interlacing techniques. Such changes would almost certainly impact the timing of the displays.

Consequences of the refresh rate
There are three important consequences of the display having a fixed and non-varying refresh cycle. First, you cannot draw or remove a display at any time. Displays may be presented or removed only when the next refresh cycle comes along. For example, if the refresh cycle is 16 ms, displays may be changed at delays of 0, 16, 32… milliseconds from the time of the vertical blank that initiated the display sequence. The actual display of the stimuli that are half way down the screen will be halfway through the refresh cycle, or at 8, 24, and 40 ms.

Second, displays that are not a full refresh cycle in duration may possibly not be visible, and this varies depending on the position of the stimulus. For example, let us consider the same fixation-probe- mask experiment used above, but this time let us change the display specifications so that the fixation "+" is presented at the time of the first vertical blanking event, the probe word "CAT" is written 5 ms later, and finally the mask ("***") is written 5 ms after the probe word. Given this stimulus timing, the "+" will actually never be presented to the participant, and instead the video monitor would present "CAT" at 8 ms and "***" at 24 ms (see Figure 4). This is because the "+" would be overwritten by" CAT" before it is every refreshed. The first refresh of the pixels in the center of the screen occur at time 8 ms, but the write of the probe "CAT" occurs at time 5 ms, which is 3 ms prior to the refresh of the pixels in that area of the screen.
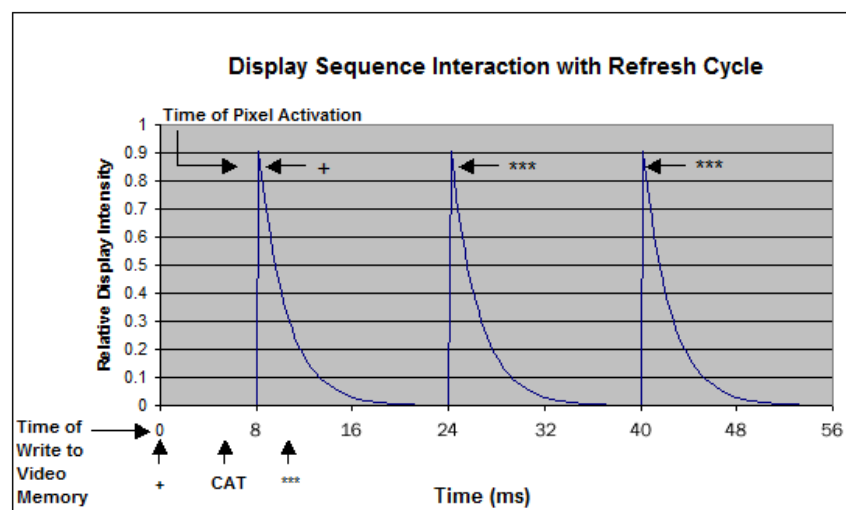

*Figure 4. Presenting displays for less than a refresh cycle produces partial displays. The arrows on the bottom show writing of the stimuli "+", "CAT" and "***" to video memory at 0, 5, and 10 ms respectively. The participant sees "CAT" at 8 ms, and "***" at 24 ms and again at 40 ms. Note that the "+" is overwritten by "CAT" before it is ever displayed to the participant because it is not refreshed before it is overwritten.*

The third consequence of the refresh cycle is that while you may specify a fixed display duration, if it is not an exact multiple of the refresh rate then the duration will vary up to one refresh cycle less than or greater than the desired time. For example, if the refresh rate is 60Hz (a refresh duration of 16 ms) and the specified display duration is 200 ms, the participant will see durations that are either 192 ms (12 refreshes) or 208 ms (13 refreshes). Since the requested duration (200 ms) is 50% of the distance between the onset times of the two refresh cycles (192 and 208), 50% of the refreshes will be at 192 ms and 50% at 208 ms.

A final concern with the refresh rate is that Windows and/or the video card manufacturers can mis-report the actual refresh rate being utilized. With respect to the operating system, Windows provides some type of interface to enable the user to select a refresh rate and screen resolution (for example, via the Display settings in the Control Panel). While this interface suggests that the user is setting the refresh rate, in actuality Windows treats this value as a request, and allows the display device driver to provide its best approximation to the request.

Further, with respect to the video card, the company that implemented the display driver code configured the software to display an acceptable image for business applications, but did not ensure that it was accurately reporting the refresh rate, or reporting back to Windows a close estimate of accuracy.

Auditory and Video Stimuli
As is the case with graphic files, time is required to read an audio file from the computer's hard drive into memory. However, there are additional issues regarding timing when you are presenting auditory stimuli. The sound card, the sound device drivers, and the operating system all combine to determine the speed with which sound files can be played. When presenting sound files as stimuli, there is always a delay between when the audio card receives the command to play and the sound is actually played. This period is referred to as the sound latency. Sound latency varies between cards, but our testing also shows that it varies with the selected sound card driver, selected API (application programming interface), and operating system as well.

Psychology Software Tools conducts on-going testing of a range of sound cards, particularly when a new operating system has been released. See our Knowledge Base articles: KB 1307 INFO: Sound Latency – Not all sound cards provide optimal millisecond timing and KB 4347 INFO: Sound Latency – Windows Vista/Windows 7 and 8 (and beyond).

4.2.4 Response Collection Device Challenges

The keyboard is frequently used as a response collection device. However, when responses are collected through the keyboard, mouse, or touchscreen, there is a delay built into the response due to the nature of the hardware switches that are processing the response. It is important to note that the magnitude of the delay, typically around 12 ms for a keyboard, may be so small when viewed in the context of the typical standard deviation for average reaction times, typically over 100 ms, as to be acceptable. See the timing test results reported in Appendix A of this manual for a more complete presentation of our timing test results with keyboards and mice.

If the delays that are built into the responses that are sent by the keyboard, mouse, or touchscreens meet your experiment requirements, then any of these devices can be used with E-Prime 2.0. See Chapter 3 of this manual for accepting input and logging data from a response device. However, if your paradigm requires more precise timestamps of participant responses than what is available from a keyboard, mouse, touchscreen or joystick, then refer to Sections *4.6.2 Experiment Advisor (Page 114)* and *4.4.4 Response Collection Solutions (Page 110)*, which presents alternatives to the keyboard and mouse for collecting a response.

4.3 The Timing of E-Objects

E-Prime 2.0 provides multiple tools to enable you to check the critical timing sequences in your experiment. Effective use of these tools, however, requires that you first understand the process by which E-Prime 2.0 objects execute. You will then be able to understand the dependent variables corresponding to the steps in this process which you can choose to log in your data file. This section illustrates the major steps in the execution of a series of E-Prime 2.0 objects by examining an increasingly complex sequence of events, and then defines the variables that log the timestamps associated with these steps. The final section of this chapter illustrates the tools that enable you to work with the variables of interest to confirm the timing of the critical experimental sequence.

4.3.1 Object Execution: a simple overview

An object's execution occurs from its StartTime until its StopTime. OnsetTime is defined as the timestamp, in milliseconds, from the start of the experiment to when the object's onset action, in this case drawing the display to the screen, begins. OffsetTime is defined as the timestamp, in milliseconds, from the start of the experiment to when the object's offset action, in this case stop execution, begins. The duration of an object's execution is the difference between the OffsetTime and the OnsetTime. Figure 5 below illustrates the TimeAudit variables for a single object executing in Event Mode.
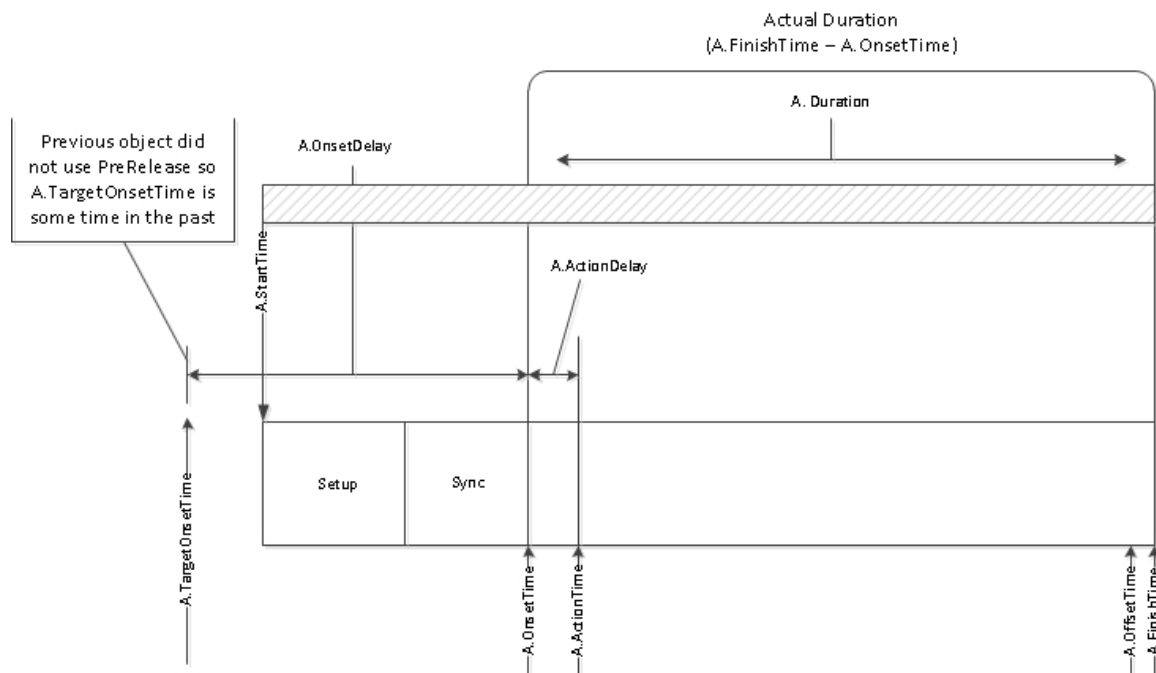

Figure 5. Simplified timing diagram for an E-Prime 2.0 Object

OnsetTime is a critical item to understand, so we'll examine it a bit further here. For a display object, the OnsetTime provides a timestamp of the exact time at which the display object begins to draw. Monitors are always drawn starting with the upper left-hand corner of the monitor, and OnsetTime reflects this event. In other words, OnsetTime is the time immediately after the vertical refresh occurs and the draw command begins to execute. Note that this process describes how the monitor operates; E-Prime 2.0 can also handle special situations where the video cards provide information in a different manner.

E-Prime 2.0 can NOT tell you when each component of the display is placed on the screen. (Note that we are describing the typical behavior for CRT and LCD monitors; DLP and other styles of monitors may behave differently.)   For example, consider a fixation cross in the center of the screen. If the monitor has a refresh rate of 60Hz, then it takes 16.6666 ms for all of the pixels that comprise the display to have their information updated. In the case of a CRT display, this means refreshed with the electron gun; in the case of an LCD display, this typically means resetting the video hardware to prepare for the next set of data. Therefore, any information that is displayed in the center of the screen

would appear approximately halfway through the screen update or approximately 8 ms after the object's OnsetTime. As a final note about this simplified view of object execution, notice that the actual Duration that is shown in the figure is defined as the difference between the OffsetTime and the OnsetTime. This is NOT the same value that is logged in the data file as <object>.Duration. The Duration variable that is logged in the data file is the requested duration that is specified in the Duration field on the Duration/ Input tab. However, E-Prime 2.0 provides an easy way to obtain a close approximation to how long a stimulus appeared on screen. The OnsetToOnset time, which indicates how much time elapsed from the Onset of the current object to the onset time of the next object, can be logged in the data file by utilizing the TimeAudit feature (but see *Section 4.6.1 TimeAudit (Page 113)* for important cautions regarding the interpretation of this value).

4.3.2 Object Execution: a more complete description
A more complete picture of object execution, reflecting important actions prior to an object's OnsetTime, is shown in Figure 6. This example continues with Event Mode timing.
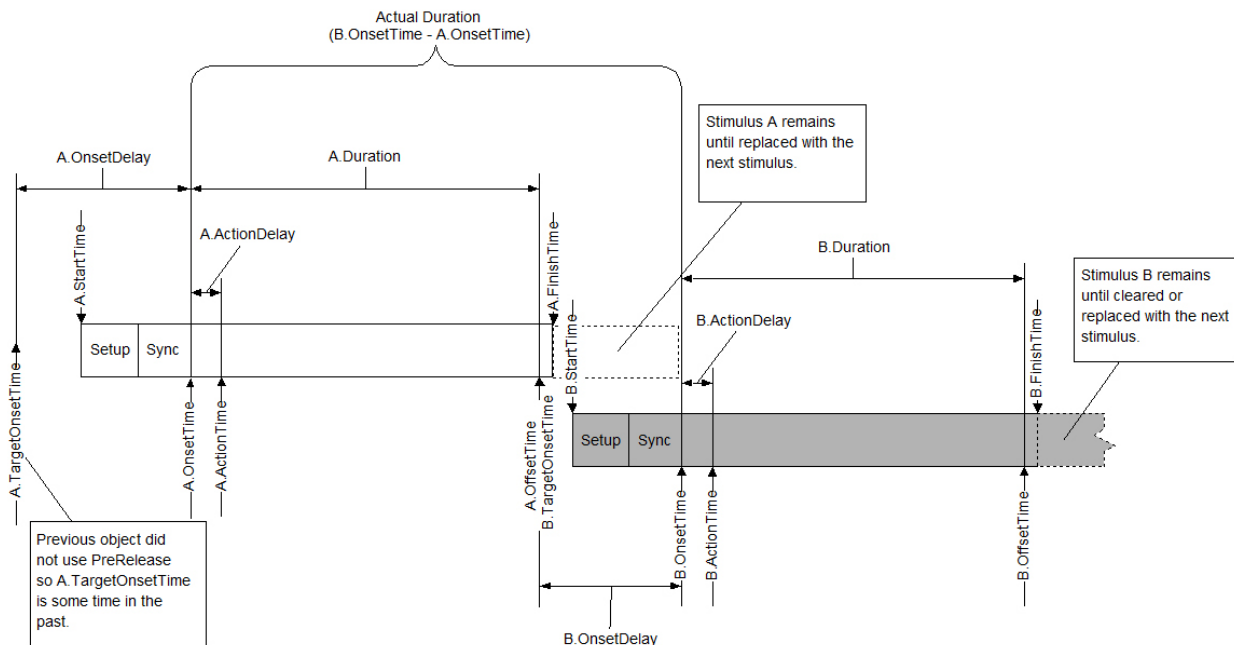

Figure 6. Event Mode with no PreRelease

Figure 6 introduces several new object properties: ActionDelay and ActionTime:
As was stated above, the OnsetTime is the timestamp at which the object begins its action. For a visual display object, it represents the time at which the object begins to execute its draw command; for a SoundOut object, it is the time at which the object begins to play the sound file. More precisely, the OnsetTime represents the timestamp at which E-Prime 2.0 passes off execution for stimulus presentation to the appropriate hardware's drivers (e.g. the video or sound card) and to the operating system. The ActionTime represents the timestamp at which the hardware's driver returns control back to E-Prime 2.0.

Typically, the difference between the OnsetTime and the ActionTime will be less than 1 millisecond. Any cases where they are not within a millisecond of each other may signal a potential problem with your hardware device. Therefore, when you check your pilot data, include an evaluation of ActionDelay, to ensure that no significant delays are occurring.

OnsetDelay and TargetOnsetTime: OnsetDelay is the difference between the TargetOnsetTime and the OnsetTime. The TargetOnsetTime is a timestamp calculated internally by E-Prime 2.0 at the conclusion of the prior object. With a discrete trial paradigm, OnsetDelay should be as small as possible[5].

[5] In some cases, OnsetDelay can be negative. This can occur when using RefreshAlignment. See Section 4.4 for details.

4.4 Techniques to obtain the most accurate timing possible

E-Prime 2.0 provides a two-fold approach to obtaining the most accurate timing for your experiment. First, to obtain the best performance possible, E-Prime 2.0 provides techniques to obtain the fastest stimulus presentation and most accurate response collection possible. Some of these techniques are embedded within the E-Prime 2.0 architecture and are available to you automatically; other techniques are user-configurable. Second, E-Prime 2.0 enables you to log detailed information about the timing characteristics of the trial events. This in turn enables you to flag trials with potentially problematic timing for more careful analysis; such trials can then be excluded, if necessary, from your data analysis. This section describes E-Prime 2.0's solutions to timing challenges, and *Section 4.5 E-Prime Best Practices (Page 111)* describes the tools you can use to confirm your timing.

4.4.1 Operating System Solutions

E-Prime 2.0 attempts to minimize all interactions with the operating system. Although we cannot deliver measurements every millisecond, our methods make delays and/or missed reads of the clock tick very rare, and make sure that the results are statistically valid and accurate. Appendix A reports the results of the Refresh Clock Test on a mid-range and high-end computer, in which there were 0 missed ticks.

Priority
The first technique that E-Prime 2.0 utilizes is to tell the operating system to execute the E-Prime 2.0 program (E-Run) in the highest priority possible when collecting data. E-Prime 2.0 takes into consideration a number of factors when determining the proper priority level for your system.

PST's testing shows that performance on either dual-core or multi-processor systems is enhanced over single processor systems. When running with more than one core or processor, it is less likely that E-Run will be interrupted. However, while such interruptions are minimized, they are not wholly eliminated. In addition, concerns remain over access to and performance of peripheral devices for stimulus presentation. Therefore, even when E-Prime 2.0 runs on a dedicated process, you still must utilize E-Prime 2.0's other solutions in order to optimize the performance of the time-critical components of your experiment.

⚠️ **NOTE:** *One significant action that users can take, however, is to remove any unnecessary programs running on the E-Run data collection computer, including active network or internet connections. We recommend using Microsoft's MSCONFIG utility to identify programs which may be running in the background and then shutting down any that are not necessary for system's operation.*
See the Knowledge Base article: [KB 2621](#) INFO: How to use MSCONFIG to troubleshoot machine configuration and reduce background applications.

Timing Mode
Timing delays will occur in experiments due to the needs of the operating system, the time required to prepare stimuli, and the limits of the hardware systems used to preset stimuli. While this section describes the numerous techniques that E-Prime 2.0 offers to reduce timing delays to the greatest extent possible, such delays cannot be completely eliminated. Therefore, it is important for you to decide how E-Prime 2.0 should respond when these delays occur. E-Prime 2.0 offers two timing modes to answer this question.

Consider the following example: assume that you want to present a sequence of five displays, with each display appearing for 100 ms each. Further, assume that after display #1 occurred, the experiment paused for 20 ms while a bitmap was loaded. This introduces a timing error. The first display is presented not for 100 ms as requested but 120 ms, because during the 20 ms delay the display #1 continues to appear. As a further consequence, the second display will not begin until 20 ms after it was expected, i.e. it will start 120 ms after the first display appeared rather than the 100 ms that was originally specified.

What should be done about this error? Should E-Prime 2.0 present the second display for 100 ms, thereby delaying the onset of the other three subsequent displays? Or, should the duration of the second display be reduced in order to compensate for the delay and keep all subsequent displays on time? E-Prime 2.0's timing mode feature allows the user to determine how to handle timing delays such as these. We discuss each mode in detail below, but in general Event mode is used for most discrete trial paradigms where there is no communication with additional computers. Cumulative mode is often used for fMRI studies and other studies which must synchronize E-Prime 2.0 with a scanner.

In Event mode, delays in the onset of an event will not affect the specified duration of the event. This results in a delay of the onset of all subsequent events due to the error, and an accumulation of timing delay across events. Continuing with our example, if there was a timing error of 20 ms before the second stimulus, and a 30 ms delay before the fourth event, the durations of all four events would be 100 ms but the relative start times would be at 0, 120, 220, 350, and 450 ms.

⚠️**NOTE:** *The delay of the second display both lengthens the effective duration of the first display, from 100 to 120 ms, and also changes the start time of the second display.*

Further, the last display of the five "100 ms displays" actually occurs at 450 ms from the first display rather than the expected 400 ms due to the cumulative timing error. These effects are illustrated in the top half of Figure 7 (Event mode).
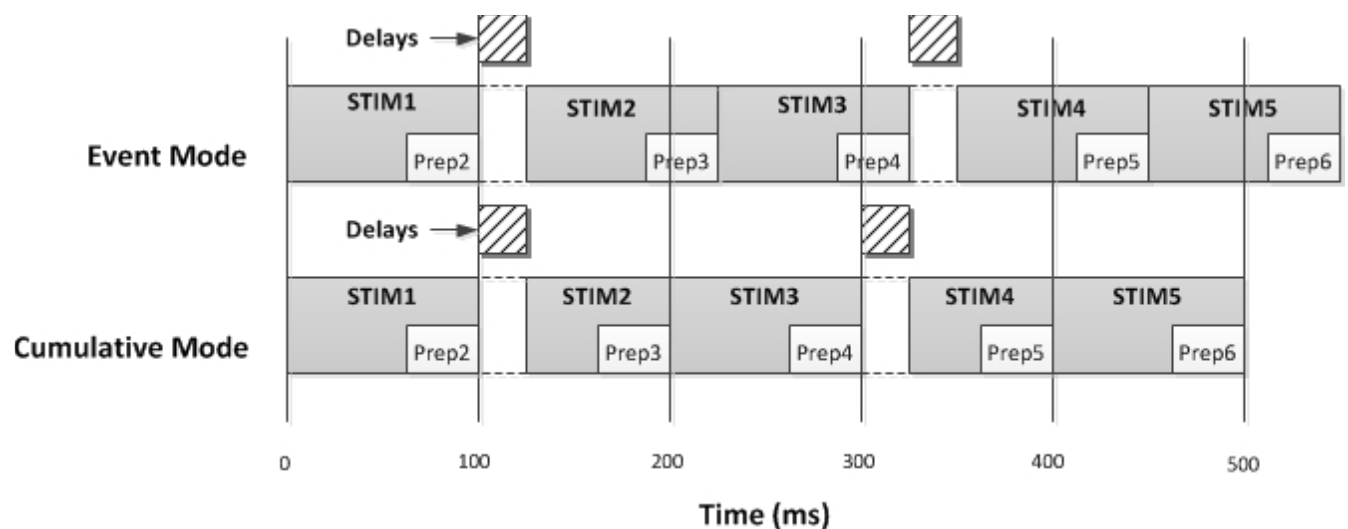

Figure 7. The Effects of Event and Cumulative Timing Mode on a stimulus presentation

In Cumulative mode, delays in the onset of an event result in an equivalent reduction in the duration of the event, such that the cumulative timing error is minimized. For our example, the delay of 20 ms before the second event would cause a shortening of the duration of the second event by 20 ms. Thus, for Cumulative mode timing, delays of 20 ms and 30 ms before the second and fourth events result in onsets of 0, 120, 200, 330, and 400.

⚠️**NOTE:** *The delay of the second display does not change the start time of the third display, but rather changes the duration of the second display.*

Even with two delays, the fifth display occurs at the expected onset time of 400 ms after the first event. These effects are illustrated in the bottom half of Figure 7 (Cumulative mode). Notice, however, that with cumulative mode, the first object executes as in Event mode, and then all subsequent objects are run in cumulative mode.

There are a few scenarios that can occur in an experiment which can temporarily defeat or limit the effects of cumulative timing mode. First, if an unexpected timing error or delay is sufficiently long enough (or equally if the duration of the stimulus presentation is sufficiently short enough) there may not be enough time available during the presentation of the current stimulus to entirely "absorb" the error. If this happens, any number of subsequent presentations may likewise have their durations shortened until all of the cumulative error is accounted for. For example, a stimulus presentation event with a specified duration of 14 ms cannot possibly shorten its duration to account for a delay of 20 ms. In this case, the current stimulus will shorten its duration as much as it possibly can and then subsequent stimuli will attempt to follow suit until all of the timing error is absorbed and the presentation sequence eventually gets back in sync with the expected stimulus onset times.

Second, there may also be an interruption in the timing sequence if a stimulus is terminated by a participant response (e.g., one of its input masks has its End Action property set to Terminate). When this occurs, E-Prime 2.0 assumes the user wants the next stimulus to occur as soon as possible after the current event is terminated; otherwise, the End Action property would have been set to (none). E-Prime 2.0 handles this case by instructing the next stimulus object to set its onset time to the time of the response rather than the onset time it was originally expecting. Note that this has the same effect as temporarily switching to Event mode timing for the current object and then resuming the Cumulative timing sequence. For this reason, stimulus presentations should NOT be terminated by response events (e.g., if an input masks End Action is set to "(none)" then the response will be accepted, scored and logged as expected but the stimulus sequence will not be interrupted by the collection of the response).

Event mode and Cumulative mode studies are used in two different classes of paradigms. If the goal is to present a single stimulus, or to present short sequences of stimuli (e.g., fixation, probe, mask) where the inter-stimulus interval may vary, then you should use Event mode. If the goal is to maintain a constant rate of presentation (e.g., a memory task presents stimuli every 2 seconds without cumulative error or drift), you should use Cumulative mode. The default timing mode for all objects in E-Prime 2.0 is Event mode. When using event and cumulative mode within the same experiment it is important to make sure the timing works as expected by running the experiment and examining the data.

4.4.2 Stimulus Preparation Solutions

As we discussed in *Section 4.1 Introduction (Page 87)*, the process of preparing a visual stimulus to be displayed or a sound or movie to be played requires some preparation time. This preparation time can be significant: for visual stimuli, delays can be more than half of a refresh rate, and for auditory stimuli, delays can be in the tens of milliseconds. E-Prime 2.0's stimulus preparation solutions are designed to reduce these delays, and/or move as much of it as possible to a non-critical interval in the trial, such as before the presentation of the key trial sequence.

PreRelease
In the checkerboard example that we examined at the start of this chapter, we saw that stimulus setup time for image files added about 36 ms to the median display time. To correct the problem, the next display must be setup while the participant is looking at the current display; the time-consuming reading and loading of the next image file needs to occur before it is time to present the image.

E-Prime 2.0 introduces the concept of using PreRelease time to address this problem. This feature allows the current stimulus to release a portion of its execution time to a following stimulus in order to allow the following stimulus to perform setup activities (e.g., read the data from disk and get the image ready in memory). Then, at the scheduled time for the following stimulus to be presented, the display is already prepared and it is simply copied to the screen. By overlapping the setup time for the following display with the duration of the current display, the delay caused by the setup time can be reduced and potentially eliminated. If the PreRelease time allotted is greater than the setup time required, the images can be loaded and presented without delay. In the checkerboard example, a PreRelease time of 50 ms would provide ample time for setup, and there would be no added variability.

E-Prime 2.0 introduces a new default value for PreRelease, "(same as duration)". By accepting this value, you do not need to estimate or experiment with different values of PreRelease to find the optimal value, and you provide the system with as much time as possible to prepare the next display object.

An important aspect of how PreRelease works is that the stimulus presentation and response processing occur in parallel. This means that although the next stimulus may be prepared while the current stimulus is still being presented, any responses entered prior to the actual display of the second stimulus will be scored according to the settings of the current stimulus (i.e., the response is always scored by the stimulus prompting the response).

Let's review the checkerboard example. The results from the example that was programmed in E-Prime 2.0 but without properly using the timing tools showed that the average the preparation or setup time approximately 36 ms. So, after 36 ms the first stimulus was presented to the participant. When PreRelease is used, the second stimulus is selected and prepared for presentation while the first stimulus is still being presented. After 72 ms has elapsed, the second stimulus is prepared and ready to be displayed. The program then waits the remainder of the specified duration for the first stimulus (200 ms from the onset of the first display at 36 ms), and then presents the second stimulus (at time 200 ms). At this time, the third stimulus is prepared while the second stimulus is being displayed. All the stimuli would then occur with the intended 200 ms interval. The displays would occur starting at 36, 236, 436, 636 ms, etc. From the participant's point of view, the experiment would present a stimulus every 200 ms with no apparent setup time or cumulative timing drift.

Figure 8 shows a diagrammatic representation of the timing with and without a PreRelease time specified. With PreRelease, as long as the preparation time is less than the display time, there will be no added delays. Again, using the default value of "(same as duration)" ensures that the maximum amount of time is made available to E-Prime 2.0 to prepare the next display.
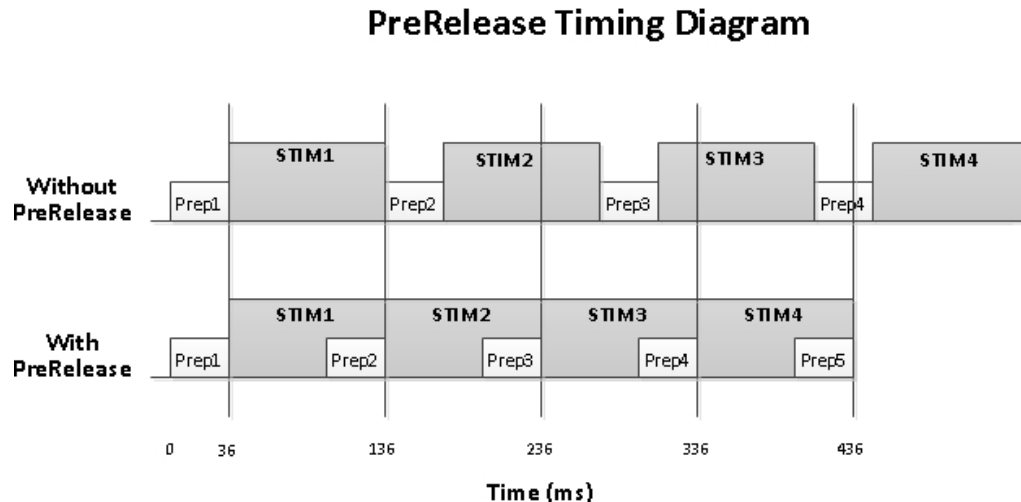

Figure 8. Timing diagram of sequential stimulus displays contrasting the use of PreRelease with no PreRelease.

The proper use of PreRelease also substantially corrects for the second timing problem, introduced in *Section 4.1.1 What are the challenges?*, that actual display durations can be significantly longer than the intended durations. Recall that Windows can arbitrarily halt the program for periods of tens and sometimes hundreds of milliseconds. If one of these operating system delays occurs during the critical sequence of "terminate the last stimulus, generate the next stimulus, and present the next stimulus," it could potentially produce delays in what the participant sees. When PreRelease is used, E-Prime 2.0 accomplishes as much work as possible, as early as possible (i.e., in some cases as soon as the first object performs its critical action, time can be released to the next object). It also prepares the display information in an off-screen bitmap so it is ready to go, and unlikely to be touched by the operating system. The code for monitoring when to display the stimulus is in a tight loop to keep execution short

and keep the code in memory. If the operating system takes control and steals time that is less than the PreRelease time, the tight code segment of putting up the display will not be affected. Although the operating system can still stop the tight code loop at presentation time, the probability of such an event is greatly minimized by making the final critical actions brief.

While the PreRelease mechanism is a powerful and useful tool, there are a few caveats when using PreRelease of which you should be aware. First, PreRelease provides extra processing time for the next object in the Procedure; it does not provide any additional processing time for the object on which it is enabled. Therefore, there is typically no benefit to using PreRelease on the last object in the procedure, unless you:

• Have GeneratePreRun set to TopOfProcedure
• Are not playing a movie or audio file on the last object of the procedure (otherwise, PreRelease on this last object stops the playing of the audio file or the movie)
• Are not scoring data from responses that have a time limit set to the end of trial (this could cause data loss because the trial would end and be in the process of prepping the next trial before the response is able to be registered)

When all of these conditions apply, then PreRelease on the last object can provide a significant benefit.

Secondly, you should not specify a value of PreRelease that is greater than the object's Duration, which includes the case where the object's Duration is set to 0. Unintended timing problems may arise when PreRelease is greater than the object's Duration. Generally, however, the default value of "Same as duration" works best is most situations, and eliminates the need for careful testing of a range of values to find the best one.

PreRelease and Input Masks

There are three potential conflicts that can arise with the default value for PreRelease. First, if an object is using PreRelease and the next object is a FeedbackDisplay, then the statistics shown to the participant during the experiment run could be inaccurate because the object that pre-released may still be accepting responses. To assist with this situation, use the FeedbackDisplay.ProcessInputObjectPendingInputMasks property. By default, this property (accessible from the General Tab on the FeedbackDisplay object) is selected, as shown in Figure 9.
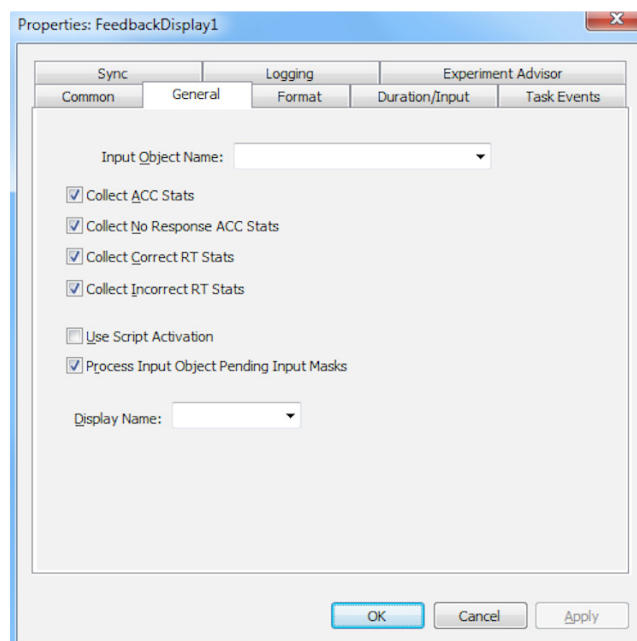

Figure 9. Enabling the Process Input Object Pending Input Masks

When this property is selected, then the scoring script that is executed as part of the Feedback Display is not performed until all of the input masks for object have been terminated, either by timing out or from receiving the maximum number of responses.

There is a similar concern if the next object that appears after an object using PreRelease is an InLine or PackageCall. It can't be assumed that the prior object has been cleared or is no longer collecting responses from input masks. In this case, script should be used to check the status of any pending input masks. See the Knowledge Base article: KB 2627 FEATURE: PreRelease defaults changed to promote better timing accuracy, for details on how to implement this check.

A final concern is when the very last object on a Procedure has PreRelease. In this case, it will release into the final logging of the Procedure and select the next exemplar for the trial. The Procedure. ProcessPendingInputMasks will assist with this.

In summary, the FeedbackDisplay.ProcessInputObjectPendingInputMasks property and the Procedure. ProcessPendingInputMasks property enable users to routinely accept the default value for PreRelease ("same as duration") without having to add special script to handle these cases. In the event that you would need to handle these cases, script such as that Sample Process Template, can be used.

GeneratePreRun

As we discussed in *Section 4.1 Introduction*, the process of loading movie, image and sound files is a time-consuming one. For example, in the checkerboard experiment, the time that was required to prepare each graphics file was an average of 36 ms, and most of that preparation time involved reading the file from the hard drive and loading it into video memory.

Internally, E-Prime 2.0 utilizes two separate steps to present sound, images or movies. The first step is performed by the object's Load method, and results in the media file being read from the hard drive into memory. The second step occurs when object's .Run method executes and results in the presentation of the media file. In E-Prime 2.0, a media file cannot be presented without first being loaded into memory, in part because reading directly from the disk is prohibitively slow.

E-Prime 2.0 introduces a new property, GeneratePreRun, which determines when media files are loaded. This feature, which is available on all RteRunnableInput objects (i.e. Text, Slide, ImageDisplay, MovieDisplay, SoundOut) enables you to direct the time-consuming process of loading of a stimulus file from disk into memory to occur prior to the critical timing sequence in the trial. To do so, GeneratePreRun should be set to "TopOfProcedure", which is the default value. With this setting, E-Run will load any sound, graphics, or movie files that are set from an attribute before the object which is presenting the media file begins to execute. More specifically, the <object>.Load method is placed in the Procedure_Start section of script, prior to the start of the object execution. This is illustrated in Figure 10 below, left hand panel.
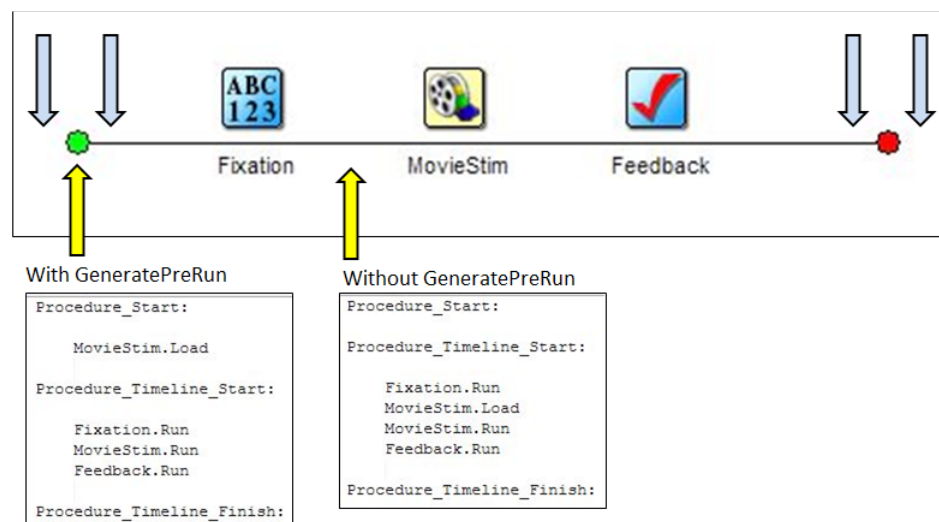

Figure 10. Object.GeneratePreRun's effect on the .Load method

Conversely, when GeneratePreRun is set to "BeforeObjectRun", the <object>.Load method is placed directly before the <object>.Run call. As a result, the media file will not load until the object's .Run method is called. In this case, the time required to load the media file is reflected in the object's OnsetDelay.

The GeneratePostRun feature is the mirror/opposite effect, but is not currently utilized by any Toolbox objects in E-Prime 2.0. GeneratePostRun is made available for PackageCalls when necessary.

⚠ **NOTE:** *When using the TopOfProcedure GeneratePreRun option, take care when using [attrib] references for any properties on the objects that are modified by c.SetAttrib calls on the Procedure. Doing so will successfully update the value of the attribute in the Context, via InLine objects on the Procedure but since the object has already performed its .Load operation, it will not take into account for the new value.*

4.4.3 Stimulus Presentation Solutions

Section 4.2.3 Stimulus Presentation Challenges described the constraints that the refresh rate and the vertical blank event impose on the ability to precisely control display durations. The first five solutions described in this section, both individually and collectively, work with the physical constraints of the video card and monitor to control, as precisely as possible, the duration of stimulus displays. The features that are examined here are all set on the Display Device dialog box and are shown in Figure 11. To view these features within E-Studio, select "Experiment" from the "Edit" menu, select the "Devices" tab, and then select "Edit" with the Display selected.
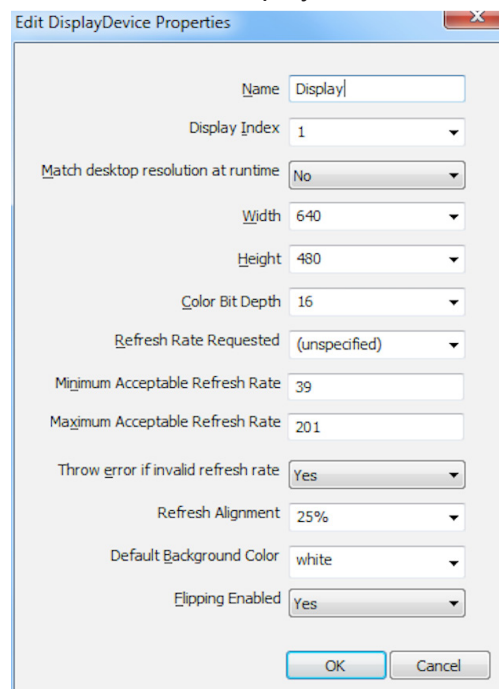

Figure 11. Display Devices Properties

Set the display resolution
The refresh rates that are available on a given system are determined by both the monitor and the video card. The display resolution also affects the refresh rates that are available. While most low- to mid-range video cards and monitors will be able to run in the highest resolution at the highest refresh rate supported by the card and monitor, there may be some combinations that are not supported. For higher-end video cards and monitors, there is a greater likelihood that some combinations are not supported. Therefore, the first step in selecting the display properties for your experiment is to select the screen resolution.

To set the screen resolution, navigate to the Edit Display Device dialog box as described above and select values from the Width and Height dropdown boxes. The default screen resolution is 640 x 480,

which is a lower resolution for graphic files. However, it was deliberately selected because it is almost universally supported across video cards and monitors. Further, you should use the lowest screen resolution possible. For example, there is no reason to run in the HD resolution of 1900 x 1080 if you are only displaying a text-based cross fixation that takes up a 10 by 10 pixel area.

There are several important cautions and notes regarding the display resolution. First, if you are using a graphics package to create visual stimuli, set your computer to the screen resolution to be used while running the experiment before creating or editing your stimulus files. Select a screen resolution that is fairly common and supported by the majority of computers. Lastly, test each laboratory computer to confirm that it supports your specified screen mode. At this point, once you have confirmed the screen resolution on the data collection computer(s), you can create or edit the graphic files in your editor of choice, such as Adobe Photoshop.

If your set of stimulus files were not generated or saved in a common resolution, then we strongly recommend that you standardize them in a third party graphics package. While the size and appearance of graphic files can be manipulated somewhat within E-Prime 2.0 by manipulating the Frame settings, such as Stretch, StretchMode, MirrorLeftRight, and MirrorUpDown, this practice is not recommended. Any image manipulation that is performed by E-Prime 2.0 introduces some timing delay, which will be reflected in the OnsetDelay for the object that is presenting the image.

Also, if you are running on a wide screen monitor and use a 4:3 display resolution, then the screen will either show black bars on the sides, or the image will stretch to fill the screen. This is a function of the video card and wide-screen monitor and is not configurable by E-Prime 2.0. Therefore, you should select your aspect ratio first, prior to generating any visual stimuli.

We generally recommend that you do not match the desktop resolution at runtime. The "match desktop resolution" feature was introduced in E-Prime 2.0, and its default value is not selected. However, there is one scenario where you may want to have E-Prime 2.0 match the monitor to the desktop resolution: when you have multiple display devices active for a single experimental session. E-Prime 2.0 Professional introduces the ability to add additional displays and direct output to them[6]. In some cases, E-Prime 2.0 has difficulty determining the proper display settings for the additional display devices. When you are creating the additional display devices, you can enable the "Match desktop resolution at runtime" feature. E-Prime 2.0 will then take the resolution being used when the experiment is launched and set the designated display device to that resolution.

Confirming the available Refresh Rates
Section 4.2.3 Stimulus Presentation Challenges described the need to present stimuli for integer multiples of the refresh cycle of the monitor. In order to set your display duration to a multiple of the refresh rate, you need to know the refresh rates that are supported by your system.

E-Prime 2.0 can and will get the refresh rate that you specify in the DisplayDevice properties. This value may or may not match what is configured through Windows control panel. E-Prime 2.0 logs the refresh rate in the data file for each experiment under the variable name Display.RefreshRate. (If running with two monitors, the second refresh rate value is logged as Display2.RefreshRate.)   After specifying the refresh rate in E-Studio (explained in the next section), you should always run a test experiment, examine the resulting data file in E-DataAid and confirm the value logged in the E-Prime 2.0 data file. Figure 12 shows a typical result from a 60 Hz refresh rate.

---

[6] Note that this paragraph is referring to E-Prime 2.0 Professional only. E-Prime 2.0 standard introduced the ability to specify more than one monitor for each display device; this can be useful, for example, when running an experiment from a laptop and directing the output to a CRT monitor that is connected to the laptop. With such a scenario, you still do not want to match the desktop resolution.

Figure 12. Checking the Refresh Rate in E-DataAid

Notice that the example shows a logged value of 60.011 rather than the integer value of 60 that was specified in E-Studio. While refresh rates are often reported as whole numbers by the vendor, they are rarely the exact value reported. For example, a monitor running at 60 Hz may actually be running at 59.98 or 60.02 Hz. These small variations are within the fault tolerances of the manufacturer, and do not pose any problems for accurate display presentations provided that you understand that and make your durations divisible by this refresh duration value. The upcoming sections on specifying the display duration and using RefreshAlignment explain how E-Prime 2.0 works successfully with non-integer refresh rates.

Set the Refresh Rate
The Refresh Rate is specified on the Display Device properties tab. The default value is "unspecified". If you click on the "Refresh Rate Requested" field, a list of common refresh rates appear. As the name of this field suggests, E-Prime 2.0 treats this rate as a request. E-Prime 2.0 attempts to set the video system to this rate. If it cannot set it to either this rate or a value between the "Minimum Acceptable Refresh Rate" and the "Maximum Acceptable Refresh Rate", and if the "Throw error if invalid refresh rate" is set to "Yes", E-Prime 2.0 will generate a runtime error. Unless you are developing and/or running your experiment in virtualization mode, which is not recommended for data collection and which is also explained below, you should keep the default value for "Throw Error".

The "unspecified" value is appropriate to use when you are in the early stages of experiment development in E-Studio. However, as soon as you confirm what the refresh rate is on your laboratory experiments , you should set this rate here, as well as design the display durations around the specific refresh rate in your data collection lab (e.g. display durations should be divisible by the refresh rate).

A note about virtualization mode
When E-Prime 2.0 is running on a system where video hardware is emulated (such as Standard Display Adapter in Vista) or under virtualization software, which is typically used to mimic an operating system, E-Prime 2.0 cannot reliably detect the refresh rate. The Display Device Property "Throw error if invalid refresh rate" controls E-Prime 2.0's behavior in this situation. When set to the default value of "yes", E-Run generates a fatal error and does not allow the experiment to run.

⚠ **NOTE:** *E-Prime 2.0 cannot maintain accurate timing when run under virtualization software.*

However, in some circumstances it may be convenient to develop and test an experiment with a computer that is running virtualization software while the actual data collection will occur on machines that are running under native Windows mode. Alternatively the experiment may not use reaction time as the dependent measure. In these circumstances, the error can be turned off by switching the property to "no".

Set the Display Durations to be multiples of the Refresh Rate
⚠ **NOTE:** *The display durations that you can use for your experiment as a function of your refresh rate.*

We've already examined examples where a 60Hz refresh rate equates to displays that are multiples of 16.6666 ms. Other than monitors with a refresh rate of 100Hz, which equates to display times in multiples of 10 ms, the refresh rate will not be a whole number. Since E-Prime 2.0 requires that durations be defined as integer values, then you will end up with some display durations that are an extra refresh duration. The RefreshAlignment feature was designed expressly for this purpose, to enable E-Prime 2.0 to start looking a bit earlier than indicated by the display duration for the refresh event. See the Knowledge Base article: KB 3027 FEATURE: RefreshAlignment locks into nearest refresh vertical blank to promote timing accuracy.

Synchronize to the vertical blank event
All E-Prime 2.0 stimulus presentation objects allow synchronization with the refresh cycle via the object's properties. The Sync tab located in the stimulus presentation object's (e.g., TextDisplay, ImageDisplay, SlideDisplay, SoundOut, etc.) Property pages allow the user to specify whether the onset time for the next stimulus should be delayed until the next vertical blanking event. When the Onset Sync property is set to vertical blank, the object will delay until the next vertical blanking before presenting the stimulus. Note, the second stimulus is not starting at an integer multiple of milliseconds but rather as soon as the vertical blanking event occurs.

When the synching signal (i.e., vertical blanking) is detected, the next event considers this signal as the actual onset time of the stimulus, and processes the duration from this start time rather than from the end time of the previous stimulus. This is necessary in order to stay synchronized with the vertical blanking events. If timing does not re-synchronize with the actual vertical blanking, the display timing and display updating will drift and cause display tearing where part of the previous screen stays on the screen while only a portion of the new screen occurs. When using Display.FlippingEnabled, the Sync tab is required to be set at vertical blank.

Using Refresh Alignment
As we've just described, display duration times should be specified as multiples of the refresh rate, particularly for display durations that are less than 100 ms. However, it is often not possible to enter a display duration that is divisible by the refresh rate, for two reasons. First, a 60 Hz monitor refreshes the screen every 16.6666667 ms, but E-Prime 2.0 requires you to specify display durations as non-decimal values. Therefore, in the case of a 60Hz monitor, the display duration would have to be specified as either 16 or 17 ms. Similarly, the refresh rate itself is rarely a whole number. For example, a monitor that we commonly refer to as running at 60Hz isn't running at exactly 60Hz; instead it is likely running around 59.98 Hz or 60.02 Hz. These differences are considered to fall within the acceptable hardware tolerances of the hardware manufacturer.

Eventually, you will encounter a scenario where the next scheduled display time occurs just after a refresh occurs, and therefore the system will have to wait another full refresh of the screen before the next display can be written to the screen. Let's consider a specific example with a Refresh rate of 60 Hz. The experimental sequence involves two displays, Display1 and Display 2, each with a duration of 1000 ms. Assume that Display1 draws at time 1234; Display is then scheduled to appear at time 2234. Assume further that the next refresh is scheduled to occur at time 2233. Without the use of the RefreshAlignment feature, Display2 will not display until time 2249 (2233 + 16), which makes Display1 on the screen effectively for 1015 ms as shown in Figure 13.

RefreshAlignment instructs E-Prime 2.0 to start looking for the refresh event earlier than what is indicated by the current display object's duration. It is set on the Display Device properties page in E-Studio. RefreshAlignment is specified as a percentage of the refresh rate, and the default value is 25%. Therefore, on a 60 Hz monitor with 16.6667 ms per refresh, RefreshAlignment will being looking for the refresh event up 4.1555 ms early. In the case of the current example, E-Run will start looking at time 2229, and therefore detect and respond to the refresh event which occurs at time 2233.

This makes the effective duration of Display1 = 999, and Display2 starts within one ms of its target value. If RefreshAlignment were not be utilized, then Display1 would have an effective duration of 1016, because it would be displayed for an additional refresh after E-Prime 2.0 missed the early refresh, and Display2 would be displayed 16 ms after its target value.

This example assumes that RefreshAlignment is being used in conjunction with Event mode. With RefreshAlignment and event mode, the next TargetOnsetTime is adjusted after an early refresh is detected. Conversely, when using RefreshAlignment in Cumulative mode, the next target onset time is not adjusted. As a result, Display1 is displayed for a shorter amount of time (999 ms, the same as in event mode), but Display 2 is lengthened.

In Figure 13, the black solid line at time 998 shows when a Refresh will occur. The gray line at time 1000 shows the next target onset time. The black/gray line shows when the next refresh after 998 would occur, assuming that you are running at 60 Hz. With RefreshAlignment on (see top panel), the object that is targeted to start at 1000 begins 2 ms sooner to catch the refresh at time 998. However, since we are in cumulative mode, the next target onset time is not changed, and as a result Display2 then runs for 2 ms more. The effective error on two objects is -2 ms and +2 ms.

With Refresh Alignment (Cumulative)

Duration = 998, Error = -2   Duration = 102, Error = +2

998 1000   1014   1100

Without Refresh Alignment (Cumulative)

Duration = 1014, Error = +14   Duration = 86, Error = -14

998 1000   1014   1100
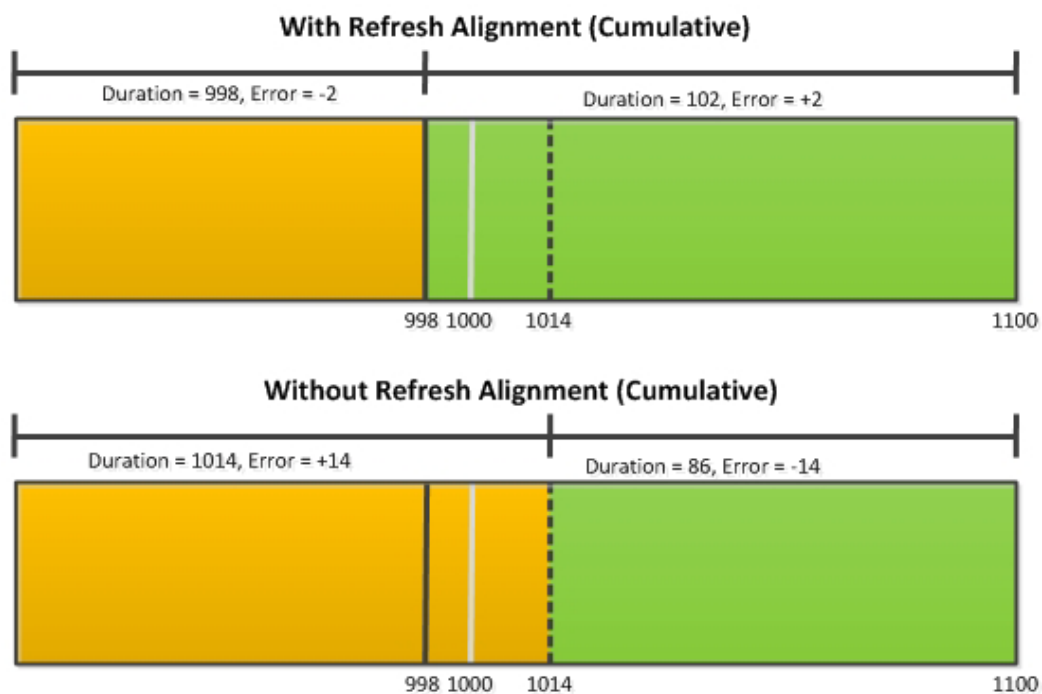
Figure 13. The effect of Refresh Alignment

If the same experiment were run in cumulative mode but with RefreshAlignment off, then the object targeted for 1000 ms has to wait until 1014, since the early Refresh at 998 was missed. Therefore, Display 1 appears for one extra refresh. Since we are in cumulative mode, the duration of Display2 is reduced by one refresh. The effective error in this example is +14 ms for the first display and -14 ms for the 2nd display. Notice that with the user of Refresh Alignment, negative OnsetDelays will occur and they are acceptable – a negative OnsetDelay does not indicate a problem in the trial sequence unless you get them for a large number of trials which might indicate that the Duration is not divisible by refresh rate.

Summary, Display Durations, Refresh Rate and synchronization
The combination of specifying a reduced display duration and synchronizing with the vertical blanking allows the experiment displays to be accurate even with minor timing variations in the refresh rate. It is important to note that even using the same model display card on two identical computers may result in slightly different refresh rates. Hence, a 50 ms duration on one monitor might result in a period of 49.9 to 50.1 ms across cards.

If Event timing mode is in use, the actual duration for which the participants sees a stimulus is equal to

the Duration specified for the stimulus object plus the OnsetDelay of the next object that alters the visual location of the first stimulus. For example, if the duration for the first stimulus was set to 90 ms, and the next stimulus had an onset delay of 10 ms, the true duration of the first stimulus would be 100 ms.

Flipping

E-Prime 2.0 offers the ability to use Display flipping to perform any drawing operations. The use of flipping is recommended especially for larger displays to reduce the possibility of display tearing (when part of the new/old screen appears at the same time). Use of flipping requires a PreRelease value of approximately one refresh duration for the object that precedes the drawing object to ensure proper scheduling. Without ample PreRelease, the object will schedule for the following refresh period. This would be reflected in the OnsetTime and OnsetDelay properties.

⚠ **NOTE:** *When flipping is enabled, all E-Basic Canvas calls targeting the on-screen canvas will internally wait for the vertical blank refresh period. The use of Display.WaitForVerticalBlank will thus not be necessary and if used would cause the Canvas operation to draw a refresh later. Whether using flipping or not, it is recommend when performing any drawing commands to do so on an off-screen canvas and then use one Canvas.Copy operation to update the screen.*

Sending output to multiple monitors

Researchers are sometimes interested in sending output to more than one display. The best execution for this depends on the motivation for the multiple outputs in the first place.

When running off of the laptop, especially those built prior to 2010, users often want to send output not to the built-in LCD display but instead to an external VGA CRT display. Given the decrease in the display latencies for LCD monitors, this has become less of a concern and there is less need to do so. Further, as VGA connectors become less common, this becomes a less available option. However, there are no adverse timing implications when running E-Prime 2.0 on a laptop while sending the display signal exclusively to a CRT monitor (built in LCD disabled).

Conversely, there are adverse timing implications when using the duplicate, mirroring, or cloning modes to send output to multiple monitors. When duplicating, mirroring, or cloning the display, the equipment will make attempts to synchronize displays, but this may result in settings that you are not aware of. To be safe, if you must run in such a scenario, you need to confirm the settings/performance of each display device with a third-party tester such as the Black Box Toolkit.

Users who are performing MRI and fMRI experiments with E-Prime 2.0 often need to send the same display information to two monitors, one located in the MRI room and the other in the technician's observation room. The best solution in such cases is to utilize Psychology Software Tools projection system Hyperion. With this solution, the output port on Hyperion is used to direct output to the control room display. Therefore, both the MRI and the control room monitors are able to view the same screen, but without using the duplicate/mirror feature.

Caution also needs to be exercised when sending output to a television, including plasma, LCD LED, and OLED-based systems. Television-based systems have more "tricks" to squeeze more performance out of the system, including how they interpret interlaced signals to eliminate any appearance of interlacing. In general, though, you should select the television options that are recommended for gaming mode, since gaming users are often pushing the envelope of display performance.

Sound Latency Solutions

E-Prime provides several tools to address the sound latency problem. 1) The Sound Tester application helps you to identify the optimal configuration for your computing environment. The Sound Tester is described in the KB article: KB 4348 INFO: Use of SoundTester to determine machine compatibility with ASIO or CoreAudio/WASAP API. 2) The CodecConfig utility helps to identify the optimal codecs to use for particular sound files. CodecConfig is described in the KB article: KB 3162 FEATURE: CodecConfig provides ability to choose codecs used for Movie and Sound rendering. 3) PST periodically tests different sound cards and reports our latest results. See our Sound Startup Latency Tests page for details.

4.4.4 Response Collection Solutions

Section 4.2.4 Response Collection Device Challenges described the challenges of obtaining an accurate timestamp for certain switched-based response devices.

Serial Response Box

The Serial Response Box (SRBox) manufactured by Psychology Software Tools is designed for optimal performance with E-Prime. The SRBox transmits the current state of its inputs to E-Prime every 1.25 milliseconds. When E-Prime observes an initial transition on any of the SRBox inputs, it generates a response with a timestamp that corresponds to this initial transition. E-Prime will then ignore any subsequent transitions on that input for a brief period of time known as the debounce period. The debounce period prevents E-Prime from registering switch bounce as additional responses from the participant. The initial transition is timestamped in order to minimize the variance caused by the inconsistent duration of switch bounce. When the debounce period for the input expires, E-Prime will resume observation of the input for further initial transitions.

Using Third-Party response devices

Response boxes can be made in-house or purchased from another vendor. When working with such devices, however, note that the SRBox input commands in script are only to be used with a Psychology Software Tools- supplied response box. When working with a non-Psychology Software Tools response device, you will need to use either Port or Serial commands, depending on how the box is connected to your machine. If a third party vendor claims that their device is compatible with E-Prime 2.0, then contact that vendor for integration information.

4.4.5 Advanced Topic: Synchronization with External Equipment

There are numerous scenarios in which researchers need to communicate between an E-Prime 2.0 experiment and another hardware device. Common examples include having E-Prime 2.0 send a signal to another computer that is collecting biological monitoring data during key points in the trial sequence, such as when the stimulus is presented or when a response is made. In these scenarios, activities such as the starting a trial after receiving a "go" signal from another computer, or sending the onset of a stimulus to the other system from E-Prime 2.0, are common. E-Prime 2.0 provides the ability to communicate with external equipment in several ways, which are described below.

Industrial Partners

Psychology Software Tools has partnered with several vendors who perform eye tracking, head motion tracking, and MRI data collection. See our Partners page on the PST web site for details.

Task Events

E-Prime 2.0 Professional introduces TaskEvents. Task Events are always the preferred communication route with external devices, because they are closely integrated within E-Prime 2.0's runtime internals and will execute more precisely than script-only alternatives. TaskEvents offer a wide variety of options to take action when specific time critical events occur during an experiment. These capabilities, which were difficult to perform in previous editions of E-Prime 2.0, can now be achieved using the E-Studio interface relying less on complex design or advanced E-Basic scripting techniques. For more details on Task Events, please see the following resources:
- E-Prime 2.0 KB 4803 - FEATURE: Task Events
- Task Events described in a YouTube Video

Port Communications

E-Prime 2.0 supports communications via serial, parallel, and socket communications.
Please see the following Knowledge Base Articles:
- KB 1319 - INFO: General Serial Device Information
- KB 1521 - INFO: Serial Communication in E-Prime 2.0
- KB 3023 - INFO: Recommend Parallel Port Adapters for Machines without a Parallel Port
- KB 1318 - INFO: How do I notify external equipment at the exact time an event occurs in E-Prime 2.0?
- KB 896 - FEATURE: Add a socket object or device to support TCP/IP communications

SNTP

Any time you correlate data from two different computers, you will need some way to accommodate clock drift E-Prime 2.0 supports the use of an SNTP clock to address this issue. A full treatment of SNTP (Simple Network Time Protocol) is beyond the scope of this document, but in brief it is an internet standard protocol for synchronizing time across computers. E-Prime 2.0 can be set to use an SNTP clock rather than the default real-time clock. In switching to the SNTP clock, E-Prime 2.0 runs as a client whose clock is constantly adjusting, or slewing, to synchronize with the server clock. The SNTP client looks at time information that is sent from the NTP server, compares this information with its local clock reads, and adjusts the results of the local clock read accordingly. Ideally, E-Prime 2.0 is included on a closed network with no internet access and no other additional computers, although this is not always possible. In our experience, running on a LAN that utilizes switches rather than hubs results in acceptable performance.

4.5 E-Prime Best Practices

In the earlier sections of this chapter, we have explained the challenges that modern personal computers present to researchers who want to obtain millisecond accuracy, along with the solutions that E-Prime 2.0 provides and detailed explanations about how the E-Prime 2.0 solutions work. This material makes for a lot of reading!   This section, E-Prime 2.0 Best Practices, is designed as a final check-list that can be easily referred to as your begin the implementation of your experiment or prepare for data collection.

4.5.1 Operating System Solutions

DO run MSCONFIG to identify the processes that are running on the data collection computer and determine if these processes can be shut down.

DO disconnect from the network prior to collecting data.

DO run experiments available on the [Product Service and Support site](#) to check the timing accuracy of your data collection computer.

DO select the appropriate timing mode (Event or Cumulative) for your design.

DO include a set of practice trials at the start of the experiment, to allow the operating system to obtain stimuli files from disk and place in its cache.

4.5.2 Stimulus Preparation Solutions

DO use GenerateTopOfProcedure where possible to enable media files to be loaded during the non-critical timing inter-trial interval.

DO use PreRelease same as duration except when it would be an issue.

4.5.3 Stimulus Presentation Solutions

Visual Stimuli

DO use the Canvas to draw stimulus first to an off-screen canvas and use Canvas.Copy to transfer it to the screen.

DO use Flipping (DisplayDevice.FlippingEnabled = yes) to avoid display tearing.

DO use PreRelease when possible to preprocess/preload the next item to be displayed. However:

- DO NOT use PreRelease on any object with a Duration = 0 ms
- DO NOT set PreRelease to be greater than the object's duration
- DO Take care when using PreRelease on the last object of a Procedure, or if the next object after the PreRelease is an InLine, PackageCall, or FeedbackDisplay

DO use RefreshAlignment.

DO use less than the full screen when possible. E-Prime 2.0 introduces a new Frame size default value of 75% for the Width and Height settings on Slide, ImageDisplay, TextDisplay, and MovieDisplay objects. Most stimuli do not require the full screen.

DO NOT match the desktop resolution.

DO NOT use ClearAfter to clear the display.

ClearAfter is a deprecated property for TextDisplay, ImageDisplay, and MovieDisplay objects, meaning that earlier versions of E-Prime 2.0 included it but now its use is discouraged. ClearAfter operates by writing a blank line to every line that is available. For example, if the experiment is running in 1024 x 768 resolution, the system would write 768 blank lines to the screen. This can be a (relatively) time consuming process which could introduce a delay in the execution of the next item in the display sequence. Further, ClearAfter may adversely affect the execution of PreRelease, port pulsing, TaskEvents, and OffsetSync for vertical blank, because it adds a delay to the FinishTime of the object which is being cleared.

If you need to remove the contents of the current display and there is another display immediately following, then you can simply use the second display to 'clear' the screen; the contents of the second display item will overwriting the initial display with the new content. This requires that frame of the second display is in the same location as the frame for the first display, and is at least as large as the first display. If you need to clear the screen but don't have a second display object, then you should add a blank TextDisplay object after the display to be cleared. Implementing the ClearAfter as its own object provides for proper scheduling and logging of events for later analysis.

4.5.4 Stimulus Display: Sound and Movies
DO install a comprehensive codec package such as [K-Lite Mega Codec Pack](#) and the latest version of [DivX](#).
DO run Codec Config to confirm the API and codec mappings used to render your sound and movie files.
DO check for dropped frames.

Synchronizing with external equipment
DO ensure that you have the most recent versions of all drivers, including the Serial Response Box, Codec drivers, and E-Prime 2.0 version.

Additional best practices
These items do not fall into any of the categories above, but are recommended practices.
DO run pilot participants through all of the between- and within-participant manipulations:
DO use the E-Run Test to automatically generate responses to all input screens. This enables the largest number of conditions to be tested. For example, if you run through the full set of exemplars on a TrialList, then you confirm that all files that need to be loaded throughout the course of the experiment can be found. Doing the E-Run Test sessions also confirms that all stimuli files are available.
.

4.6 How do I confirm my timing?
4.6.1 TimeAudit

E-Prime 2.0 provides the ability to log timestamp information for many of the actions related to preparation and actual display of the stimulus presentation objects, e.g., SlideImage, MovieDisplay, TextDisplay, most of which have been included in the previous section's diagrams. For example, OnsetTime is the E-Prime 2.0 variable that logs the timestamp of when the object began its critical action, such as drawing on the screen. In general, examining the value of a particular timestamp variable is not particularly useful for confirming the timing of a stimulus event or series of events. However, careful examination of particular combinations of timestamp information can prove to be extremely useful when attempting to confirm the timing accuracy of an E-Prime 2.0 experiment.

The set of timestamp variables is broken into two separate categories, Time Audit and Time Audit (extended). The Time Audit category contains four basic variables, as shown in the table below:

OnsetTime	Timestamp (in milliseconds from the start of the experiment) when the stimulus presentation actually began.
OnsetDelay	Difference in milliseconds between the actual onset time and the expected or "target" onset time. The "target" onset time is calculated internally by E-Prime 2.0 as it prepares the object to be presented/processed.
DurationError	Difference in milliseconds between the actual duration and intended duration before the command ended. If the event is terminated by a response, a value of –99999 will appear in the column (i.e., since the duration error cannot be computed in that situation). Using (same as duration) for PreRelease makes this value obsolete.
OnsetToOnset	Difference in milliseconds between the timestamp of the next object's OnsetTime and this object's OnsetTime

The Time Audit variable for a given object are logged when the Data Logging field on the Duration/Input tab for the object is set to "Time Audit". The list of available variables that can be logged are shown on the Logging tab of the object, as shown in Figure 14. The following values should be examined in E-DataAid:

• **OnsetDelay**: The OnsetDelay for a given object is the difference between the TargetOnsetTime for an object and its actual OnsetTime. In other words, OnsetDelay means the time elapsed between the intended onset time, as set by E-Prime 2.0 while the experiment is executing, and the actual onset time. If PreRelease is set to 0, then the intended onset time is essentially the OffsetTime of the previous object. OnsetDelay values greater than zero occur when the object needs time to load (e.g. images, video) or when it is waiting for the next refresh of the monitor. Ideally, you should see OnsetDelays of 0 or a negative value. (Negative values occur due to refresh alignment). Again, this only applies when PreRelease is not being used.

• **OnsetToOnset**: The OnsetToOnset value is the best metric to examine how long an object was visible. However, exercise caution when using the OnsetToOnset value as this value does not account for position of objects on the screen (i.e. objects displayed in the center of the screen do not appear until approximately ½ of a refresh rate later). This value does also not account for the use of an LCD monitor, which can add display latency. Finally, if the second object in the display does not paint over the first object, then the first object is still on the screen and visible.

The TimeAudit Extended category contains a large number of variables. These will only be logged by manually selecting them; there is no option to enable them all in the Duration/Input tab.
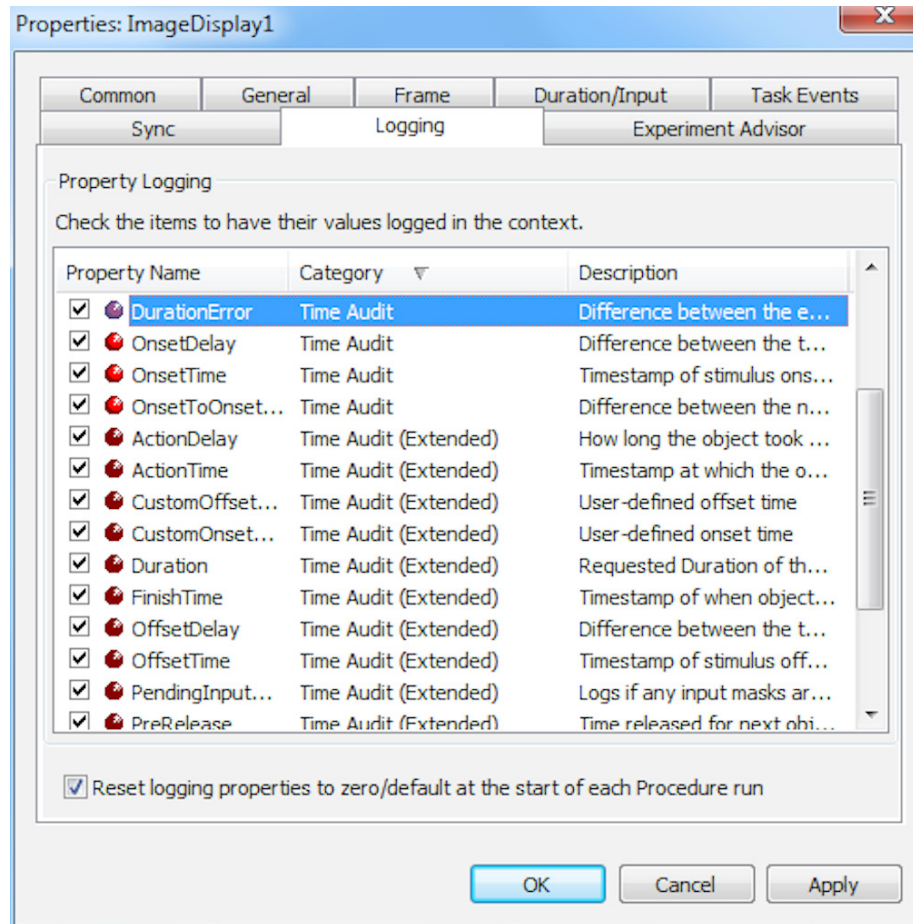
Figure 14. The Logging tab

The following values should be examined in E-DataAid:
• **FinishTime** and **OffsetTime**: There should be a 0 or 1 ms difference between these two values. Further, when PreRelease is set to "(same as duration)", these values will be near the same as the OnsetTime.
• **ActionDelay**: The ActionDelay should be 0 or 1 ms. Any other values may indicate a hardware problem, particularly with the sound card or devices used with Task Events.

4.6.2 Experiment Advisor (E-Prime 2.0 Professional Users only)

TIMING CHALLENGE: Confirming timing results with Experiment Advisor (E-Prime 2.0 Professional Version only)		
The Experiment Advisor provides E-Prime 2.0 Professional users the ability to detect design and timing errors in E-Studio and E-Run. The Experiment Advisor modules are enabled or disabled through the Experiment Object, on the Experiment Advisor tab. Experiment Advisor reports are automatically generated for any enabled modules when the experiment is compiled or run. For further information, see KB 4806 FEATURE: Experiment Advisor. Experiment Advisor findings are reported in the Output window in the following ways:		
E-Prime 2.0 Tools	**User Actions**	**For Further Information**
Display adapter set to clone mode	The display adapter of the machine is set to clone or mirror mode. Mirroring can cause timing issues since vertical blank synchronization is not accurate between multiple displays, and determining which display is considered primary can cause confusion and may not be consistent between computer configurations.	KB 5014

	TIMING CHALLENGE: Confirming timing results with Experiment Advisor continued (E-Prime 2.0 Professional Version only)	
E-Prime 2.0 Tools	**User Actions**	**For Further Information**
DisplayDevice uses Match Desktop	A DisplayDevice has the "Match desktop resolution at runtime" property set to Yes which can lead to timing and performance issues.	KB 4989
Experiment Advisor suggests advice on objects not divisible by refresh.	When duration values are not divisible by the refresh duration, large OnsetDelay values can occur. This Experiment Advisor Module will report a finding any time an object has a Duration value not divisible by the refresh rate and suggest an alternative.	KB 3795
Evaluate SoundDevice. API	A SoundDevice is using the DirectSound API on Windows Vista or later which typically cannot achieve latency values under 30 ms. CoreAudio or ASIO APIs are recommended for Windows Vista or later.	KB 4979
Large OnsetDelay	An object is collecting OnsetDelay Stats and has an OnsetDelay that is larger than a single refresh. Setting the Duration property to a value divisible by the refresh rate and the PreRelease property to (same as duration) will typically reduce OnsetDelay values.	KB 4978
Non-Visual object using OnsetSync or OffsetSync	A non-visual object (SoundOut, SoundIn, Wait) has either of its OnsetSync or OffsetSync properties set to (vertical blank) which is usually unnecessary unless syncing with external equipment.	KB 4975
Experiment Advisor detects when inappropriate mixture of Cumulative and Event TimingMode in use	Using an inappropriate mixture of cumulative and event timing mode can lead to unexpected results.	KB 4504
Use of ClearAfter	An object has its ClearAfter property set to Yes. ClearAfter is a legacy property that should be set to No. Consider using a blank TextDisplay object to clear the screen.	KB 4990
Use of Stretch	A visual object has its Stretch property set to Yes, which can cause display timing anomalies. Instead of using Stretch, consider editing the source material to match the size and proportions you want to display during the experiment.	KB 5057
WaitForVerticalBlank with Flipping	The DisplayDevice.WaitForVerticalBlank E-Basic command is used when the DisplayDevice.FlippingEnabled property is set to Yes/True. Calling WaitForVerticalBlank while flipping is enabled can cause timing concerns.	KB 4993
Volatile Clock on Windows OS	The E-Prime 2.0 Primary Clock is configured in a way that could cause timing inconsistencies on Windows OS. Contact PST Tech Support to configure the clock for more stabilized performance when using Windows OS with this finding.	KB 4974

Chapter 5: Using E-Basic

5.1 Stage 1: Why Use E-Basic?

E-Studio is a robust application, with E-Basic as the underlying scripting language. Specifically, the graphical design created in E-Studio is translated to E-Basic script when the experiment is generated.

By utilizing the various objects in E-Studio, users can implement a wide range of experiment designs. However, E-Studio's objects cannot support all experiment designs. That is where E-Basic is useful. If E-Studio does not provide an object that supports the needs of a particular experiment, then E-Basic affords the ability to accommodate individual user's unique needs.
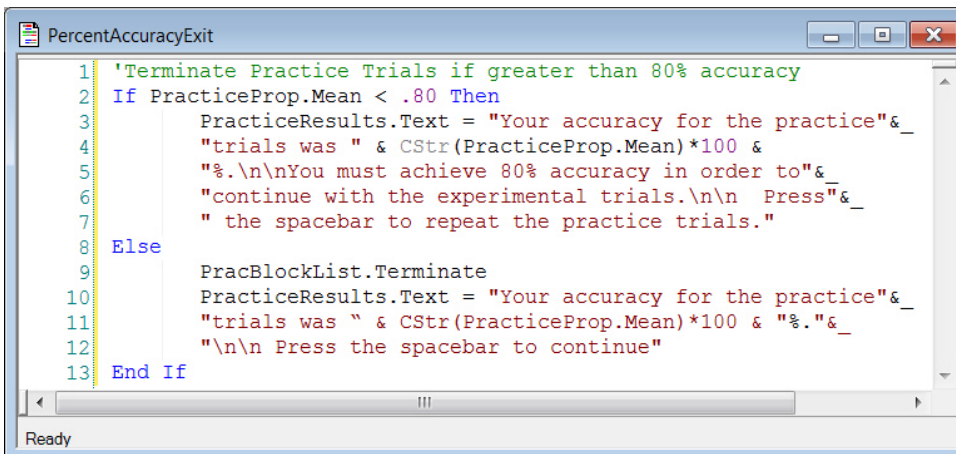
E-Studio provides the foundation for experiments in E-Prime 2.0. It is recommended that all users, regardless of their programming expertise, take full advantage of the graphical design interface in E-Studio rather than writing complex E-Basic scripts. E-Studio's graphical interface can do most, if not all, of the work. Many users will not even need to use E-Basic.

The majority of all experiments can be done using E-Studio's graphical design interface. The basic structure of almost all experiments includes blocks of trials within a session. It is more efficient to use the E-Studio design interface to set this up by dragging and dropping icons onto procedural timelines than it is to write the equivalent script.

E-Basic is the underlying scripting language of E-Prime 2.0, and the power of the E-Prime 2.0 system. Where E-Studio leaves off, E-Basic may be used to extend the system. This is similar to the use of Visual Basic for Applications within the Microsoft Office suite of applications. When E-Basic is used in conjunction with E-Studio and E-Run, power and flexibility abound.

E-Basic really excels when used to extend the functionality of E-Studio. For example, the ability to exit a set of practice trials based on a percentage correct criterion is something that not all users are interested in, and for which there is no graphical option. However, this function can be accomplished through a few lines of E-Basic script strategically placed in an InLine object or two. The following example illustrates most of the script necessary to set up a percent accuracy exit:

Upon reading this script carefully, it is not difficult to guess what it is supposed to do[1]. One of the nicest features of E-Basic is that the language is similar to ordinary English. If the example looks like a foreign language, rest assured, this chapter has a section devoted to the beginner user (along with sections for intermediate and advanced users). The goal of this chapter is to get the user accustomed to writing E-Basic script. For users with little or no programming experience, the following section also recommends some additional sources.

```
PercentAccuracyExit
1  'Terminate Practice Trials if greater than 80% accuracy
2  If PracticeProp.Mean < .80 Then
3          PracticeResults.Text = "Your accuracy for the practice"&_
4          "trials was " & CStr(PracticeProp.Mean)*100 &
5          "%.\n\nYou must achieve 80% accuracy in order to"&_
6          "continue with the experimental trials.\n\n  Press"&_
7          " the spacebar to repeat the practice trials."
8  Else
9          PracBlockList.Terminate
10         PracticeResults.Text = "Your accuracy for the practice"&_
11         "trials was " & CStr(PracticeProp.Mean)*100 & "%."&_
12         "\n\n Press the spacebar to continue"
13 End If
```
Ready

---

[1] This example is intended to illustrate the readability of E-Basic script only, and is not a complete example. The script shown in this example must be used in conjunction with an E-Studio file that includes other objects, such as a TextDisplay object named PracticeResults, For a complete .es2 file that illustrates criterion-based exit, please see the sample experiment CriterionBasedExit.es2 which is available from the Support web site

Stage 1, Step 1: Before Beginning

Before attempting to write any script using E-Basic, a few critical pieces of information must be known. Even the most expert programmer should read the following section.

Learning to write script will minimally require learning the basics of programming. The complexity of the experiment task will determine the amount and complexity of the script required, and therefore, the amount and complexity of script-writing knowledge necessary. Most experiments involving user-written script will minimally require the user to be able to write and add functions (e.g., to display an internal variable, or indicate contingent branching based on responses). For someone with programming experience, these types of functions might be accomplished in a few minutes or hours, while someone without programming experience may need a day or more to accomplish the same goal. More complicated tasks, such as creating a simulated ATM, will require more comprehensive programming and may require considerable programming experience. Again, the length of time required to accomplish the task will depend on programming knowledge, skill, and the task itself. Complex systems interactions, such as creating new SDKs, would require significant programming knowledge and expertise, and are best handled by professional programmers.

The present chapter discusses the E-Basic language and the basics of entering user-written script. If the reader has experience programming with languages such as Basic, C, or Pascal,   the information in this chapter should be straightforward. Users who are new to programming are advised to become familiar with script by taking a programming course, or by carefully working through script examples. E-Basic is very similar to Visual Basic for Applications. Visual Basic for Applications would be the best programming course to choose, but Visual Basic would also be useful.

For more help, PST recommends the following reference sources:

For the novice user with little to no programming experience and for those new to VBA type languages: *VBA for Dummies*, 5th Edition, John Paul Mueller, John Wiley & Sons, Inc., Hoboken, NJ, 2007.

For more advanced users with substantial programming experience: *VBA Developer's Handbook*, 2nd Edition, Ken Getz & Mike Gilbert, Sybex Inc., San Francisco, CA, 2001.

Another efficient method of learning to program using E-Basic is to examine script examples in the E-Basic Help. The E-Basic Help may be launched from the Start button, or through the Help menu in E-Studio.

Stage 1, Step 2: Introduction to Programming

Regardless of programming experience, every user should follow the same basic steps when writing script.
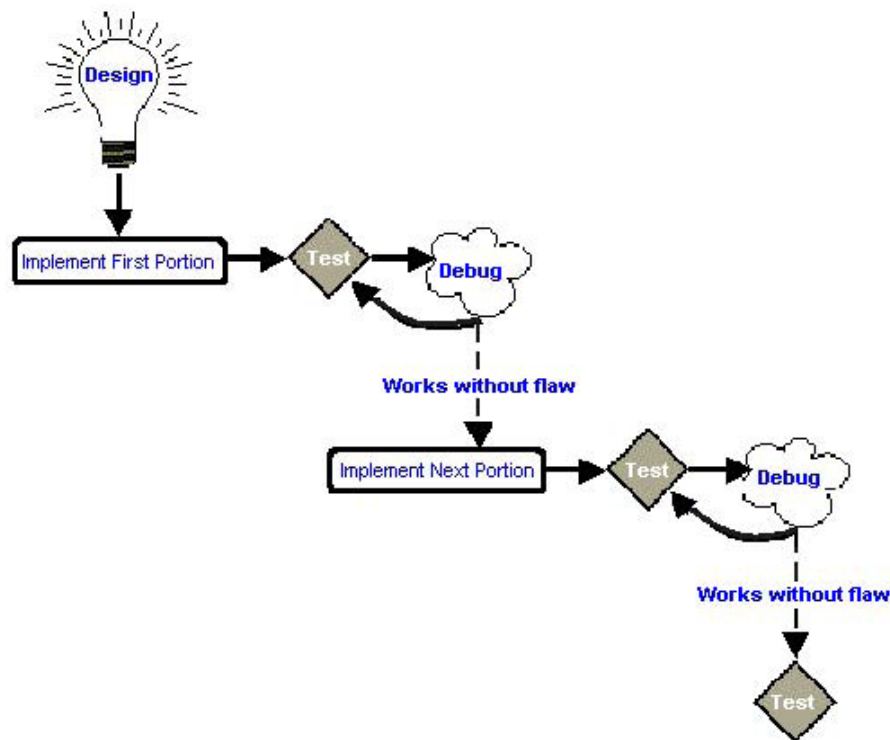
Design

It is critical to thoughtfully design the components of the experiment. Often the simple act of creating a flow chart to represent the experiment can help to remain focused on the task, and not be overwhelmed with the experiment as a whole. It is extremely helpful to think of an experiment in logical pieces rather than the sum of its parts. This is true of experiments in general, but more so of script.

The First Segment

Implement First Segment

Once the task is broken into segments, begin to implement just the first portion or segment. Often users will try to do too much and then lose sight of the relatively simple steps required to reach the goal. Start simple. For instance, if the goal is to display a question on the screen and then collect and score the response, the flowchart might contain the following steps:

Step 1: Display a word on the screen
Step 2: Display a question on the screen
Step 3: Collect a response
Step 4: Score the response

It is helpful to get through step one before moving on to step two and so on. Start small and build on a solid foundation.

Test the First Segment

Once the first step is implemented, it MUST be tested thoroughly. If a problem is discovered, it is important to fix it now rather than later. Testing is particularly important in research. Precision testing is described in *Chapter 3: Implementing your Experiment in E-Studio*.

Debug the First Segment
Search through the code to find the problem and then correct it. There are many tricks to make debugging less painful than taking a stab in the dark. Debugging tips are included later in this chapter, in *5.8 Stage 8: Debugging in E-Prime 2.0 (Page 155)*.

The Next Segment
Implement the Next Segment
After the first segment of code is thoroughly tested and you are confident that it is working as expected, begin to implement the next section of the task.

Test the Next Segment
Repeat the testing procedure. It is important to be confident in the code.

Debug the Next Segment
Repeat the debugging process if necessary.

Keep Testing
Repeat the testing procedure until you are certain that the code is working as designed. When there is confidence in the functionality of the code, be certain to run a few pilot participants. This is critical, because it is often the case that the participant will uncover a flaw. Rather than losing an actual participant, it is best to find the problem with a pilot participant.

Basic Steps
Included in this chapter is a section presenting the Basic Steps for Writing E-Prime 2.0 Script (see *5.4 Stage 4: Basic Steps for Writing E-Prime 2.0 Script (Page 127)*). Even if there is familiarity with writing script, and especially if not, the basic steps offer helpful information for scripting within E-Prime 2.0. Be sure to review the Basic Steps for Writing E-Prime 2.0 Script section as an effective tool in learning to use E-Prime 2.0.

5.2 Stage 2: Introducing E-Basic

E-Basic is a standard object-oriented programming language with over 800 commands, which has been customized to better fit the needs of real-time research. Some of the E-Basic commands were specially developed by PST to accommodate the unique requirements of behavioral research. Unlike BASIC or Pascal which operate by executing a list of tasks, E-Basic contains distinct sets of objects that are able to perform independent functions and interact with each other to perform assigned tasks.

E-Basic is user-friendly, unlike other more advanced languages (e.g., C++), and is nearly identical to Visual Basic for Applications. Essentially, the only part of VBA that will not transfer to E-Basic is the forms used in VBA.

All edits to and experiment should be made via the E-Studio graphical interface using an .es2 file. In E-Studio you will be working with E-Basic script via E-Object from the toolbox. E-Basic script can be used in these objects once they are placed on procedural timelines within the experiment> E-Basic script can also be entered in the User tab from within the Script window, or via an InLine object.

E-Basic script (.ebs2 file) generated as a result of the graphical components used to build the experiment may be used as an informative tool; reading through the .ebs2 file is a good way to become familiar with E-Basic and its components. The script required by an experiment (.ebs2 file) is written into an E-Basic script file simply by pressing the Generate button. Pressing the generate button also overwrites any existing .ebs2 file.

User-written E-Basic script is generally placed in one of three places in an experiment:

1) In an InLine object to be executed at a given time during a Procedure.
    This is the most common placement of script. The script included in an InLine object is inserted "as-is" into the .ebs2 file. The location is determined by the placement of the InLine object in the structure of the experiment.

2) On the User tab in the Script window to declare global variables, functions sub-routines and user-defined types.
    Refer to *Stage 3, Step 4: Variable Declaration and Initialization (Page 125).*

3) In an InLine placed at the beginning of the experiment to initialize variables.
    For example, global variables declared on the User tab must be initialized prior to their use, and it is common to do so at the beginning of the experiment.

For details about the InLine object as well as the User tab in the Script window, refer to *Stage 3, Step 4: Variable Declaration and Initialization (Page 125)* and *5.4 Stage 4: Basic Steps for Writing E-Prime 2.0 Script (Page 127).*

Stage 2, Step 1: Syntax

The E-Basic language is object-oriented. Each object has a list of associated properties and methods. An object appears in code using object.property or object.method, where "object" equates to the object's name and the item after the dot (.) refers to either the object's particular property or method.

Objects
Objects are the core components of E-Basic. An object in E-Basic is an encapsulation of data and routines into a single unit. The use of objects in E-Basic has the effect of grouping together a set of functions and data items that apply only to a specific object type.

In E-Studio, there are a variety of E-Objects including TextDisplay, Slide, List and so on. The graphical representations of those objects are accessible in E-Basic using the object.property syntax. In the example below, Instructions is a TextDisplay object. The statement follows the object.property syntax to set the Text property for the Instructions object.

```
Instructions.Text = "Welcome to the experiment"
```

Object.Properties
Objects expose data items, called properties, for programmability. Usually, properties can be both retrieved (Get) and modified (Set). Just as each E-Object in E-Studio has a set of associated properties, so do objects in script. The property of the object is referenced using the object.property syntax.

Essentially, properties store information regarding the behavior or physical appearance of the object. For instance, a TextDisplay object has a Text property, which represents the text to be displayed on the screen.

Object.Methods
Objects also expose internal routines for programmability called methods. In E-Basic, an object method can take the form of a command or a function.

Object.Commands

Commands are quite literally the action or actions that the object can perform. An object command is referenced using the object.command syntax. Commands may or may not take parameters. For example, the Clear command (i.e., object.Clear), which may be used to clear the active display, requires no parameters, while the Rectangle command associated with the Canvas object requires parameters to define the position and size of the rectangle to be drawn. Note, not all commands are available to all objects. Refer to the E-Basic Help for a listing of commands available to each object.

Object.Functions

An object method which returns a value is called a function. An object function is invoked using the object.function syntax. Functions may or may not take parameters. For example, the Mean function (i.e., object.Mean) requires no parameters to return the mean of a collection of values contained within a Summation object. However, the GetPixel function associated with the Canvas object requires parameters in order to return the color value at a particular x, y coordinate position.

Stage 2, Step 2: Getting Help

The E-Basic Help is launched through the Start button (select Start, E-Prime 2.0, Help, E-Basic Help), or through the Help menu in E-Studio. The E-Basic Help command opens the Help Topics dialog, containing Contents, Index and Find tabs.

Contents

The Contents tab lists a table of contents for the Help topics available within the E-Basic Help system. The main Help topics are displayed as books, which may be opened to display related subtopics. The topics within the Contents list may be expanded or collapsed by double clicking the book. When a topic is expanded, information concerning individual subtopics may be displayed by double-clicking the desired item.

Index

The Index tab displays an alphabetical listing of topics and commands within E-Basic. This is typically the best way to find a command or function name. The Help information for a particular topic may be displayed by first selecting and then double-clicking the topic. The listing of topics in the index may be searched by either typing directly in the keyword field, or by scrolling through the topics contained in the index using the scroll bar on the right side of the Index listing.
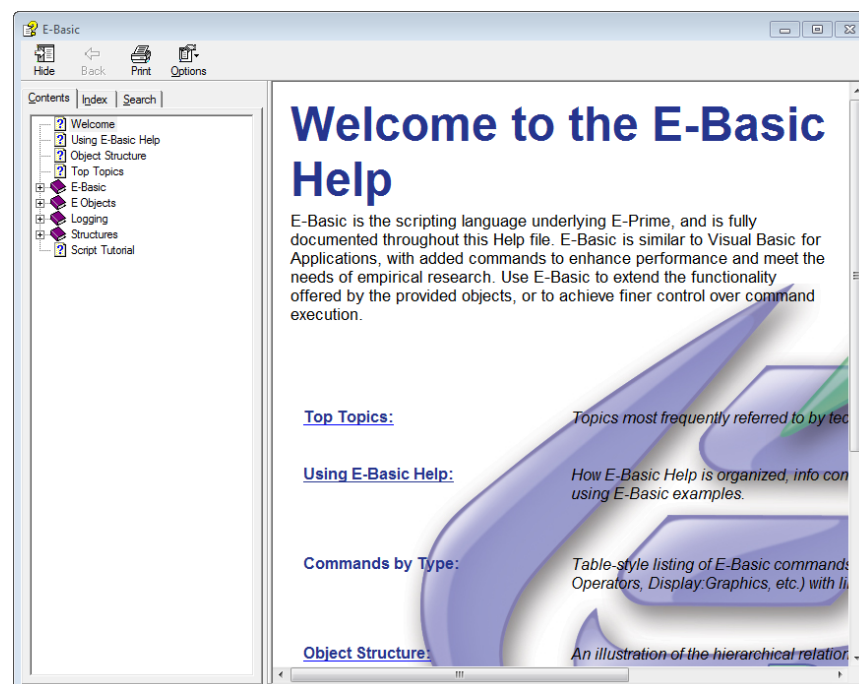
Search

The Search tab allows searching for specific words or phrases within the Help topics. After typing in a word to search for in the Help topics and hitting the Enter key or clicking on the List Topics button, the topics in which the word appears are listed, as well as additional suggestions to narrow the search. The Help information for a particular topic may be displayed by double clicking the topic.

The Search page is useful for locating all commands that reference a particular topic (e.g., all topics referencing the "Abs" string). However, since any reference to the string searched for is included, this method often gives a long list of topics that must be searched through in order to locate relevant information.

Reading Help Topics

E-Basic Help topics are presented using a standard format for all topics. Each description within the E-Basic Help topics describes the use of the statement, function, command or object, the syntax and parameters required, any considerations specific to the topic, an example, and a listing of direct links to related topics.
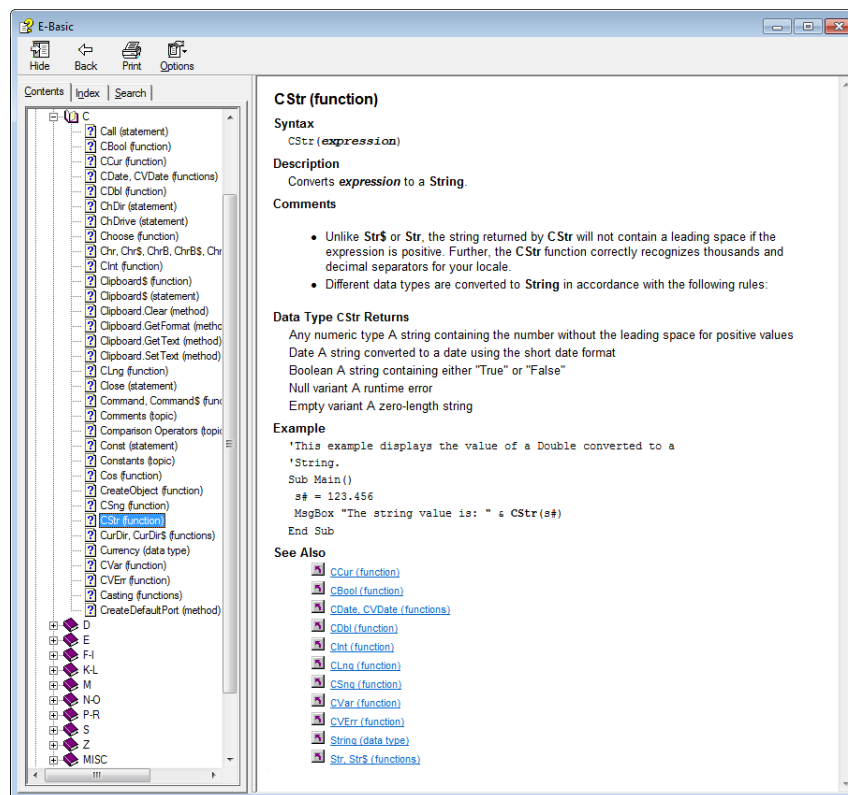
Section	Purpose
Syntax	Describes the parameters necessary for use with the statement, function, command or object.
Description	Describes the purpose of the statement, function, command or object.
Comments	Lists specific considerations for the statement, function, command or object.
Example	Actual script example illustrating the use of the statement, function, command or object.
See Also	Direct links to related statements, topics, functions, commands or objects.

5.3 Stage 3: Communicating with E-Prime 2.0 Objects

Before attempting to write E-Basic script, it is useful to understand how information is managed within E-Basic and E-Prime 2.0. Within E-Basic, data and routines acting on that data may be encapsulated into units called "objects". For example, at the trial level, the List object is an encapsulation of the trial level data (e.g., stimuli, independent variables, etc.) and the routines applied to that data (e.g., TrialProc).

In the image below, each line in the TrialList object lists a specific exemplar. The Stimulus and CorrectAnswer attributes contain the trial level data necessary to present the stimulus and score the input collected from the participant. The Procedure attribute indicates which routine is to be associated with the specific piece of data. In this case, the TrialProc Procedure, which calls individual objects to present a fixation, stimulus, and feedback, is applied to the specific stimulus (i.e., X or Y).

Thus, when the experiment is executed, a specific exemplar is selected from the TrialList object. An exemplar from a List is an entire row (or level). In the current example, a chosen exemplar includes the data to present the stimulus (Stimulus = either X or Y) and score the response (CorrectAnswer = 1 or 2), as well as the Procedure using that data (TrialProc).

The TrialList object calls the TrialProc object (containing the Fixation, Stimulus and Feedback objects) using the Stimulus and CorrectAnswer information from the chosen exemplar. In this way, the trial level information encapsulated in the TrialList object is used to run a series of trials. Likewise, the block level data and routines are encapsulated in the BlockList and BlockProc objects.

Stage 3, Step 1: Context

As with the block and trial levels, the data and routines involved in the overall experiment are encapsulated in object form at a higher level. The experiment data and associated routines are combined to define the Context object. Within the Context object, information is hierarchical. This hierarchical nature allows values from upper levels to be either inherited or reset at lower levels, while search procedures to determine the value of attributes occur at the current level and continue in an upward direction until the information is located.
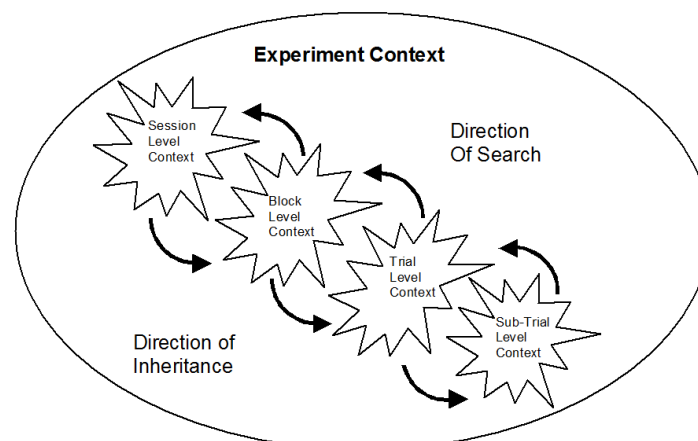
For example, if an attribute is defined at the block level, this attribute will be available at the block level and all levels subordinate to the block level (e.g., trial, sub-trial, etc.). When values are requested, the current level is searched first, and the search continues upward through the levels until a value is found or until all levels have been exhausted. This defines a hierarchy of context levels within the overall Context. If a value for an attribute is requested during a trial, E-Run will search the trial level context first, then the next level of context (e.g., block level), and so on, to resolve the value for the attribute.

For example, perhaps a series of blocks of trials is to be run, with the instructions for a series of trials changing only when the next block is run. An attribute named "Instructions" would most likely be created at the block level, since the value of the Instructions attribute would only need to vary from block to block. However, perhaps the instructions are to be displayed at the beginning of each trial as a reminder of the task. A display during the trial Procedure would necessarily reference the Instructions attribute (i.e., [Instructions]), which is defined at the block level. E-Run would first check the trial level context, then the block level context, and then the session level context in order to resolve the value of Instructions. In this example, the search would terminate at the block level, where the Instructions attribute is defined.

Attributes

Data within the Context object is manipulated through the creation and setting of attributes. Attributes differ from variables in that they generally define the experimental conditions and are logged by default. The logging may be disabled for individual attributes.

Attributes must be entered into the Context of the experiment before they may be referenced or modified. Although the Context object outlines the overall experiment, the Context is hierarchically structured. Attributes defined at a specific level of the Context may be seen by any level lower than the one at which the attribute is defined, and upper level information may be inherited by lower level attributes. However, attributes defined at lower levels of the Context cannot be "seen" beyond the level at which they are defined. Thus, in order to reference or modify an attribute, it must be placed in the Context and set at the appropriate level. When attributes are created via the graphic objects in E-Studio, specifically with the ListObject, E-Studio automatically places the attributes in the Context. When the user creates attributes manually with script, the user must also manually first place the attribute into the Context. This process is described in *5.4 Stage 4: Basic Steps for Writing E-Prime 2.0 Script (Page 127)*.

Stage 3, Step 2: Object.Properties

Properties are data items associated with objects that may be both retrieved and modified. For example, the Stimulus object shown above has properties that may be accessed and modified through E-Basic script. Properties of an object are accessed using the dot operator, which separates the property from the object with which it is associated. The example below illustrates the assignment of the Text property associated with the Stimulus object. In this example, the Text property is assigned a value of "X."

```
Stimulus.Text = "X"
```

Not all properties may be modified. Read-only properties may not be modified through E-Basic script. A listing of all properties associated with particular objects is available in the E-Basic Help.

Stage 3, Step 3: Object.Methods

Objects also have associated methods, which cause objects to perform certain actions. Like properties, methods are accessed using the dot operator in conjunction with the object. Methods may be broken down into Commands and Functions.

Commands
Commands are used to instruct the program to perform an operation. For example, the Run command is used to launch the object named "Stimulus" within the experiment script.

> Stimulus.Run

Commands may or may not take parameters. A listing of all commands associated with particular objects is available in the E-Basic Help.

Functions
Like commands, functions are used to instruct the program to perform an operation. In contrast to commands, however, functions are used to perform an operation that returns a value. Functions may also be used within a script statement. For example, a function could be written to calculate the mean value for a set of values. Once defined as a function, a single line of script would be necessary to run the function to calculate a mean.

Stage 3, Step 4: Variable Declaration and Initialization

Declaring Variables
Variables are distinguished from attributes in that they are not specifically related to the Context object. Variables are defined and accessible within a particular scope. That is, variables are temporary, and are discarded after the scope (e.g., Procedure) in which they are defined is exited. Note, variables declared in the User tab of the Script window are defined globally, so their scope spans the entire program.

A Dim statement within a Procedure, subroutine or function declares variables locally to that Procedure, subroutine or function. Variables declared within a particular scope are not "seen" outside of that scope, and once that scope is exited, the variable is discarded. For example, a variable declared at the trial level with the Dim command is discarded after the trial Procedure is completed, and is not automatically logged in the data file. A variable must be set as an attribute of the Context object in order to be logged in the data file.

Variables declared at the top level of the experiment (i.e., global variables) must be declared on the User tab in the Script window using the Dim statement. Global variables may then be initialized within the structure of the experiment using an InLine object. Most commonly, the initialization of global variables would be entered in an InLine object at the beginning of the experiment structure. Variables declared globally may be accessed at any point in the structure of the experiment (i.e., their scope includes the entire experiment).

Naming Variables
Variable names must start with a letter, and may contain letters, digits and the underscore character[2]. Punctuation is not allowed, although the exclamation point (!) may appear in a position other than the first or last character[3]. Variable names may not exceed 80 characters in length, and cannot be from among the list of reserved words (see the Keywords topic in E-Basic Help for a listing of reserved words in the E-Basic language).

---

[2] E-Objects named in E-Studio do not permit the use of the underscore character.

[3] If the exclamation point is entered as the last character, it is interpreted as a type-declaration character by E-Basic.

Rules
• Must begin with a letter
• Numbers are acceptable but may not appear in the first character position
• Punctuation is not permitted, with the exception of the underscore one character and the exclamation point (see above paragraph for details)
• Illegal characters are @#$%^&*(){}-+[]=><~`:;
• Spaces are not permitted
• Maximum number of characters is 80
• Cannot duplicate a reserved word
• Cannot use the same name more than once within the same scope
• The backslash character ("\") may not be used. This is an escape character within E-Basic, which signals E-Basic to interpret the character following the backslash as a command (e.g., "\n" would be interpreted by E-Basic as "new line").

As long as the rules above are followed, variables may be named almost anything. However, it is highly recommended that the user follow a logical naming scheme, because this makes programming much easier in the long run. Specifically, give variables logical names to quickly remember their purpose. If the variable is nothing more than a counter, give it a single letter name such as "i," "j" and so on. If the variable's purpose is to keep track of a participant's date of birth, name the variable something like "participant_birthdate" rather than "xvariable."

Stage 3, Step 5: User Script Window vs. InLine Object

The User Script window is accessible via the Script command in the View menu. Notice the two tabs at the bottom of the Script window, User and Full. The Full tab is useful for viewing the code generated by E-Studio. It is not possible to edit script on the Full tab. The User tab allows entering the user's own high-level script.

The User Script tab in the Script window may be used to declare global variables. When variables are declared on an InLine object, they are only accessible throughout the scope of the Procedure in which they are declared. For example, if a variable is declared at the trial level, the variable can only be seen, or referenced, within the scope of the trial. In order for variables to be accessible at any level of the experiment (e.g., at both the block and trial level), the variables must be declared within the scope of the entire experiment rather than within a specific Procedure. Script on the User Script tab is entered into the .ebs2 file globally.

The User Script tab in the Script window may also be used to declare functions and subroutines. In fact, the User Script tab is the only place where functions and subroutines can be declared, due to a syntax constraint[4]. The User Script tab is also the only place where user-defined data types may be declared; the declaration and use of user-defined data types is described in *Stage 7, Step 2: User-Defined Data Types (Page 153)*.

InLine objects are used to insert segments of user-written script within an experiment. Script contained within InLine objects is inserted as a unit into the .ebs2 file. The point of insertion is in relation to the location of the InLine object within the structure of the experiment. For example, if an InLine object is called by the trial Procedure, the script contained within the InLine will be inserted in the .ebs2 file at the point at which the trial Procedure is run. As was described above, global variables cannot be defined InLine; rather they should be declared in the Script window on the User tab.

---

[4] Each subroutine must be declared within a Sub…EndSub wrapper. E-Prime 2.0 automatically includes this wrapper for each object that is placed in the Experiment. Therefore, any InLine object that you create for your own script will automatically have the Sub_EndSub wrapper, You cannot add your own Sub…EndSub wrapper to the InLine object, and you cannot declare functions and subroutines without this wrapper. Therefore, you cannot create functions and subroutines on an InLine object.

5.4 Stage 4: Basic Steps for Writing E-Prime 2.0 Script

The following steps, to be covered in this section , introduce helpful information for writing E-Basic script or accessing values determined through script.

1) Determine the purpose and placement of the script
2) Create an InLine object and enter the script
3) Determine the scope of variables and attributes
4) Set or reference values in script
5) Reference script results from other objects
6) Debug
7) Test

Stage 4, Step 1: Determine the purpose and placement of the script

This step requires the user to consider what task the script is supposed to accomplish and when the action must occur. For example, an experimenter might want to choose a random number from 1-499 to be used as the stimulus display for each trial. The purpose of the script, therefore, is to select a random number and to place the selected value into a form usable by E-Prime 2.0. The placement of the script must be within the Procedure driving the events of the trial. Specifically, since the random number is to be used during the display of the stimulus, the script determining the value must be entered in the trial Procedure prior to the object displaying the stimulus.

Stage 4, Step 2: Create an InLine object and enter the script

Once the appropriate placement of the script has been determined, create an InLine object at that location. Continuing the example from Step 1, if the script is to determine a random number during the trial Procedure (TrialProc) prior to the stimulus display, the most appropriate location for the InLine object containing this script is as the first event in the TrialProc.
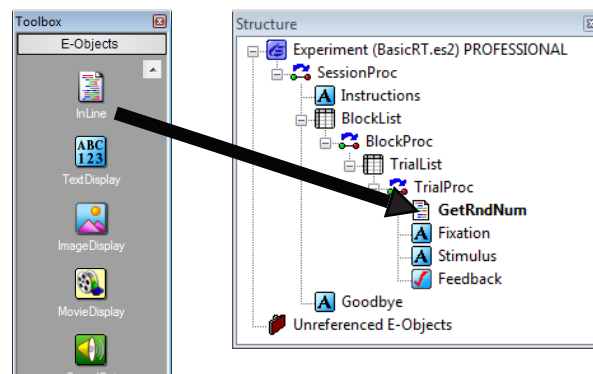
Actually, any time prior to the event displaying the stimulus would be appropriate, but it is good practice to separate the setup events from the critical events of the trial (e.g., Fixation-Stimulus-Feedback). It is also good programming practice to apply useful names to the user-created InLine objects. In this example, the default name of InLine1 has been changed to the more meaningful GetRndNum.

After placing an InLine object in the appropriate location, double click the InLine object to open it, and enter the script required to accomplish the task. To select a random number (1-499), and to place the selected value into a form usable by E-Prime 2.0, enter the following script.
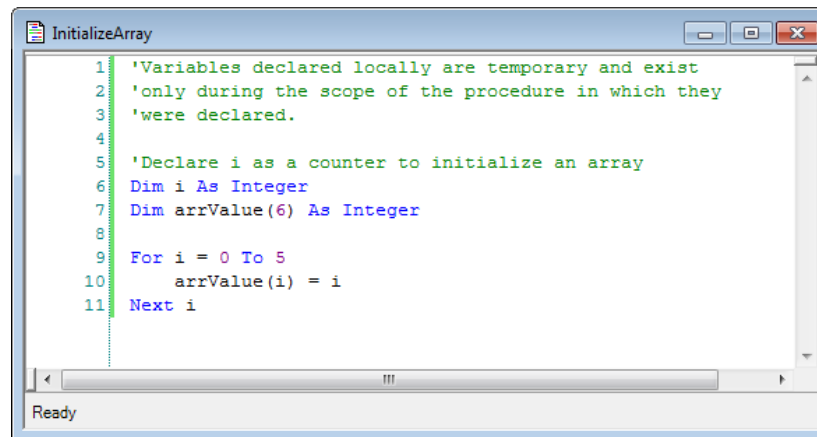
127

The Random function is used to select a random value from 1-499, and the SetAttrib command is used to place that value into an attribute for later access. Notice the single quote character used to enter a comment in the script above. The single quote character is used to skip all characters between the apostrophe and the end of the current line. An alternative method to create comments is with the Rem statement (for "remark"). Refer to the Comments topic on the Contents tab (listed under E-Basic) for more information.

Stage 4, Step 3: Determine the scope of variables and attributes

Variables or attributes declared within an E-Prime 2.0 experiment are limited to the scope of the Procedure in which they are defined. Variables to be used only during a single trial may be declared and initialized at the trial level. For example, a variable used as a counter may be declared locally, as in the following example:
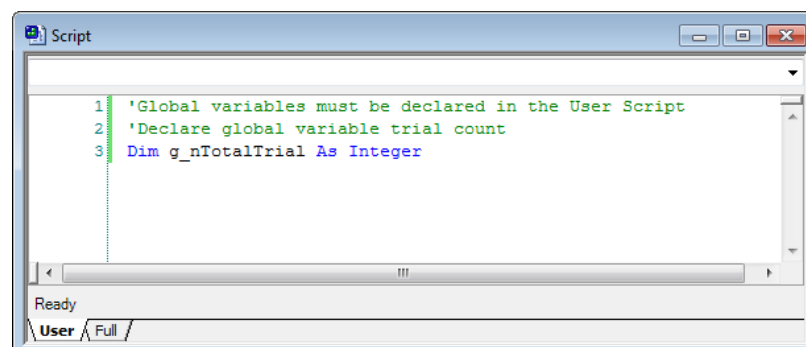
When InitializeArray is placed in the trial Procedure, both "i" and "arrValue" will be accessible during the trial. At the end of the trial Procedure, the "i" and "arrValue" variables will be discarded (e.g., the value of arrValue will not be accessible by an event at the block level).

If a variable is to be maintained or accessed across multiple executions of a Procedure (e.g., performance over a series of blocks), the variable must be declared globally on the User tab in the Script window. For example, a variable might be declared to keep track of the total number of trials. Use the View menu to display the Script window in E-Studio, and select the User tab to declare a global variable.

Initialization of global variables cannot occur on the User tab; instead, an InLine object would be used for this purpose. To initialize a global variable prior to its use (e.g., to initialize the number of trials to 0), use an InLine object placed appropriately. It is a good practice to initialize variables as the first event in the Procedure in which they will be used as part of the Procedure setup events. In this case, global variables exist throughout the scope of the experiment, so initialization should take place as the first event in the Session Procedure.
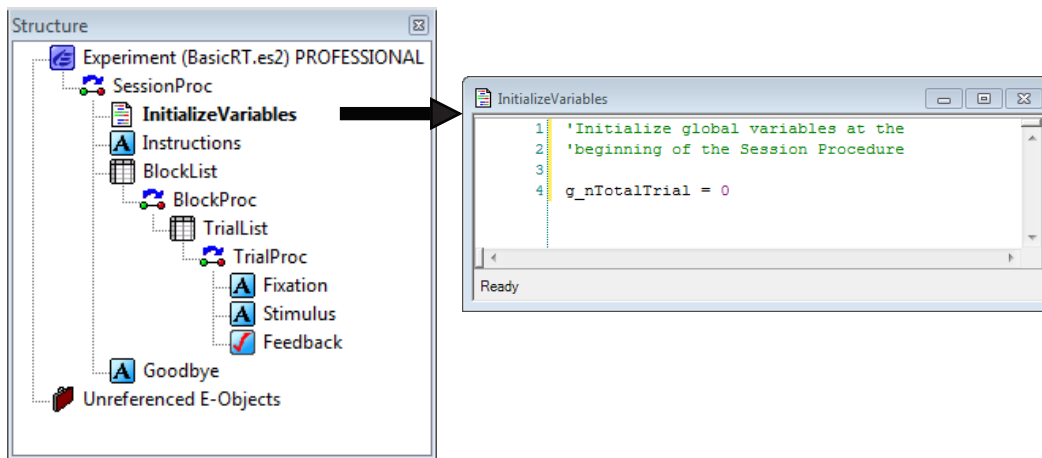
The global variable may then be updated during the trial level Procedure to maintain a count across multiple executions of the Procedure. To use the global variable to update the trial count, insert an InLine object as the first event in the trial Procedure, and enter script to increase the count.
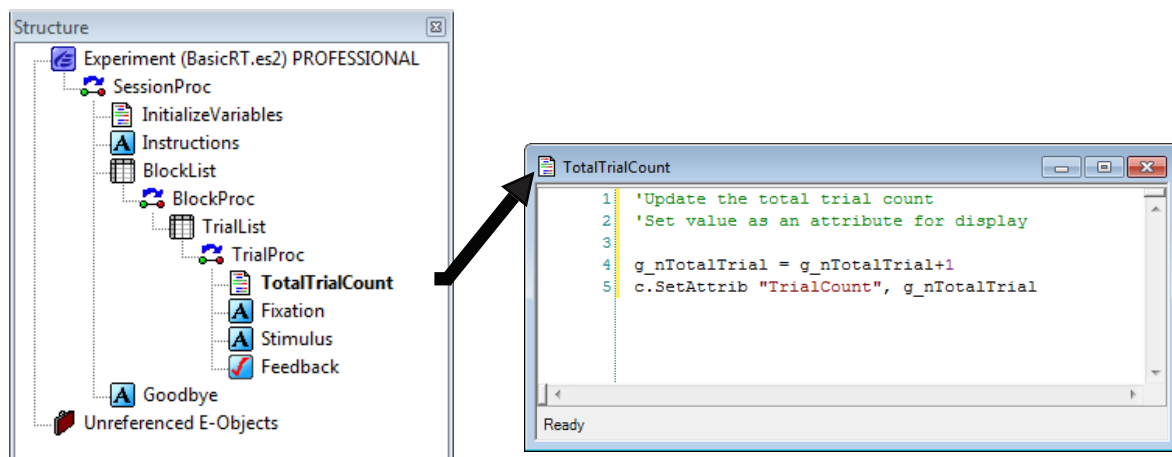
The total trial count variable is updated simply by increasing the previous value by one (i.e., g_nTotalTrial+1). The second line of script referring to the c.SetAttrib command is used to place the value of g_nTotalTrial into an attribute (to make it accessible by other objects with the bracket notation and to log the value in the data file. Refer to the next step for information pertaining to setting values as attributes.

Stage 4, Step 4: Set or reference values in script

If variables are to be retrieved and/or logged, they must be placed into the experimental context as attributes. Otherwise, they exist only temporarily, and are discarded at the conclusion of the Procedure in which they are defined. Globally defined variables, created on the User tab, are discarded when the Experiment Object terminates.

Attributes defined in a List object are automatically placed into the experimental context. For example, if a List object defines an attribute named "Stimulus ," and the values of Stimulus are used to define the stimulus displayed during each trial, the value of Stimulus for each trial will automatically be logged in the data file.

In order to enter a temporary variable (e.g., g_nTotalTrial from the previous step) into the experimental context, either for the purpose of logging that value in the data file or for later access by another object, use the c.SetAttrib command. In the previous step, the value of the g_nTotalTrial variable was set as an attribute (i.e., c.SetAttrib "TrialCount," g_nTotalTrial). The TrialCount attribute may then be accessed by another object using bracket notation, and the value of TrialCount will be logged in the data file.

129
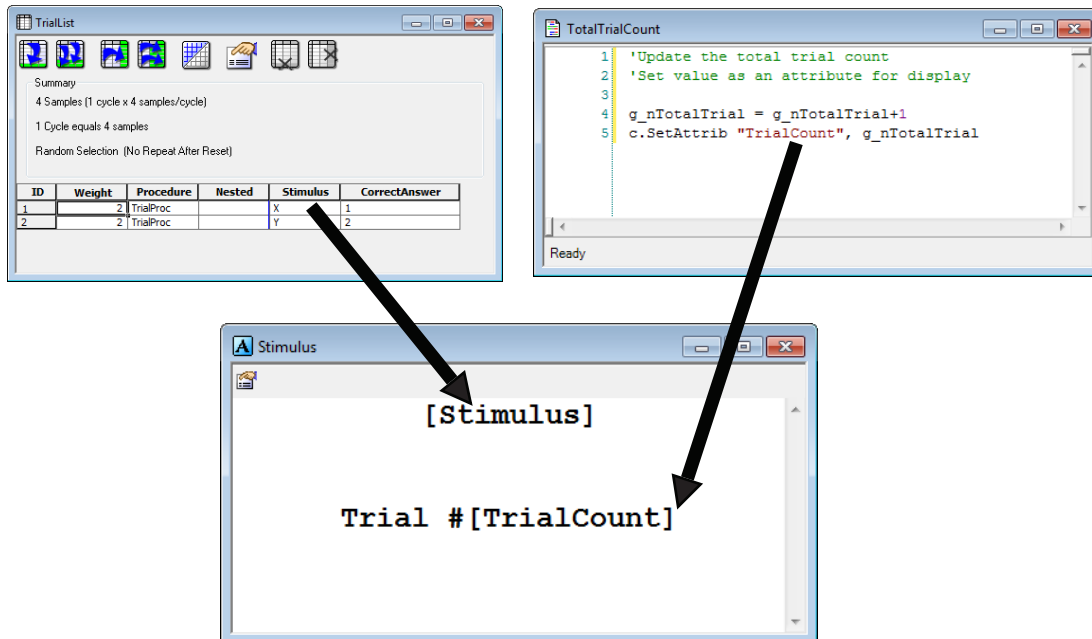
The values of attributes and properties from the experimental context may also be accessed via script on an InLine object using the c.GetAttrib command. For example, the stimulus might display a random number selected from a range of values depending on the condition. "Condition" could be entered as a List attribute defining the possible conditions, and this value could be retrieved at runtime to set the possible range of values for the random number selection.
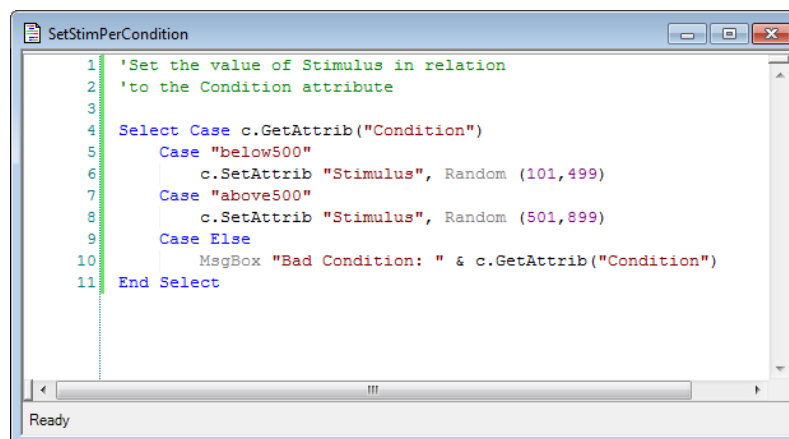
Note the use of the "Else" condition in the script above. It is a good practice, and ultimately the programmer's responsibility, to cover all possible conditions when writing script. If the values of the Condition attribute are carefully entered, the "Else" condition should not be necessary. However, it is better to consider the possibility of error than to have the program fail. Here, we put up a message box to tell the experimenter a bad stimulus condition has occurred.

The properties of objects may also be set or retrieved through script as long as the property is not read-only or design-time only (i.e., not able to be modified at runtime). For example, it is possible to vary messages presented at runtime based on the speed of the response collected. Such a procedure would require accessing one value (i.e., the reaction time from the input object Stimulus.RT), and setting another (i.e., the text to be displayed by the object presenting the message).
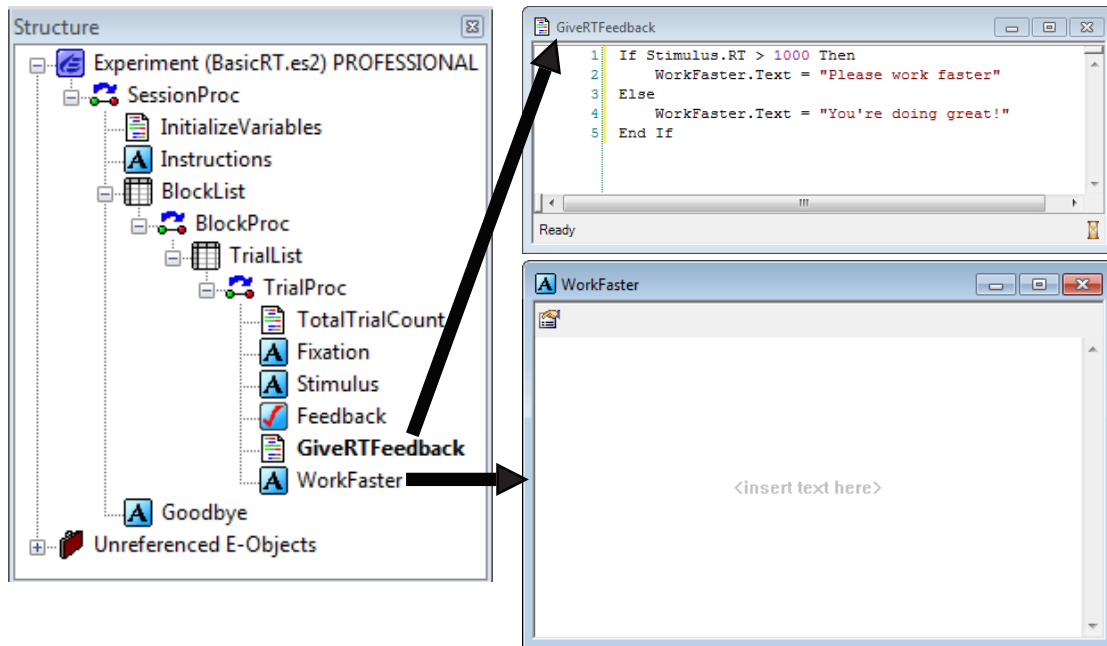
The GiveRTFeedback Inline object containing the script above sets the Text field for the WorkFaster TextDisplay object at runtime. Thus, no value need be entered for the Text field in the WorkFaster object in E-Studio.

Stage 4, Step 5: Reference script results from other objects

After a variable has been entered into the experimental context as an attribute, that attribute may then be accessed by other E-Prime 2.0 objects occurring within the same scope. For example, once the random value has been set as an attribute at the beginning of the trial Procedure, that attribute may be referenced by a TextDisplay object in order to display the value as the stimulus during the trial. To refer to an attribute, use square brackets surrounding the attribute name (e.g., [Stim]) in the Text field of the TextDisplay object.
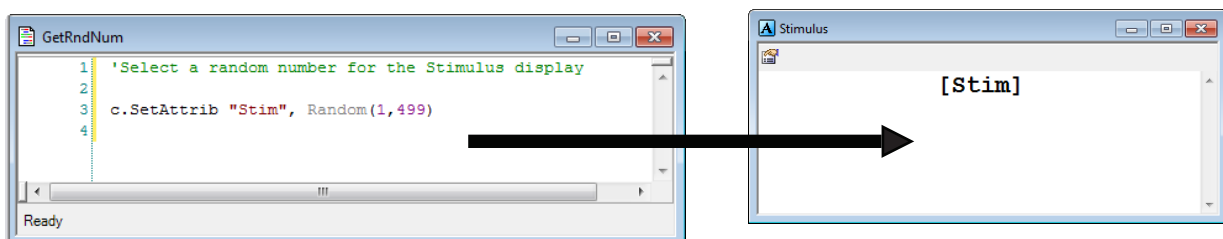
Note that most properties of E-Studio objects may be changed through script as well. For example, to change the location of the display area defined by a TextDisplay object, the X property for a TextDisplay object named TextDisplay1 could be set as TextDisplay1.X = 100. This command sets the horizontal location of the display area to begin at pixel location 100.

Stage 4, Step 6: Debug

Debug commands are useful in tracking down problems, verifying values, or simply reviewing the execution of the program. The Debug object, when used with the Print method, is used to send information at runtime to the Debug tab of the Output window for examination following the run. For example, the following script could be used to write the value of the Stimulus attribute to the Debug tab in the Output window during each trial.
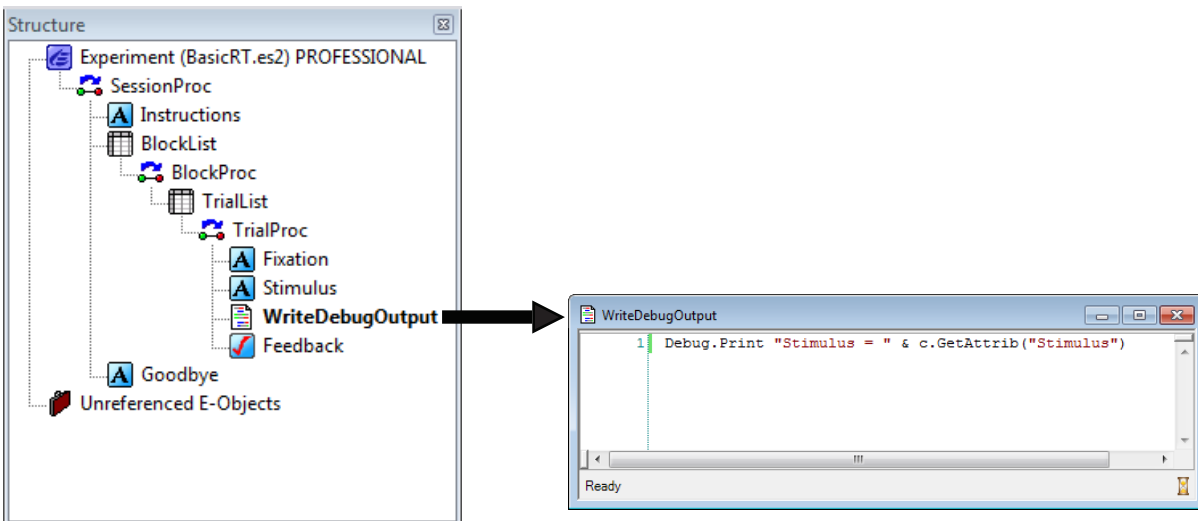
The script above will send the value of the Stimulus attribute per trial to the Debug tab in the Output window. After the run, the Debug output may be viewed by displaying the Output window within E-Studio. Select Output from the View menu, and in the Output window, select the Debug tab.
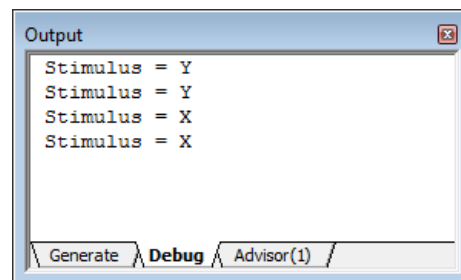
Debug.Print may also be used to monitor timing and response events. The following script uses Debug. Print to write the stimulus onset time and the participant's response to the Output window.
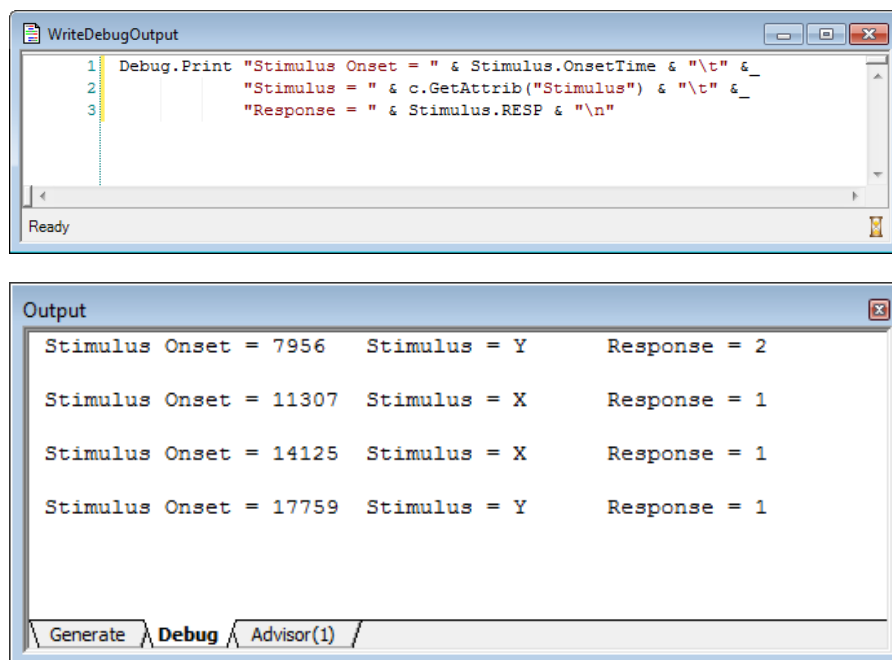
The contents of the Debug tab may be copied to the clipboard by first selecting the lines of information in the Debug tag that are of interest and then right clicking in the Output window. Then, the Debug output may be pasted into Excel to print it or use spreadsheet calculations to check the timing of events.

The Debug object is most appropriate during the development and testing of new programs. After an experiment has been fully tested, the Debug commands may be disabled by setting the Debug object's Enabled property to "false" (i.e., Debug.Enabled = false). This allows the user to leave Debug commands in the script for future testing purposes, but to improve the efficiency of the program during data collection.

A final method of debugging involves the use of comments to indicate the purpose of certain sections of script. This method may be employed more often as a preventative measure than as a diagnostic tool, but can be of tremendous importance during debugging (especially if someone other than the author of the script is given the task of debugging). There is no standard style for adding comments, but in general they should be brief descriptions of the purpose of the segment of script. Many of the script examples used in this chapter make use of brief comments to summarize the purpose of the script for the reader. In E-Basic, comments are separated from script that is generated through the use of the apostrophe (') keyword or the Rem statement. Refer to the Comments topic in the E-Basic Help within E-Prime 2.0 for further discussion of comments.

Stage 4, Step 7: Test

After the experiment is functioning without errors, it is important to completely test it prior to running participants. The Debug object is especially useful for testing an experiment. Use the Debug.Print command to send variable values to the Output window, and keep track of the progress of the experiment with a pencil and paper as well. After running the experiment, examine the output on the Debug tab to view the values. Or, the contents of the Debug tab may be copied to the clipboard and pasted into another application for more thorough viewing or analysis. A written-record and the Debug output should be compared to the data file to verify that all of the variables are being logged, the values are in the correct range, and the sampling is occurring as expected. It is important to test the script completely, including tests of unlikely responses and invalid ranges. For more discussion concerning debugging and testing, refer to *3.7 Stage 7: Testing the Experiment (Page 73).*

5.5 Stage 5: Programming: Basic

This section introduces some fundamental information about writing script in E-Basic. Unless noted otherwise, the small script examples in this section can be placed on an InLine object within any E-Studio example.

Stage 5, Step 1: Logical Operators

Logical operators allow comparison and conditional execution (e.g., If trial > 5 Then…). Often, in script, expressions are used, which compare items using simple mathematical expressions.

>	Greater than
<	Less than
=	Equal to
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Logical operators are also commonly used in compound expressions. Essentially, logical operators evaluate two expressions and use a set of rules to determine if the total expression is true or false.

Operator	Returns True	Example	Result
And	If both expressions are true	5 > 2 And 6 + 3 = 9	True
		3 * 3 = 9 And 7 < 6	False
Or	If either expression is true	5 > 7 Or 8 * 2 = 16	True
		8 < 4 Or 3 > 2	False
Xor	If only one expression is true. Note that it is False if expressions are either both true or both false.	3 + 2 = 5 Xor 5 + 5 >10	True
		3 + 2 = 5 Xor 5 + 5 = 10	False

Stage 5, Step 2: Flow Control

Controlling the flow of the script is a critical component of programming. There are two major types of controlling the flow: conditional statements and loops.

Conditional Statements
Conditional statements determine or specify which part of the script should be executed based on a condition. Contingent branching is often utilized in behavioral research and is based on whether a specific item is true or false. Specifically, If…Then statements are very commonly used to control the flow of the code.

If…Then statements simply are equated to making a choice (e.g., if I have money, then I can go to the movies). If…Then statements are the foundation of all logic. Although they seem to be very simple statements, they are quite powerful in a programming language.

There are actually 3 conditional expressions available in E-Basic. They are: If…Then, Select Case, and Do…Loop. These expressions are conditional statements simply because they perform a task based on a single true or false test. All 3 types of conditional expressions are explored in this section.

If…Then statements
The single most commonly used flow control statement is If…Then. Simply put, an If…Then statement will execute a block of code if the condition is true. If the condition is false, it will do nothing unless "Else" is used in the flow control statement.

```
If condition Then
    <Block of code statements to execute if the condition is true>
End If
```

Rules:
• Notice the "Then" portion of the statement is on the same line as the "If" portion. If there is a need to drop it to the next line, indicate that to the E-Basic compiler. This is done by placing an underscore (i.e., E-Basic line continuation character) at the end of the line to be continued.

• The End If statement is critical. It identifies the last statement in the block of code to be executed based on the condition.

• When the If…Then statement is only a one-line statement, the End If is not used. In fact, it will produce an error if used. In the following two examples, the code works exactly the same, but it is syntactically different.

Example 1: One-line If…Then statement

```
Dim a As Integer
a = 12


If a > 10 Then MsgBox "A is greater than 10."
```

Example 2: If…Then…End If statement

```
Dim a As Integer
a = 12

If a > 10 Then
   MsgBox "A is greater than 10."
End If
```

If…Then…Else statements
If the program must choose between two alternative blocks of code to execute based on the conditional, then the Else statement is included.

```
Dim a As Integer
a = 12

If a > 10 Then
   MsgBox "A is greater than 10."
Else
   MsgBox "A is not greater than 10."
End If
```

Select Case statements
Nested If…Then statements are ideal for testing different values before executing the next block of code. Select Case statements are more appropriate for testing the same value against many different conditions.

```
Select Case variable
   Case test1
        <block of code statements to be executed if the
         value of variable meets test1 criteria>
   Case test2
        <block of code statements to be executed if the
        value of variable meets test2 criteria>
   Case Else
        <block of code statements to be executed if the
         value of variable doesn't meet any of the
         listed Case criteria above>
End Select
```

The Select Case structure indirectly uses condition expressions. An expression may be A + B > C. In the example above, think of the variable being what is to the left of the operator (A + B) and test as everything to the right, including the operator (>C).

Rules:
• An unlimited number of cases may be used as test criteria.
• The Else case is optional. It is not required to be used, but can be useful in behavioral research.

Loops
The loop flow control structures are also quite commonly used in programming. They are useful when a block of code needs to be executed more than once. There are three major types of loops available in E-Basic: Do…Loop, For…Next, and For Each…Next.

Type of Loop	Function
Do…Loop	Repeats the block of code until a condition is true.
For…Next	Repeats the block of code a specified number of times.
For Each…Next	Repeats the block of code for each object within a collection.

Do…Loops
There are a variety of Do …Loops available in E-Basic.

Statement	Description
Do…Loop	Repeats the block of code until a condition is true and then executes an Exit Do not End Do statement.
Do…While…Loop	Repeats the block of code only while a condition is true.
Do Loop…While	Executes the block of code once and then repeats it until the condition is false.
Do Until…Loop	Executes and repeats the block of code only while the condition is false.
Do…Loop Until	Executes the block of code once and then repeats it until the condition is true.

Do While… Loop
The most typically used Do…Loop is the Do While…Loop. The basic syntax is:

```
Do While condition
   <block of statements that are executed while condition is true>
Loop
```

E-Basic evaluates the condition when it encounters a Do While statement. If the condition is true, then the block of code within the Loop structure will be executed. When it reaches the Loop statement, the entire process is repeated including re-evaluation of the condition. When the condition is false, the entire loop structure is skipped and E-Basic executes the next statement immediately following the Loop statement of the structure. In theory, no limit exists on the amount of times the block of code within the structure may be executed.

Do…Loop While
The only difference between a Do While…Loop and a Do… Loop While is the location of the condition. The Do While …Loop evaluates the condition before executing the block of code within the structure. The Do Loop…While evaluates the condition after executing the block of code within the structure. The While in this case determines if the block of code within the structure should be repeated based on the value of the condition. The major resulting difference is that a Do… Loop While will always execute the block of code at least once.

```
Do
    <block of statements that are executed while condition is true>
Loop While condition
```

The Do…Loop While structure is useful when the block of code within the structure sets a value for the condition before it is evaluated. This structure is also useful when performing an action on an item which has more than one element (e.g., a string or an array). Since the item has at least one element, execute the block of code at least once and then repeat it based on the total number of elements within the item.

Do Until…Loop
The Do Until Loop is essentially equivalent to the Do While…Loop structure. They both execute a block of code after evaluating a condition. The Do While structure will repeat a block of code until the condition is false. A Do Until structure repeats a block of code until the condition is true.

```
Do Until condition
    <block of statements that are executed while condition is true>
Loop
```

Do…Loop Until
Like the Do While Loop varieties, the Do Until Loop also offers the option of setting when the condition is evaluated. In this case, the condition is evaluated at the end of the block of code. Therefore, the block of code is executed at least once.

```
Do
    <block of statements that are executed while condition is true>
Loop Until condition
```

Do Loops with If..Then or Select Case statements

Exit Do
Occasionally, the Do Loop structure doesn't quite meet the need for the intended flow. For instance, a loop may need to be broken immediately within the block of code contained within the structure. This is accomplished by nesting an If…Then…End If or Select Case structure within the block of code executed by the Do Loop.

```
Do While condition1
    If condition2 Then
        Exit Do
    End If
        <block of statements that are executed while condition is true>
Loop
```

Exit Do is particularly helpful in debugging code. Specifically, Exit Do will allow the loop to be bypassed without having to manually comment out the entire Do Loop structure.

Do
While the previously described varieties of Do Loops evaluate a condition at either the beginning or the end of a block of code to be executed (and possibly repeated), it is also possible to evaluate the condition within the actual block of code itself. This requires the use of a nested If..Then or Select Case structure.

```
Do
    (block of statements to be executed while in the Loop structure)
    If condition Then
        Exit Do
    End If
    <block of statements that are executed while condition is true>
Loop
```

This process is useful when part of the block of code within the loop should be executed, but not the entire block of code.

For…Next Loops
If the number of times a block of code should be repeated is definite, a For…Next loop structure is most appropriate. With this structure, the loop is repeated based on the start and end values supplied. These values can be integers, variable or expressions. A counter is used to keep track of the number of times the loop is repeated.

```
For counter = start To end
    <block of statements to be executed>
Next counter
```

When the loop begins, the counter is set to start. When the Next statement is executed, the counter is incremented by one. When the counter is equal to the end value supplied, the loop terminates and the next line of code outside the loop structure is executed.

Tips:
• Keep it simple. Unless there is a specific reason to start the counter at another value, use 1 to n.

• Although the counter variable is not required to follow the Next statement, it is good practice to include it. It may seem verbose, but makes parsing through the code a bit easier.

• Avoid changing the value of the counter within the loop structure manually. Let the Next statement increment the counter unless there is a specific reason to change the counter (e.g., set it to the end value to terminate early).

For…Next loops are particularly useful when working with arrays. An array is similar to a storage bin with numbered slots. Arrays are covered in more detail in *5.7 Stage 7: Programming: Advanced (Page 150)*.

Exit For

Similar in concept to Exit Do, the Exit For statement allows the loop to terminate early. This is typically used in conjunction with If..Then and Select Case statements within the For…Next loop.

For Each…Next Loop

This version of the For…Next Loop is similar to the previous version. However, the primary distinction is that it is used to perform a block of code for each element within a set, rather than for a specified number of times. For Each…Next requires an element or variable that corresponds to the object types within the collection.

```
For Each variable In collection
    <block of statements to be executed>
Next variable
```

Notice a counter is not specifically used in this version of a For Loop. Instead, E-Basic figures out how many times to repeat the block of code based on the items in the collection specified. This is particularly useful when debugging. If a problem is suspected with perhaps one of the TextDisplay objects within a block, try 'stepping' through the processing of each object displaying a marker to the screen to help track down the problem.

Interrupting the Flow

GoTo Label

When a design calls for jumping to another place within the script based on a specific flag, the Goto statement is useful. In behavioral research, this is often required for contingent branching experiments. For example, perhaps the execution flow should continue uninterrupted until a participant responds by pressing the "a" key. If the participant presses any other letter, the execution flow should jump, or Goto a specific location within the code. In the example below, a Label is placed prior to an input statement (i.e., AskBox). The input is examined, and if it is not the required input, the execution of the program jumps back to the Label at the beginning of the script in order to perform the response collection again. If the required input is entered (i.e., "a"), the program jumps to a point later in the script.

```
Dim answer As String
LabelB:
answer = AskBox ("Type in a letter:")

If answer = "a" Then
 Goto LabelA
Else
 MsgBox "That is the wrong letter, try again!"
 Goto LabelB   'Ask for another letter
End If

LabelA:
MsgBox "Way to go!"
```

Stage 5, Step 3: Examples and Exercises

To begin adding user code, it is important that the basics of programming are conceptually understood completely. The following examples illustrate the simplicity of E-Basic code at its best. Be sure to follow through these examples before progressing to more advanced topics. It is recommended that each user actually implement each example, run it and verify that it works as expected before moving on to the next section. Some of the initial examples can be created in a blank E-Studio file; later examples should be incorporated into the E-BasicExample.es2 file. Specific instructions are provided for each section.

Example 1: Display "Hello World" on the screen

To run this example, create a new E-Studio experiment which has only an InLine object on the SessionProc. The InLine should contain the script provided below.

```
'The following statement will display a dialog box on the
'screen with the text "Hello World". By Default, an OK
'button is also displayed.
MsgBox "Hello World"
```

The result of the above example is:

Example 2: Setting Attributes in the Context Object

The SetAttrib method is used to create an attribute and assign it a value. Specifically, the SetAttrib method is used to place the attribute in the context so that it may be assigned, modified, and referenced. In the example below, "c" has been defined as the Context object. This is done internally by E-Prime 2.0, and need not be explicitly entered into the script. The SetAttrib method is used in conjunction with the dot operator to declare TotalTrial as an attribute of the context object ("c") and assign it a value of 10.

As with Example #1 above, to run this example you should create a new E-Studio experiment with an InLine object on the SessionProc and enter script below.

```
'Set TotalTrial=10
c.SetAttrib "TotalTrial"," 10
```

Once an attribute is put into the context, the information is logged in the data file and the attribute may be used to display information by way of a display object (e.g., a TextDisplay).

The TotalTrial attribute will be available in the context during the scope of the level at which it has been defined. For example, if TotalTrial is defined (using SetAttrib) during the block Procedure, it will be available during the context of the block level, and any context levels subordinate to the block (e.g., trial level, sub-trial, etc.). However, outside of that scope, the TotalTrial attribute value will not be available. There is no backward inheritance possible, which would allow higher levels to inherit attribute information from lower levels.
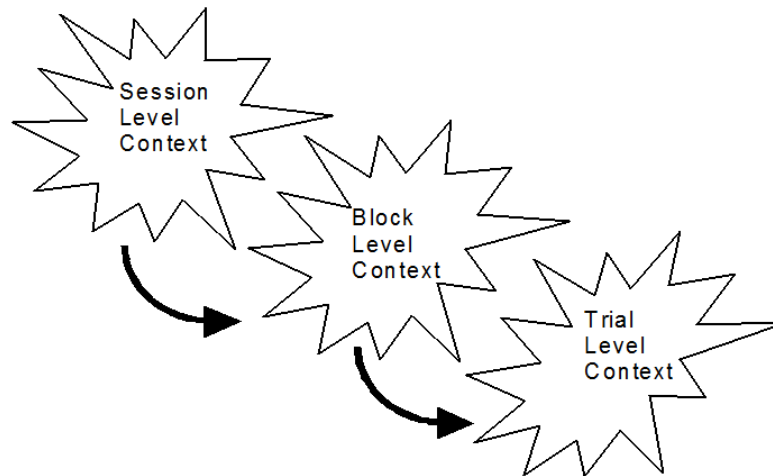
An attribute must be defined before it is referenced, or error messages will result indicating that the attribute does not exist. For example, if an attribute is defined at the trial level, and referenced at the block level (prior to the trial), an error will occur related to the declaration of the attribute.
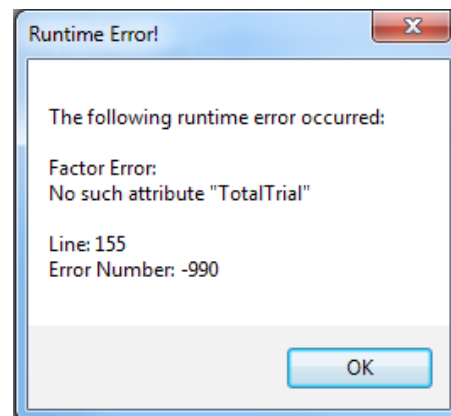
The inheritance of the value to assign to an attribute follows the hierarchical structure, and values may only be inherited by levels lower than the level at which the attribute is defined. Thus, if an attribute is defined at the block level, it may be referenced at the trial or sub-trial level. Inheritance occurs in a downward direction, while the search for a value occurs in an upward direction. For example, if an attribute is defined at the block level and referenced at the trial level, the value at the trial level will be inherited from the block level (i.e., downward). The resolution of the value occurs by first searching the trial (i.e., current) level, then continuing the search at the next highest level (e.g., block), and upward until the value is resolved.

Example 3: Getting Attributes from the Context Object
The GetAttrib method is used to retrieve a value for an existing attribute in the context. Like SetAttrib, the GetAttrib method is used in conjunction with the dot operator.

In the example below, the TextStimulus object displays a stimulus "X" or "Y." In the script, the IF… THEN clause is used to evaluate the current stimulus and set its display color. The GetAttrib method will retrieve the value of the current stimulus. When GetAttrib returns a value of "X" the ForeColor property of the TextStimulus is set to "Green," so that all X's will be displayed in the color green. When GetAttrib returns a value of "Y," the ForeColor property is set to "Blue" so that all Y's will be displayed in the color blue.

141

To run this example, open the E-BasicExample.es2 file, copy the script below into the PreStimulus InLine object, and run the experiment.

```
'Retrieve value of "Stimulus" and set display color

If c.GetAttrib ("Stimulus") = "X" Then
ElseIf c.GetAttrib ("Stimulus") = "Y" Then
End If
```

Example 4: Global Variables

Often, it is desirable or useful to determine the participant's performance over the course of the experiment, or perhaps after a specified number of blocks or trials. One method of assessing performance is to use a FeedbackDisplay object, which can automatically calculate summary statistics (e.g., mean accuracy, mean RT, etc.). Another method of assessing performance involves the use of the Summation object. This method involves more involvement from the user, because user-written script is required, but it enables you to control precisely if and when the participant is presented with performance feedback. (While the FeedbackDisplay object performs all of the performance calculations automatically, it also presents feedback automatically.) The example below uses a Summation object to determine average accuracy after a specified number of trials, but without presenting feedback to the participant. This example works with the EBasicExample.es2 file.

To use a Summation object to determine the average accuracy per condition or block, declare the Summation object on the User tab in the Script window. In most cases, accuracy would only be examined after a minimum number of trials. Thus, in order to start evaluating mean accuracy after a certain number of trials had been run, it be is also necessary to declare a counter to manually count the number of trials that have occurred. In the script below, the PracticeProp summation variable is declared for use in evaluation of the practice trial performance; the TrialCount integer variable is declared to keep track of the running trial count. Again, enter this script on the User tab in the Script window.

```
'Declare Variables
Dim PracticeProp As Summation
Dim TrialCount As Integer
```

Once declared in the User Script window, the variables are available globally, or for the scope of the entire experiment. The variables must be initialized prior to the point in the experiment at which they are referenced. This is accomplished using an InLine object, inserted on the Session Procedure timeline. The InLine may occur at any point prior to the referencing of the variables. However, it is a good practice to insert the initialization InLine as the first event in the Session Procedure, which can serve to set up or initialize any variables that will be used. This is how the Setup InLine object in E-BasicExample.es2 file is designed.

In the example below, the Set command is used to initialize the PracticeProp summation variable. This command defines the variable as a new instance of an existing object type. The TrialCount variable is initialized to zero since no trials have yet been run. Enter this script on the Setup InLine object.

```
'Initialize Summation Variable
Set PracticeProp = New Summation

'Initialize TrialCount Variable
TrialCount = 0
```

Once initialized, the variables may be assigned values and referenced within the context in which they were defined. In this case, the defined context was the top-level context (i.e., User tab at the experiment level). Thus, the variables are defined globally and may be referenced at any point in the program.

The script below illustrates how the Summation and counter variables are assigned values on each trial. The counter is manually incremented by one on each trial. The Summation variable collects accuracy statistics across trials during the entire block. In the script below, the responses are collected by the "TextStimulus" object. Thus, the InLine that contains this script follows the Stimulus object on the trial Procedure, which is the PostStimulus InLine object in E-BasicExample.es2.

```
'Increase counter by 1
TrialCount = TrialCount + 1

'Add accuracy stats to summation variable

'When trial count = 5, evaluate accuracy stats
If TrialCount >= 5 Then
 'If accuracy is 80% or better exit block
 If PracticeProp.Mean >= .80 Then
 TrialList.Terminate
 End If
End If
```

At five trials or more, the mean of the values collected by the Summation is evaluated. If mean accuracy is greater than 80%, the currently running List (i.e., TrialList) is terminated. Thus, the script examines the overall accuracy of the block and terminates the block when a minimum accuracy of 80% is reached.

Example 5: Trial Level Variables
Declaring global variables on the User tab of the script window permits the reference of these variables at any point in the experiment. Conversely, variables to be used only within a specific context (e.g., block or trial Procedure) may be declared using the Dim command in an InLine object in the appropriate Procedure. In the script below, the Dim command is used to declare a variable which collects a response on each trial. This script is entered in an InLine object, which is called from the trial Procedure object.

```
'Collect response from AskBox
Dim strAnswer As String

strAnswer = AskBox ("Type in the recalled word:")
RecallStim.RESP = strAnswer
RecallStim.CRESP = c.GetAttrib("CorrectAnswer")
```

Example 6: Using Attribute References to Pass Information
Attributes allow the passing of variable information during a Procedure. For example, in order to vary the stimulus presented per trial, the stimulus values (e.g., text, name of the picture file, or name of the audio file) would be entered as separate exemplars for an attribute (e.g., Stimulus) on a List object.
In order to obtain the value of a specific exemplar within the Stimulus attribute listing, the c.GetAttrib command is used. In the example below, a basic Stroop task is presented in which the display color of the word varies per trial. The valid display colors (Red, Green, Blue) are entered as RGB values in the "Ink" attribute on a List object.

5.6 Stage 6: Programming: Intermediate

Stage 6, Step 1: More on Variables

There is much more information pertaining to variables in addition to what has been presented thus far that should be learned. Rather than overwhelm the beginning programmer, the information is presented in this chapter by stages of complexity. Intermediate programmers will need to use variables on a frequent basis. The following section details more information on declaration and utilization of variables.

⚠️ **NOTE:** *These examples are designed to illustrate the syntax of E-Basic variable declarations only, and are not intended to be implemented and generated.*

DataTypes
When using the Dim statement to declare a variable, more information is required than just the name. The type of data the variable can hold must be specified. The script below not only reserves the word "participant_dob" as a variable, it also indicates that the variable is to hold a date (i.e., the participant's date of birth).

```
Dim participant_dob As Date
```

There are a variety of data types available within E-Basic. The following table illustrates the type and an explanation of the type:

Data Type	Description
Boolean	True (-1) or False (0) value
Integer	Whole number ranging from –32767 to 32767
Long	Whole number ranging from –2,147,483,648 to 2,147,483,647
Single	Used to declare variables capable of holding real numbers with up to seven digits of precision: • Negative: -3.402823E38 to -1.401298E-45 • Positive: 1.401298E-45 to 3.402823E38
Double	Used to declare variables capable of holding real numbers with 15–16 digits of precision: • Negative: –1.797693134862315E308 to –4.94066E-324 • Positive: 4.94066E-324 to 1.797693134862315E308
Currency	Used to declare variables capable of holding fixed-point numbers with 15 digits to the left of the decimal point and 4 digits to the right (-922,337,203,685,477.5808 to 922,337,203,685,477.5807)
Date	Used to hold date and time values
Object	Used to declare variables that reference objects within an application using OLE Automation
String	Used to hold sequences of characters, each character having a value between 0 and 255. Strings can be any length up to a maximum length of 32767 characters.
Variant	Used to declare variables that can hold one of many different types of data. Refer to the Variant data type topic in the E-Basic Help
User-Defined	Requires the Type statement

Conversion between data types

Once a variable has been declared as a certain data type, it may hold only information of that type. While E-Basic will perform numeric type conversions automatically, which means a runtime error will not occur, it is not good practice to rely on such automated conversions, in part because the results may not always be what you expect. At times, it may be necessary to convert information from one type to another in order to store it in a particular variable. E-Basic contains several functions for the purpose of conversion between data types.

Function	Description
CCur	Converts any expression to a Currency.
CBool	Converts expression to True or False, returning a Boolean value.
CDate, CVDate	Converts expression to a date, returning a Date value.
CDbl	Converts any expression to a Double.
CInt	Converts expression to an Integer.
Chr, Chr$, ChrB, ChrB$, ChrW, ChrW$	Returns the character whose ASCII value is charcode.
CLng	Converts expression to a Long.
CSng	Converts expression to a Single.
CStr	Converts expression to a String.
Cvar	Converts expression to a Variant.
Hex, Hex$	Returns a String containing the hexadecimal equivalent of number.
Str, Str$	Returns a string representation of the given number.
Val	Converts a given string expression to a number.
⚠ **NOTE:** *Some functions offer the optional use of the "$" character. When the "$" character is used, the value returned is a string. When "$" is not used, a string variant is returned.*	

Declaring Multiple Variables
Users are not limited to one variable declaration per line. Multiple variables may be declared on a single line using only a single Dim statement. However, the user should take care to specify the data type for each variable declared, even if they are on the same line. Any variable that is not specifically declared with a data type will be declared as a Variant. There are times where variants are useful, but variant data types require more memory than other data types and should be used with care. (Variants are described in more detail in the Advanced Programming section below.) In the example below, participant _dob is declared as a date, Stim is declared as a string and k is declared as an integer. The j variable is declared as a variant because it is not specifically stated to be any other data type.

```
Dim participant_dob As Date, j, k As Integer, stim As String
```

Declaring multiple variables on the same line may save space, but it also increases the likelihood of human error. A compromise might be to not mix data types within a single line; this can help to organize variables and reduces the chance of error. An example of this type of declaration is shown below:

```
Dim participant_birthdate As Date
Dim j As Integer, k As Integer
Dim stimulus As String, Dim recall As String
```

Initialize and Assign Values

Once a variable is declared, it must be initialized before it can be used. Initializing a variable is nothing more than assigning it an initial or starting value. Most often the purpose of a variable is to hold information that is assigned. An assignment statement simply consists of the variable name followed by an equal sign and then the expression (variable name = expression). In the example below, the counter variable "j" (declared in a previous example) is assigned the expression or value of 1.

```
j = 1
```

String values are also assigned using an assignment statement, but the expression value must be enclosed in quotes.

```
Dim stringVal As String
stringVal = "This is the value of the string."
```

If the expression extends past a single line, the expression statement must be divided into separate strings, which are concatenated. The ampersand (&) is used for this purpose. No linefeeds or carriage returns are included in the concatenation of strings, unless the new line (\n) character is included.

```
stringVal = "This is the value of a very long string "&_
        "extending over more than one line. The "&_
        "individual parts will be joined to form "&_
        "a continuous display." &_
        "\n\nThis will be displayed two lines lower"
```

Variables themselves may be used in expression statements. For example, in an assignment statement, the current value of a variable could be used to set the value of another variable.

```
Dim i As integer, j As Integer, k As Integer
j = k * i
```

Variables themselves may be used in expression statements. For example, in an assignment statement, the current value of a variable could be used to set the value of another variable.

Dim i As integer, j As Integer, k As Integer
j = k * i

In this case, rather than making a literal assignment to the variable j, the current values of the "k" and "i" variables are determined at the point that the script is executed, and those values are used to calculate the value of "j."

Variables may also be used to pass information to a command or function. For example, the value of stringVal defined above could be used with the MsgBox command to display the intended string.

```
MsgBox stringVal
```

147

Constants

A constant is used when requiring the use of a value that does not change. When working with a value that does not change, you could technically use a variable to hold the value, but it makes more sense to use a constant because it cannot be modified. To declare a constant, use a Const statement in the same manner a Dim is used to declare a variable. The only difference between the Const and the Dim declaration is that a value is specified immediately after the data type.

```
Const speed_of_light As String = "Really, really fast!"
Const exercise As Boolean = True
```

Stage 6, Step 2: Writing Subroutines

Subroutines are composed of a series of commands combined into a unit. This unit may then be run by a call to the subroutine from within an InLine object. A subroutine is defined using the Sub…End Sub statement. For example, a simple subroutine may be created to "clear" the background of a Canvas object to a particular color.

Example: Clear the screen to the current color

In the script below, the ClearToRed subroutine sets the fill color to red, and then uses the Clear command to clear the screen using the current FillColor setting. Load the E-BasicExample.es2 file and enter the following script on the User script tab in the Script window. Once defined, the subroutine is available to any InLine object, and the ClearToRead subroutine can be run by referring to it by name.

```
'Subroutine containing the script necessary to set the
'background to red.

Dim cnvs As Canvas

Sub ClearToRed
   cnvs.FillColor = CColor("Red")
   cnvs.Clear
End Sub
```

In the example below, the ClearToRed subroutine is called to quickly set the background to red before ten circles are drawn on the Canvas. The script below would be placed in an InLine object called during the experiment. To run this example, place the following script on the Setup InLine object in E-BasicExample.es2.

```
      'Clear the screen to red and draw 10 circles of random
      'size.

      Dim i As Integer
      Dim x As Integer
      Dim y As Integer
      Dim rad As Integer

      Set cnvs = Display.Canvas
      ClearToRed

      x = 50
      y = 100

      For i = 1 To 10
        cnvs.Pencolor = CColor("white")
        cnvs.Fillcolor = CColor("white")
        rad = Random (3, 20)
        x = x + 50
        cnvs.Circle x, y, rad
      Next I

      Sleep 1000
```

Subroutines are most useful when a section of script is used repetitively. The use of subroutines aids in the prevention of errors, and minimizes script maintenance.

Stage 6, Step 3: Writing Functions

Like subroutines, functions are units composed of a series of script commands. Functions differ from subroutines in that they may be used in a command, and may return a value (e.g., if the function requested a response from the user, or performed a data transformation).

Example: Calculate a mean value
In the example below, the DoMean function is passed two parameters (total and count). Total is divided by count (using the "/" operator) to determine the value for DoMean. This script is entered on the User tab of the Script window to define the DoMean function.

```
      Function DoMean(total As Double, count As Integer)As Double
        DoMean = total/count
      End Function
```

Once it is entered in the User Script window, the DoMean function may be used at any time during the experiment. The CalcMean InLine object below calls the DoMean Function to calculate the mean of 5 randomly chosen numbers. To run this example, after entering the script above on the User tab of the Script window, enter the script below on the Setup InLine object.

```
        Dim total As Double
        Dim count As Integer
        Dim i As Integer
        Dim next_val As Integer

        total = 0
        count = 5

        For i = 1 To count
          next_val = Random (1, 20)
         MsgBox "Value #" & i & ": "& next_val
          total = total + next_val
        Next i

        MsgBox "The total is " & CStr(total) & "\n" &_
            "The count is " & CStr(count) & "\n" &_
            "The mean is " & DoMean (total, count)
```

5.7 Stage 7: Programming: Advanced

Before moving to this section, you should be comfortable working with variables and data types. This section introduces the use of arrays and user defined data types. These are very powerful features, but understanding the basics from the prior two sections is essential before you can fully utilize the information in this section

Stage 7, Step 1: Arrays

It is often the case that multiple pieces of information need to be used as a single variable. The best way to handle this is through the use of an array. An array is a storage unit for many items of the same type. The storage unit is comprised of multiple items of one type, each being stored in their own index within the array. Think of it as a filing drawer that may contain many files in a specific order which all pertain to a single topic. The array would be the drawer while the individual files within would be the indices.

⚠ **NOTE:** *These examples are designed to illustrate the syntax of E-Basic array declarations only, and are not intended to be implemented and generated.*

Declaring Arrays
To work with an array, refer to the name of the array and the index. An array can be of any data type, but can only hold a single data type. Specifically, you can declare an array that holds strings and an array that holds integers, but you cannot declare a single array that holds both strings AND integers. However, if you need to store multiple data types in a single array, you may declare arrays of variant data type, which holds any kind of data. Be careful though; since variant data typically is more memory intensive than other data types, creating and using an arrays of variants can potentially create an overwhelming amount of overhead. Use this option wisely.

As with any variable, before using it, first declare and initialize it. The declaration of an array is similar to any other variable. The Dim statement is used, and the only significant difference is that the array declaration also takes a dimensions value.

```
     Dim position_array ( ) As Integer
```

Fixed arrays

The dimensions of fixed arrays cannot be adjusted at execution time. Once declared, a fixed array will always require the same amount of storage. Fixed arrays can be declared with the Dim statement by supplying explicit dimensions. The following example declares a fixed array of eleven strings (arrays are zero-based) see Using an Array Index below):

```
Dim a(10) As String
```

Fixed arrays can be used as members of user-defined data types. The following example shows a structure containing fixed-length arrays:

```
Type Foo
   Rect(4) As Integer
   Colors(10) As Integer
End Type
```

Only fixed arrays can appear within structures. Refer to *Stage 7, Step 2: User-Defined Data Types (Page 153)* for a discussion of user-defined types.

Dynamic arrays

Dynamic arrays are declared without explicit dimensions, as shown below:

```
Public Ages() As Integer
```

Dynamic arrays can be resized at execution time using the ReDim statement:

```
ReDim Ages (100)
```

ReDim modifies the dimensions of an array, specifying a new upper and lower bound for each dimension. After they are declared, dynamic arrays can be redimensioned any number of times. When redimensioning an array, the old array is first erased unless the Preserve keyword is used, as shown below:

```
ReDim Preserve Ages (100)
```

Dynamic arrays cannot be members of user-defined data types.

Using an Array Index

Items within an array are indexed beginning with zero. In other words, the first element within an array is located within index number 0. For instance, if an array is designed to hold 10 elements, the array should be dimensioned as in the following:

```
Dim arrResponses (9) As String
```

In the previous statement, the arrResponses array is dimensioned to hold 10 elements. Because array indices begin at 0, the "9" in the dimension statement indicates the largest legal index within the array, and the total number of elements that the array may contain is one greater than this number.

Addressing an element within an array
The individual elements within an array accessed or set simply by listing the array name, followed by subscript notation (i.e., the index number enclosed in parentheses) to refer to the appropriate index. The index of the array includes an integer value for each dimension of the array. For example, my_array(3) refers to, or identifies the value in the fourth slot in the array called my_array Data contained within an array may be used like any other variables:

Assign a value to an array element.

```
Dim a(9) As Integer
a(0) = 12
```

Assign a value stored in an array to another variable.

```
x = a(0)
```

Use the value of an array element in an expression:

```
x = 10 * a(0)
```

Assigning Data to Array indices
When an array is declared using the Dim statement, the elements composing the array are not initialized. That is, the elements contain no valid information. Before accessing the array elements, they must be assigned meaningful values. Array elements are assigned values using an assignment expression (i.e., ArrayName(Index) = Expression).

```
Dim a(9) As Integer
a(0) = 12
```

The most efficient method of assigning values to an entire array at one time (e.g., to initialize an array to consecutive values) is to use a For…Next loop.

```
ReDim Preserve Ages (100)
```

Dynamic arrays cannot be members of user-defined data types.

Using an Array Index
Items within an array are indexed beginning with zero. In other words, the first element within an array is located within index number 0. For instance, if an array is designed to hold 10 elements, the array should be dimensioned as in the following:

```
'create an array with 10 indices, numbered 0 through 9
Dim a(9) As Integer
Dim x As Integer
Dim i As Integer

For i = LBound(a) To UBound(a)
   a(i) = x
   x = x + 1
Next i
```

Arrays may also be multi-dimensional. Arrays containing more than one dimension are similar to a spreadsheet-like organization, with different dimensions handling different tables or lists of information. Multi-dimensional arrays are declared just like one-dimensional arrays, with commas separating the values specifying the size of each dimension in the array.

```
Dim multi_array(9, 6, 8) As Integer
```

The total number of elements held by a multi-dimensional array is equal to the product of the sizes of the individual dimensions. The example above would result in an array holding 630 elements (10 by 7 by 9).

Stage 7, Step 2: User-Defined Data Types

E-Basic allows the user to define data types. A user-defined data type is declared using the Type statement, and the data items organized by the data type are listed within the Type statement. User-defined data types must be declared on the User tab of the User Script window; they cannot be declared on an In-Line object.

User-defined data types are very useful for organizing related data items of various types. For example, to draw and modify a grid in which some of the cells are drawn in color, it would be useful to keep track of an ID number for each cell, the color in which each cell is drawn, the location of the cell, and other possible information about each cell. The script below uses the Type statement to declare the CellInfo data type, organizing the data relevant to each cell in the grid.

```
Type CellInfo   'keep track of cell info
  nID As Integer
  nColorState As Integer    'Is the cell a color or non-color
  nColor As Long            'Specific color used to draw the cell
  nRow As Integer           'Row in the grid where cell appears
  nColumn As Integer        'Column in the grid
End Type
```

Within the Type declaration of the CellInfo data type, variables are declared to organize the information pertaining to each cell (i.e., the cell ID, color of the cell, row, column, etc.). Once the data type has been declared, the Dim command is used to declare a new instance of the type. For example, the script below declares the CurrentCell variable as an instance of the CellInfo type (i.e., a single cell in the grid).

```
Dim CurrentCell As CellInfo
```

If the CurrentCell variable declaration is made on the user tab of the Script window, after the CellInfo type definition, then the CurrentCell variable becomes a global variable. Alternatively, if the variable declaration is made on an InLine object, then it is participant to the scope rules that were described in *Stage 3, Step 4: Variable Declaration and Initialization (Page 125)*.

Like assigning values to object properties, the dot operator is used to assign values to the component variables of a user-defined type. Below, the CurrentCell variable is assigned values for the ID number, row, column, color state, and color components.

```
'Define cell info
CurrentCell.nID = 12
CurrentCell.nRow = 2
CurrentCell.nColumn = 3
CurrentCell.nColorState = 1          '1 = color, 0 = white
CurrentCell.nColor = CColor("Blue")
```

(Recall that the assignment script can only occur on an InLine object; this script can NOT be placed on the User tab of the Script window.)

Stage 7, Step 3: Advanced Programming Examples

Contingent Branching
Launching a specific Procedure based on the participant's response

This example assumes a structure in which an input object named "participant" collects a response, and two separate List objects (List1 and List2) call separate Procedures.

```
'If the participant enters "1" run Procedure 1, otherwise run
'Procedure 2.

If participant.RESP = "1" Then
    List1.Run
Else
    List2.Run
End If
```

Arrays
Creating a single dimension array, assigning values, and accessing values for display

To run this example, enter the following script on the Setup InLine object in E-BasicExample.es2.

```
'Creating and assigning values to a single dimension array

Dim WordList(4) As String
Dim i As Integer

WordList(0) = "Every"
WordList(1) = "good"
WordList(2) = "boy"
WordList(3) = "does"
WordList(4) = "fine"


For i = 0 To 4
    MsgBox WordList(i)
Next i
```

Debugging

Using Debug.Print to verify logging accuracy an input object named "TextStimulus" which collects a response. To run this example, open E-BasicExample.es2 and enter the following script on the PostStimulus InLine object.

```
'Evaluate the response collected by the TextStimulus object.
'Send the response entered by the participant (RESP), the
'correct response (CRESP), and the accuracy (ACC) to the
'OUTPUT window separated by tabs. View using the DEBUG tab
'in the OUTPUT window.

Debug.Print TextStimulus.RESP & "\t" & TextStimulusStimDisplay.CRESP &_
        "\t" & TextStimulus.ACC
```

The script above will send information to the Debug tab in the Output window as follows:

```
Output                                    ☒
  1        1        1
  2        2        1
  2        2        1
  2        2        1
  1        1        1
  1        1        1



  \ Generate /\ Debug /\ Advisor(3) /
```
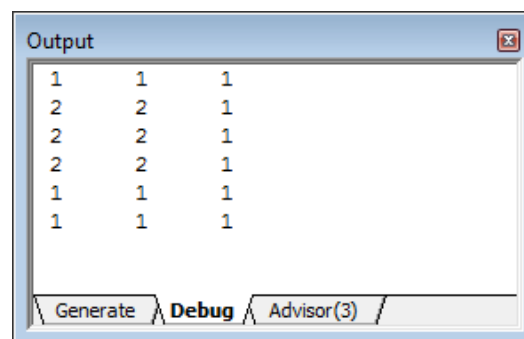
5.8 Stage 8: Debugging in E-Prime 2.0

Errors may occur during both the generation and the running of an experiment. This section describes how to access the debugging messages in both situations.

Stage 8, Step 1: Generation errors within E-Studio

When an experiment is generated within E-Studio, the Output window provides feedback concerning the status of the generation procedure. The Output window may be displayed using the View menu in E-Studio. Within the Output window, the Generate tab displays information concerning the generation process of the program. For example, if an experiment is generated without errors, the Generate tab in the Output window will display messages indicating that the script was generated successfully.

```
Output                                                                          ☒
  -Running Module:   FeedbackDisplay may report invalid stats
  -Running Module:   Use of ClearAfter
  -Running Module:   Use of Stretch
  -Running Module:   Display adapter set to clone mode
  -Running Module:   Visual Slide objects not using percentage values for coordinates
  -Running Module:   Infinite Duration with no end action
  -Running Module:   Varying filename not found
  -Running Module:   Use of (end of proc) with FeedbackDisplay
  -Running Module:   Use of (until feedback) when no FeedbackDisplay available
  -Running Module:   Procedure is not collecting data
  -Running Module:   DisplayDevice uses Match Desktop
  End:   Finished Processing Experiment Advisor Modules

  Compiling script with E-Run (please wait)...
  Script Compiled by E-Run successfully.


  \ Generate /\ Debug /\ Advisor(3) /
```

When any errors are produced as a result of generating script within E-Studio, the errors will be reported by a dialog box.
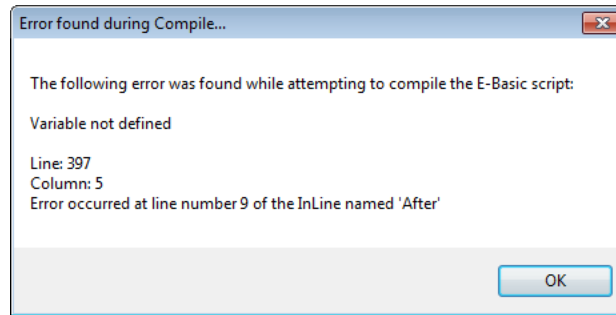
The errors will also be sent to the Debug tab in the Output window. After dismissing the error dialog, the errors may be re-displayed by opening the Output window (View menu) and clicking the Debug tab.
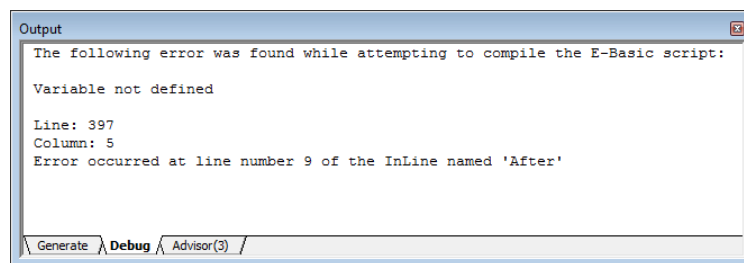
Each error will display the line number in the script at which the error occurred, and the Script window will open automatically in E-Studio to display the full experiment script. Within the Script window, the cursor will blink at the line at which the error was encountered.
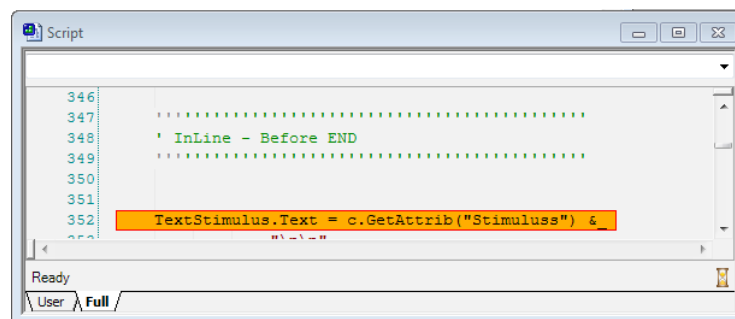
Stage 8, Step 2: Runtime errors

When errors occur during the running of an experiment (i.e., runtime error), a dialog will be displayed indicating the runtime error, and the line in the script at which the error occurred. As with generation errors, runtime error messages are sent to the Output window, and the Script window is opened in E-Studio with the cursor placed at the error location.
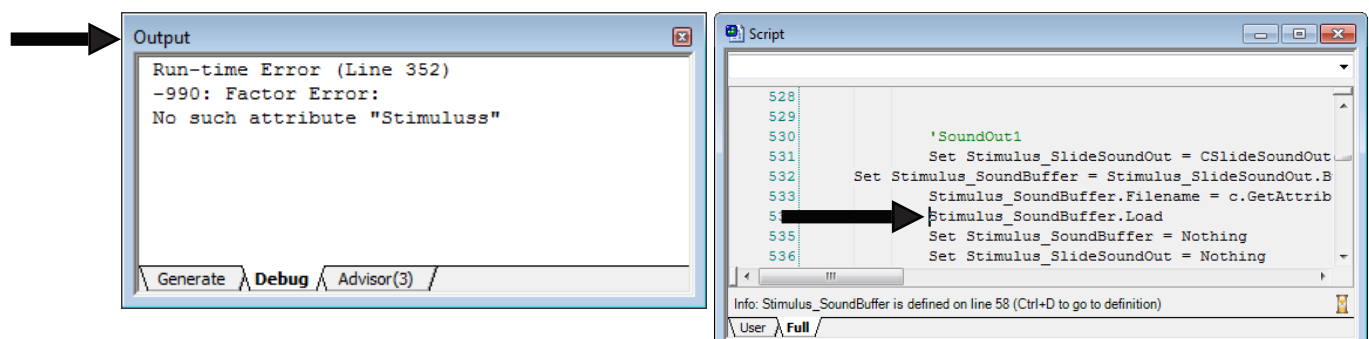
Stage 8, Step 3: Tips to help make debugging easier

• Run in fixed order to verify proper stimulus selection before adding randomization.

• Think small -- test small portions of the program for functionality before expanding the program.

• Use Debug.Print to send information to the Output window during runtime. This information may be evaluated after the run terminates in order to verify values.

• In an Inline object, use Display.Canvas.Text in conjunction with a Sleep command to display debugging information to the screen at runtime without requiring input from the user to continue.

• The MsgBox command may be used to display values during the run of an experiment. The MsgBox will not allow the program to continue until the user presses enter to dismiss the dialog box.

• Assign values as attributes so that they may be displayed on a TextDisplay or Slide object during the run of an experiment, or logged to the data file for later examination.

Chapter 6: Data Handling

6.1 Stage 1: Overview of Data Handling

With E-Merge and E-DataAid, E-Prime 2.0 provides tools to support data management and analysis. The E-Merge application merges individual data files into aggregate data files for group analysis. The E-DataAid application enables experimenters to view and edit data, generate descriptive statistics and plots of the results, and export the data and analysis results to other applications. E-DataAid provides very rapid checking of the data collection process, monitoring of the experimental results, and fast export to other packages for extended analysis.

It is recommended that the user first work through the *E-Prime Getting Started Guide* to get an overview of the use of E-Merge and E-DataAid. The materials in this chapter detail more extended options within the tools. If data merging needs are modest and are covered sufficiently by the *E-Prime Getting Started Guide*, the user may choose to skip to *6.3 Stage 3: Data Handling Using E-DataAid (Page 170)*.

6.2 Stage 2: Merging Data Files Using E-Merge

Stage 2, Step 1: Introduction

E-Merge is E-Prime 2.0's data merging application. For each participant run, E-Run generates a single participant E-Prime 2.0 data file with the .edat2 extension. Using a graphical interface similar to Explorer, E-Merge allows the user to merge these files quickly into a master E-Prime 2.0 data file for data analysis. Merged E-Prime 2.0 data files have the .emrg2 extension.
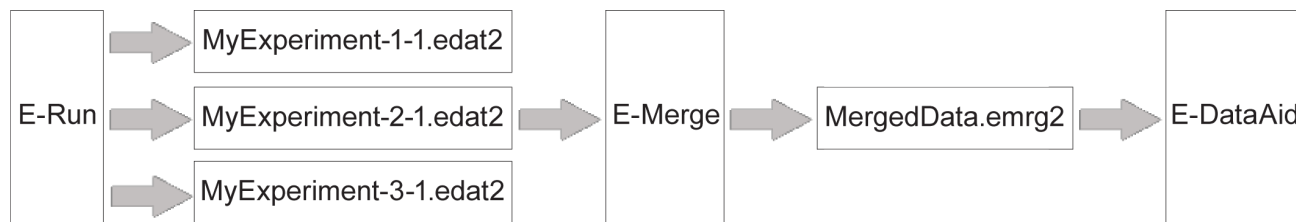
This chapter outlines the steps for data merging and introduces some advanced topics, such as using the Recursive Merge feature and handling conflicts, which are not addressed in the *E-Prime Getting Started Guide*. Before reading this chapter, work through the *E-Prime Getting Started Guide* for E-Merge and have a general knowledge of working in a Windows application.

Stage 2, Step 2: Organize Data Files

Before merging data files, the files should be organized into one folder or a system of subfolders on the hard drive. The organization of files will depend on the needs of the particular experiment. For example, data files for a single experiment may be placed in one folder (e.g., Figure 1), or may be separated into subfolders according to some grouping (e.g., condition), as in Figure 2 below.
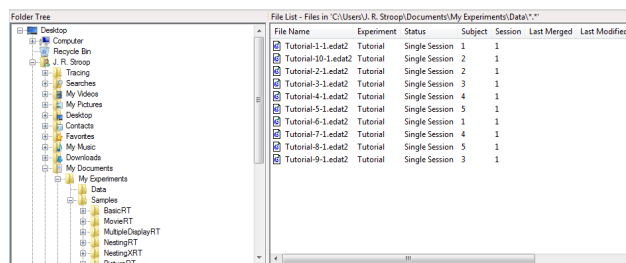
Figure 1. Single folder organization.
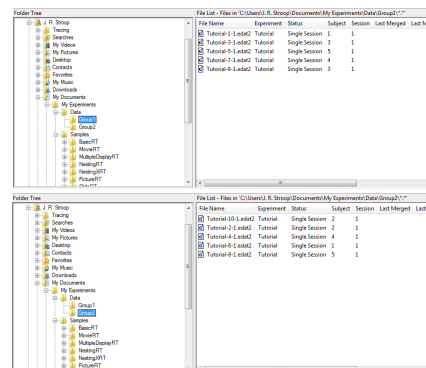
Figure 2. Subfolder organization, organizing data files for participants receiving different conditions.

Stage 2, Step 3: Merge

E-Merge offers two merge operation options: Standard and Recursive. The Standard merge operation (demonstrated in the *E-Prime Getting Started Guide*) merges data files from a single folder into a target file. This is the most common type of merge operation. The Recursive Merge operation merges data files existing in separate subfolders into a target file. The Recursive Merge operation permits the merging of files located in multiple subfolders using a single merge operation, rather than requiring multiple Standard merge operations.

Standard

To merge data files in E-Merge using the Standard merge operation, open the data folder in the Folder Tree, select the files to merge in the File List view, and click the Merge button to merge.

⚠ **NOTE:** *The images used throughout this example reflect operations upon data files included with the E-Prime 2.0 installation. With the default installation, these files exist in the …\My Experiments\Tutorials\ Data folder.*

Open Folder

Within E-Merge (refer to the *E-Prime Getting Started Guide* for information concerning launching the E-Merge application), click on the folder containing the data files in the Folder Tree. When the folder is selected, the files in that folder will be displayed in the File List.
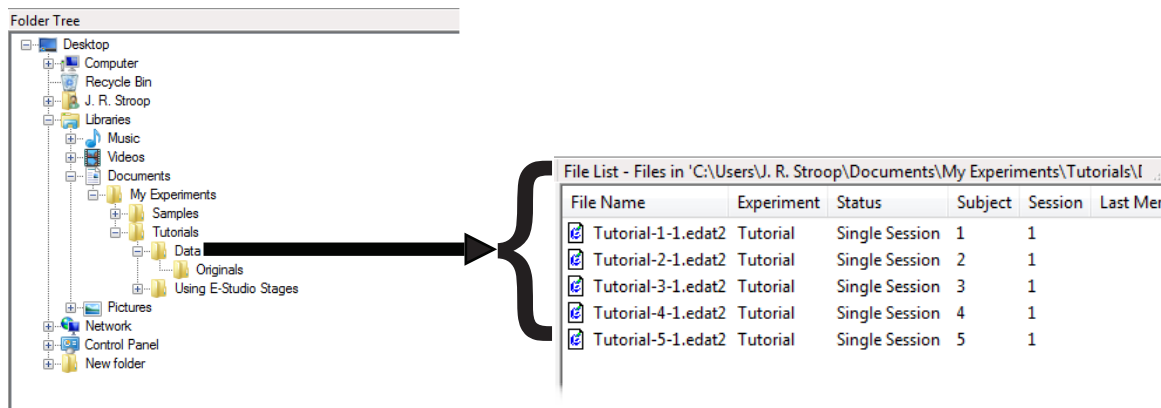
Select Files

Once the data files within a folder are displayed, select the specific data files to merge. To select all data files that have never been merged, click the Select Unmerged tool button in the toolbar.

To select all files, use the Select All command from the Edit menu. The Select All command selects all E-Prime 2.0 data files (.edat2 and .emrg2) regardless of their merge status (i.e., never merged or previously merged).
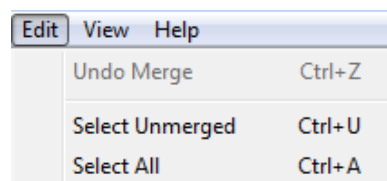
While all files are highlighted, individual files may be deselected by holding down the Ctrl key and clicking the name of the file with the left mouse button. Files in the list may be individually selected by clicking the file name with the left mouse button. A group of non-contiguous files may be selected by holding down the Ctrl key while selecting the files. A consecutive range of files may be selected by clicking the first file in the range and, while holding down the Shift key, clicking the last file in the range.

The display of files in the File List view may be limited using a filter. To display only E-Prime 2.0 .edat2 files in the list, click the Filter button to display the Filter dialog. In the dropdown list, select the "*.edat2" option and click OK.
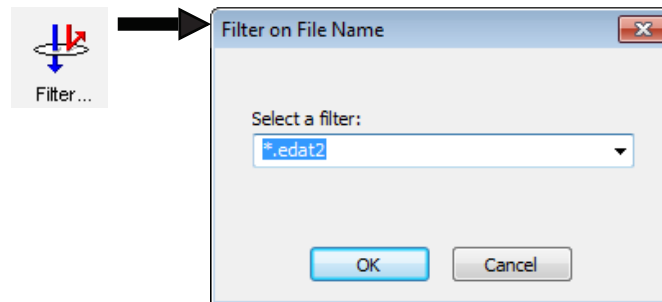
In a similar manner, the list may be filtered for only E-Prime 2.0 .emrg2 files, or for both E-Prime 2.0 .edat2 and .emrg2 files. Filters may also be specified for E-Prime 2.0 version 1 data files (.edat and .emrg). The Filter on File Name dialog permits the user to enter a custom filter as well. Filters may contain the wildcard symbol "*", and multiple filters are supported. When entering multiple filters, separate the filters by a semi-colon (e.g., MyExp*.edat2; MyExp*.emrg2).

During a merge operation, data files are merged into the target file in the order in which they appear in the list. By default, data files appear in the File List in alphabetical order by file name. The list may be resorted according to values within a specific column by clicking on the appropriate column header. The first click arranges the column in ascending order. Clicks, thereafter, alternately sort the column in descending or ascending order. For example, to make sure the data is merged into the target file in participant order, click the header for the Participant column until the participant numbers are in ascending order. (Because the file names are sorted based on string order, a file name with participant 10 in it may appear before a file name with participant 2 in it. If you have both single-digit and double-digit participant numbers, you may want to rename the files prior to merging, adding a leading zero to the subject number part of the data file. For example, you may wish to rename the file "My Experiment–2-1.edat2" to "My Experiment–02–1.edat2).

Click Merge Button
Once the files are selected in the File List, click the Merge button to begin the merge operation. The Select the Merge Operation dialog appears, prompting the user to select the type of merge operation to perform. The default setting is the Standard merge operation. Accept the Standard merge default by clicking the Next button.
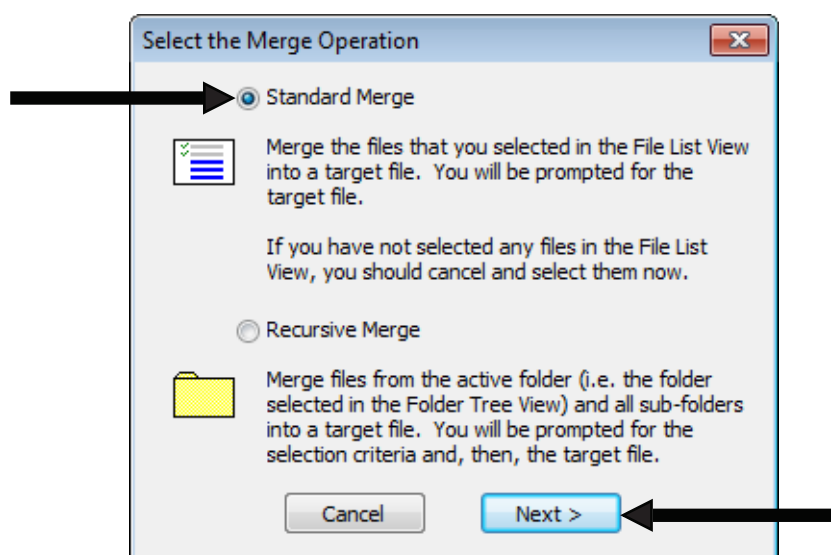
The user is then prompted to identify a target file into which the selected files (i.e., source files) will be merged. In the Set Target File dialog, either navigate to the folder containing an existing target file and then click on the file name, or enter a new name for the target file in the File name field. After identifying a target file, click the OK button to continue.
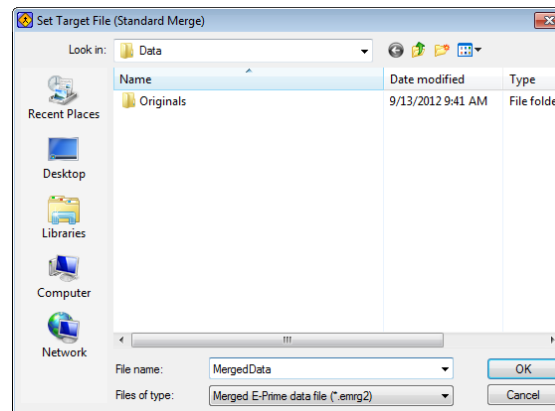
If the named target file does not already exist, the application will ask whether it should be created. Click the Yes button to create the target file and continue.
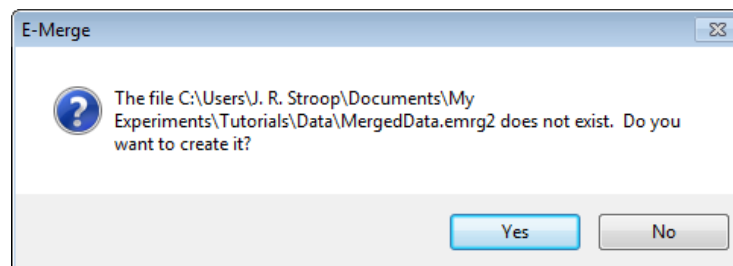
After completing the merge operation, the application will display the Merge Results dialog, summarizing the operation. In addition, the details concerning the success or failure of merging specific files is reflected in the Merge Log view.
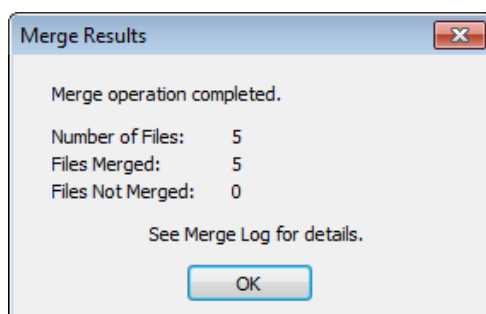
The status of a particular file (e.g., previously merged to a target file, set as the target file, never been merged, etc.) may be observed by viewing a file's icon in the File List view.
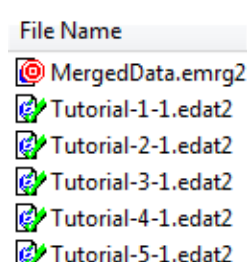
As illustrated in the image above, after merging the Tutorial experiment data files into the MergedData. emrg2 file, each file's icon receives a green checkmark, indicating that it has been merged. The icon for the file currently set as the target file is marked by a target symbol.

Recursive

To merge data files in E-Merge using the Recursive Merge operation, open the parent data folder in the Folder Tree and click the Merge button to merge. Unlike the Standard merge operation in which the data files are selected before clicking the Merge button, the Recursive Merge operation prompts the user for the file selection criteria. Note, the images used throughout this example reflect operations upon data files included with the E-Prime 2.0 installation. With the default installation, these files exist in the …\My Experiments\Tutorials\Data folder.

Open Folder

In the Folder Tree, select the parent data folder (i.e., the folder that contains the subfolders with data files). Data files will appear in the File List view depending on whether the parent folder contains data files or whether the data files exist only in the subfolders. In this example, the Data folder is the parent folder containing both data files and the Originals subfolder. Additional data files are contained within the Originals folder. Actually, the data files within the Originals folder are exact copies of the files in the Data folder (i.e., they are backup files for working with the *E-Prime Getting Started Guide*). However, these files will sufficiently illustrate the Recursive Merge procedure.
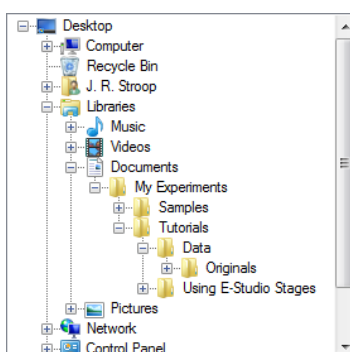
Click Merge Button

Once the parent data folder (i.e., Data) is selected in the Folder Tree, click the Merge button to begin the merge operation. In the Select the Merge Operation dialog, select the "Recursive Merge" option and click the Next button.
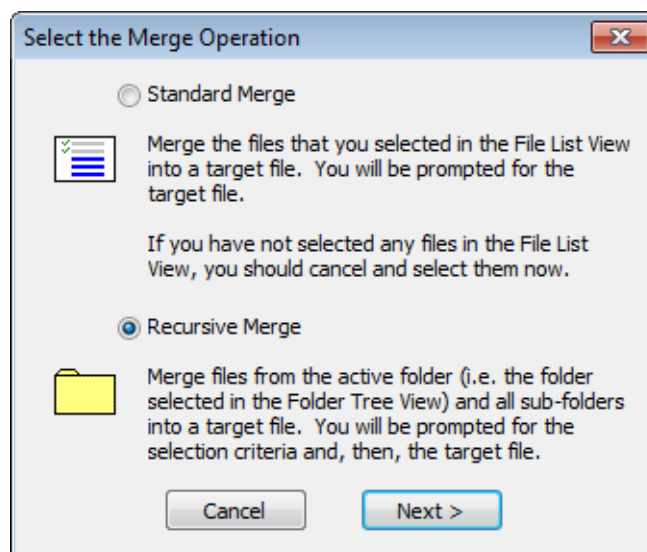
Select Files

The Enter Selection Criteria dialog is displayed in order to specify the file selection criteria. In the Recursive Merge operation, files from the parent folder and all its subfolders fitting the criteria are selected to be included in the merge operation. The criteria include a file specification (.edat2 files, .emrg2 files, or all E-Prime 2.0 data files) and a merge status (never merged, already merged, or ignore merge status). Accept the default settings, and click the Next button to include all .edat2 files that have not previously been merged.
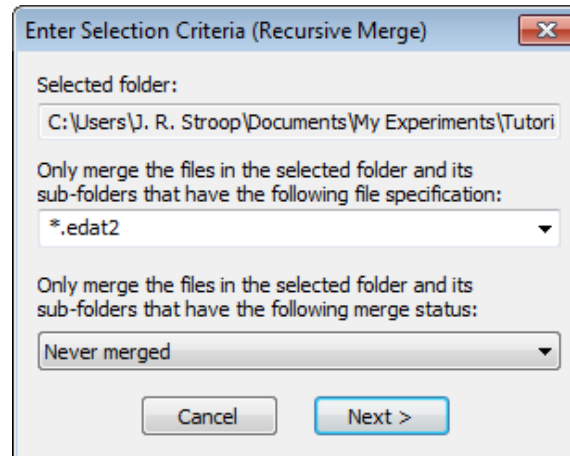
The merge operation will then continue as it does in the Standard merge (i.e., by prompting for the target file). Navigate to an existing file, or enter the name of a file to be used as the target file for the Recursive Merge operation. Click the OK button to continue.
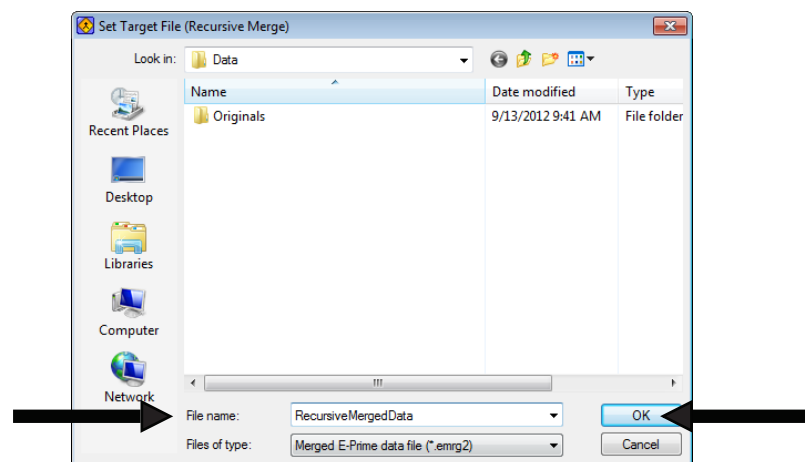
If the named target file does not exist, the application will verify whether or not it should be created. Click the Yes button to continue.
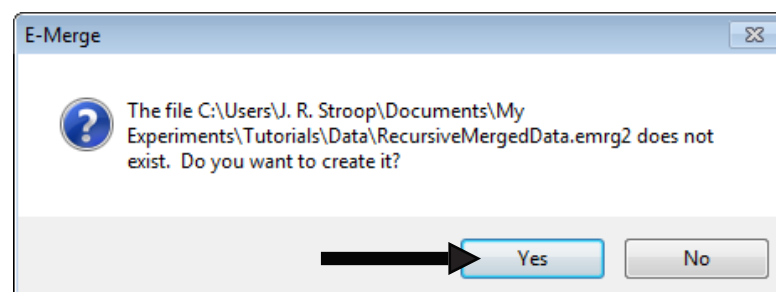
The target file will be created as specified in the Set Target File dialog, and the merge operation will commence. Upon completion of the operation, the Merge Results dialog will display a summary of the merge operation. The Recursive Merge operation just performed will merge five .edat2 files from the Data folder and five .edat2 files from the Originals subfolder into the RecursiveMergeData.emrg2 file.
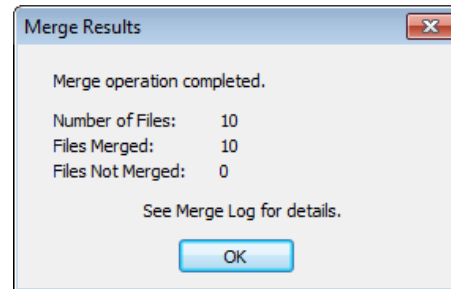
As with the standard merge operation, the Merge Log and the File List contain feedback about the merge operation. Selecting the Data folder and the Originals folder independently, in the Folder Tree, will reveal the individual data files that have been merged during the operation. Notice that the data files from the Originals folder were actually merged into a target file in a different folder location (i.e., Data folder).
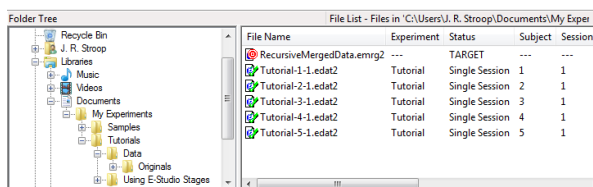
Figure 3. Merged data in the parent folder (i.e., Data).
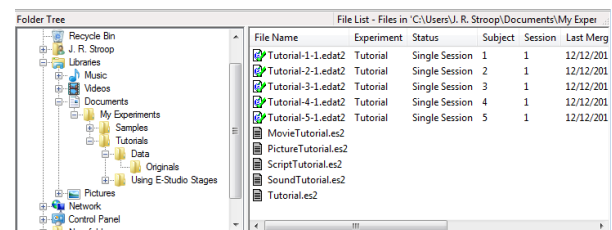
Figure 4. Merged data in the sub-folder (i.e., Originals).

As part of the E-Prime 2.0 installation, the data files included in the Originals folder are actually identical copies of those included in the Data folder. During the merge operation, E-Merge identifies the overlap and sends warning messages to the Merge Log window.
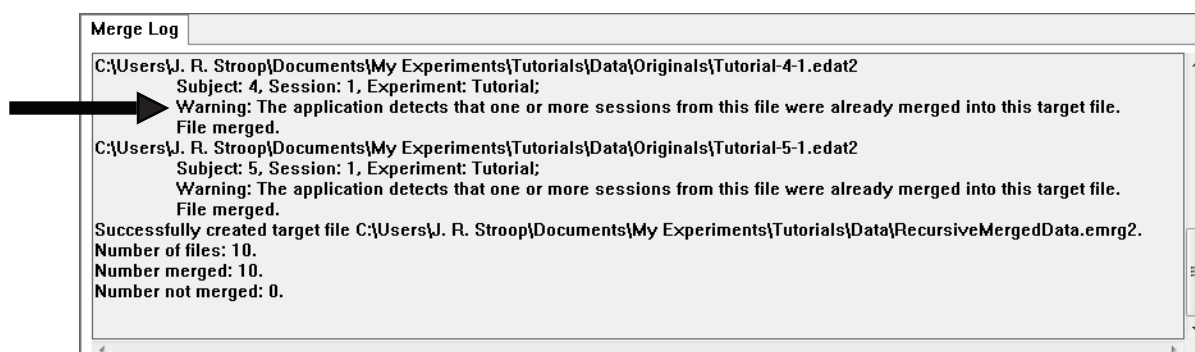
Although the "Number of files" report in the Merge Log accurately describes the operation as involving 10 files, the RecursiveMergeData.emrg2 file actually only contains 5 files. When the identical files from the Originals folder were merged into the target file, E-Merge detected that they were duplicates of the files just merged from the Data folder, and simply overwrote the files, see *Conflicts (Page 165)* for more information.

Undo Merge
During any single E-Merge session, E-Merge permits the most recent merge operation to be reversed by clicking the Undo Merge button, or by selecting the Undo Merge command from the Edit menu. E-Merge supports only one level of undo. Therefore, once another merge operation is performed, or the application is closed, the previous merge operation may not be reversed.
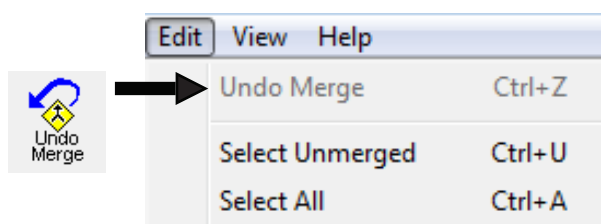
When a merge operation is undone, the files and icons return to the status maintained prior to the merge operation. For example, if an unmerged file is included in a merge operation, the file would then be classified as "already merged", and the icon for that file would receive a green checkmark. If the merge operation is subsequently undone, the file status would return to "unmerged", and the green checkmark would be removed from the icon. In addition, the "Merge operation undone" message would be displayed in the Merge Log.
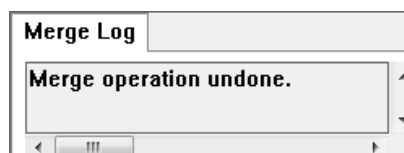
Conflicts
E-Prime 2.0 data files were designed to be as flexible as possible. For example, a merged data file can contain data from different experiments, and those experiments can vary greatly in structure. Unfortunately, in allowing this flexibility, the possibility of merging two incompatible data files becomes a reality and must be addressed. Other issues of concern include merging duplicate data into the target file and handling file errors (i.e., a file cannot be opened or read) during the merge process.
The following section addresses these issues.
Variable/Log-Level Names and Data Types

Two basic rules apply to all data files:
1) The same name may not be used for both a variable and a log-level, or for two different log-levels within a single file.

2) A single variable cannot have more than one data type in the same file.

If one of these two rules is violated, the two files (i.e., the source file and the target file) are considered incompatible, and E-Merge cannot merge them. Fortunately, these incompatibilities should rarely occur.

A violation of the first rule would only occur when merging experiments having different naming schemes for log-levels, under the condition that E-Merge could not resolve the differences without creating a conflict. For example, a data file with log-level 2 named "Sections" and log-level 3 named "Questions," is being merged into a target file with log-level 2 named "Blocks" and log-level 3 named "Trials." During the merge operation, E-Merge would try to merge the data by renaming "Sections" to "Blocks" and "Questions" to "Trials" in the target file. However, if the data file being merged into the target file happened to also contain a variable called "Trials", this would result in a target file that had both a variable and a log-level with the same name. Or, if the source file had a different log-level named "Trial," this would result in a target file that had two log-levels with the same name. E-Merge cannot resolve these conflicts, and therefore cannot merge these files.
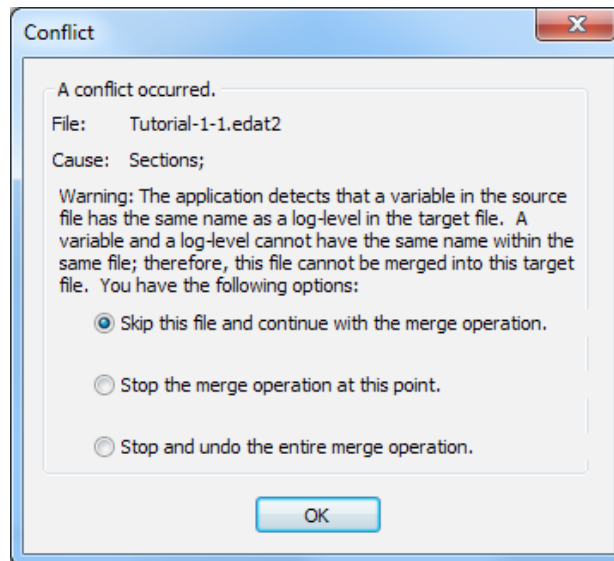
A violation of the second rule is even rarer because E-Run logs all variables, with the exception of participant and session number, as string data. This situation would occur only if a variable of an integer or float data type was created in the data file using E-DataAid and then the user attempted to merge the data file into a target file that already had a string variable with the same name.

Before merging a data file into a target file, E-Merge checks for these incompatibilities. In the event that an incompatibility is discovered, E-Merge alerts the user as to the conflict and offers the opportunity to take one of the following three actions:

1) Skip the file and continue with the merge operation.
2) Stop the merge operation at the point of the conflict.
3) Stop and undo the merge operation altogether.

The variable or log-level causing the conflict must be renamed, to merge the source file into the target file successfully, before attempting the merge operation again.

Duplicate Data Versus Different Runs
Another potential conflict involves duplicate data in a merged data file (i.e., merging the same data more than once into the same file). When attempting to merge the same data more than once, E-Merge identifies the duplicate data and attempts to resolve the conflict. E-Merge will resolve this type of conflict by overwriting the data. Therefore, the operation does not prompt the user to specify a course of action. However, after the merge operation, the Merge Log displays messages indicating that the data files were overwritten.

In contrast to duplicate data is the case in which the user attempts to merge two different runs of the same experiment having the same subject number and session number (e.g., two participants were assigned the same participant and session number). Even though the data are different, having two runs with the same experiment name, subject number, and session number in the same data file can cause confusion (e.g., an analysis by participant on this data file would be incorrect). In addition, E-DataAid's annotation capabilities rely on the experiment name, subject number, and session number to reference edits in a file. Having more than one run of data with the same experiment name, subject number, and session number in the same file could make it difficult to follow the audit trail. To some extent, E-Run tries to prevent this type of conflict from occurring by naming each data file using the experiment's name, subject number, and session number. This makes it impossible to end up with two different runs of the same experiment with the same participant and session number in the same data folder, because E-Run warns the experimenter of this situation prior to the run. The experimenter is

then given the option to overwrite the existing data file or abort the run (in order to launch E-Run again and enter different values for participant and/or session).

Of course, if the runs are collected in different subfolders or on different machines, this safety measure fails. For this reason, before merging a data file into a target file, E-Merge checks for duplicate participant and session values across data files. If this condition is discovered, E-Merge alerts the user as to the conflict and presents the opportunity to take one of the following four actions:

1) Skip the file and continue with the merge operation.
2) Merge the file anyway.
3) Stop the merge operation at the point of the conflict.
4) Stop and undo the merge operation altogether.

If the user chooses option #2 (to merge the data containing the duplicate participant and session number), the target file should be edited immediately using E-DataAid in order to avoid potential confusion.

E-Merge determines whether data is duplicate data or a different run based on a unique identifier that E-Run assigns to the data. This identifier is a 128-bit (16-byte) integer generated by the Windows system based on the date, time, and machine identifier, and it is virtually unique. This identifier stays with the data even when merged. E-Merge uses this identifier rather than the experiment name, subject number, and session number to compare the runs in the data file to the runs in the target file. This way, E-Merge can detect the data is different even when the experiment name, subject number, and session number are the same. Conversely, E-Merge can detect duplicate data even when the experiment name, subject number, or session number have been edited.

File Error
A conflict message will be generated if E-Merge cannot open or read a data file. In this situation, E-Merge alerts the user as to the conflict and presents the opportunity to take one of the following actions:

1) Skip the file and continue with the merge operation.
2) Stop the merge operation at the point of the conflict.
3) Stop and undo the merge operation altogether.

To resolve this conflict, determine whether the data file is already open in another application, such as E-DataAid. If the data file is open in another application, E-Merge cannot access it. If the data file is not already open, test the data file by attempting to open it in E-DataAid. If the data file cannot be opened in E-DataAid, it may be corrupted. In this case, the data file may possibly be recreated by the E-Recovery application using the text file generated by E-Run prior to conversion to an E-Prime 2.0 .edat2 data file. Refer to the *E-Prime 2.0 New Features/Reference Guide* (Chapter 6: E-Recovery) for more information concerning the E-Recovery application. By default, the E-Run text file is retained in the folder in which the data file was created. However, the text file may not be available if the user deleted it or enabled the option to delete the text file after successful conversion to .edat2 format.

Review Merge History

For single .edat2 files, the file names convey information about the contents of the data files, as do the Experiment, Participant, and Session columns in the File List view. However, as illustrated in the image below, the contents of an .emrg2 file cannot be determined simply by looking at the name of the file or the File List view.
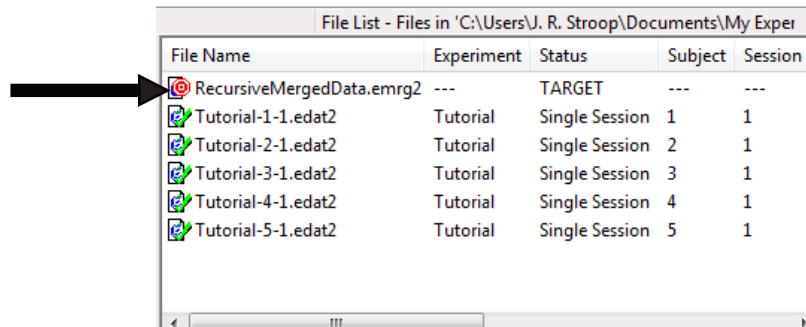
One way to determine the contents of an .emrg2 file is to open it in E-DataAid. To open a file in E-DataAid while in E-Merge, right-click on the filename and select the Open command. Alternatively, the Properties command may be used to obtain information about a file without actually opening the file. This command displays four tabbed-pages containing summary information about the selected file.
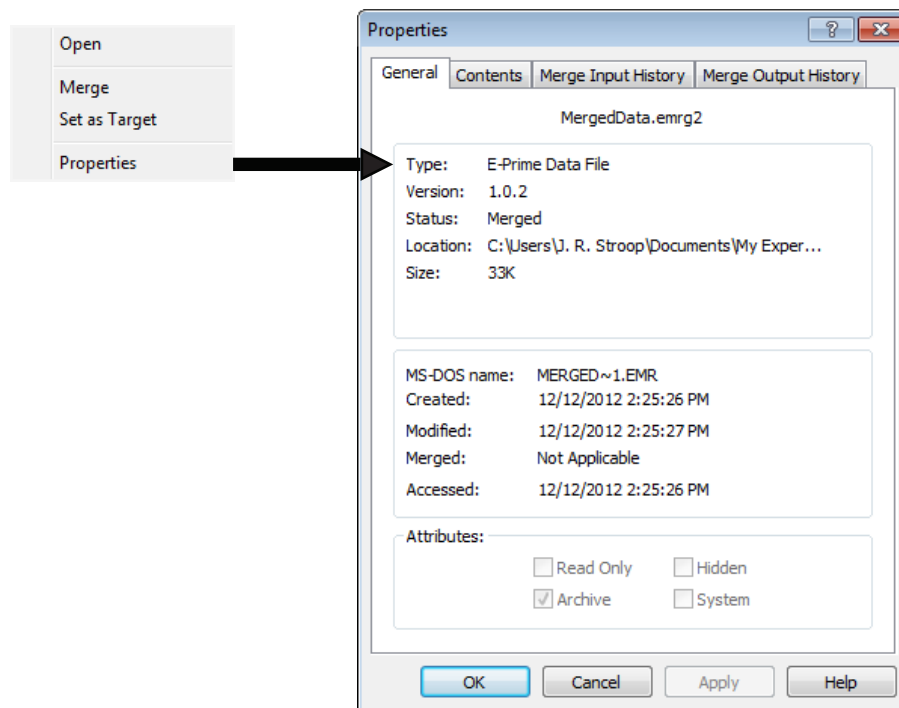
To view the contents of the file quickly, select the Contents tab. This page lists the contents of the data file by subject number, session number, and experiment name. The list may be sorted by clicking any column header (e.g., to sort by Participant number, click the Participant column header).
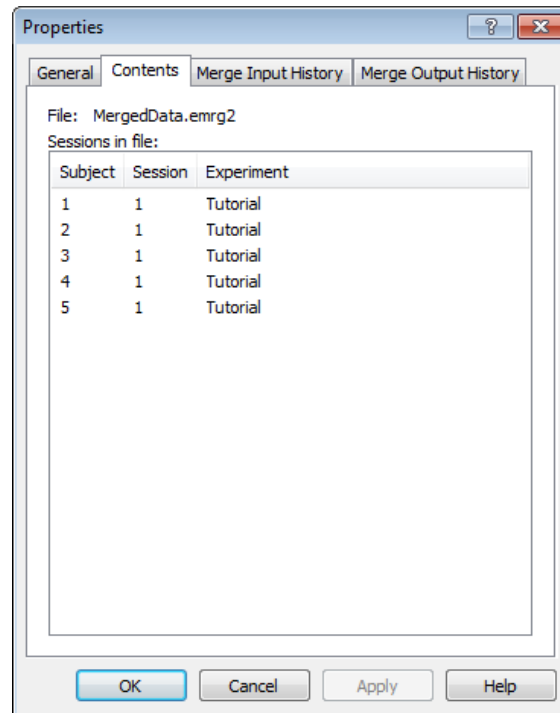
To view the files that have been merged into the file, select the Merge Input History tab (Figure 5), listing all of the files merged into the selected file in the order in which they were merged. To see the entire file name, simply rest the cursor on the name. To view the sessions that a file contained when it was merged, select the file in the list and click the Details button. To see if the selected file was ever merged into any other files, click on the Merge Output History tab (Figure 6). This page lists all files into which this particular file was merged.
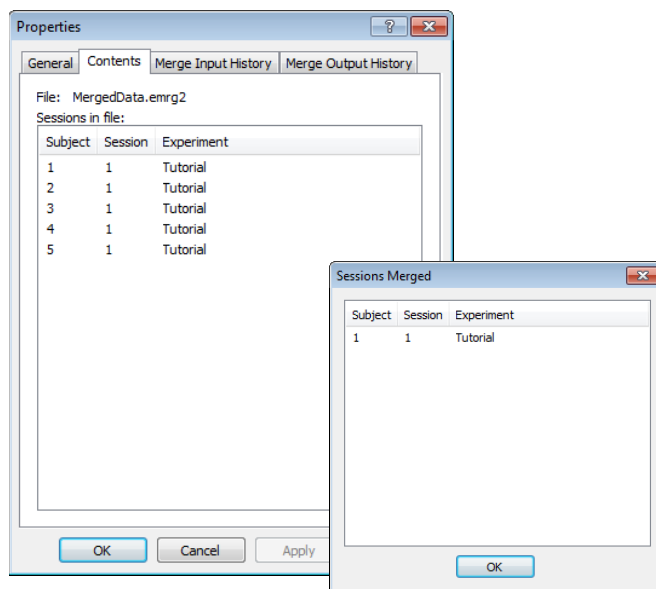
Figure 5. Merge Input History and details show sessions contained within a merged data file.
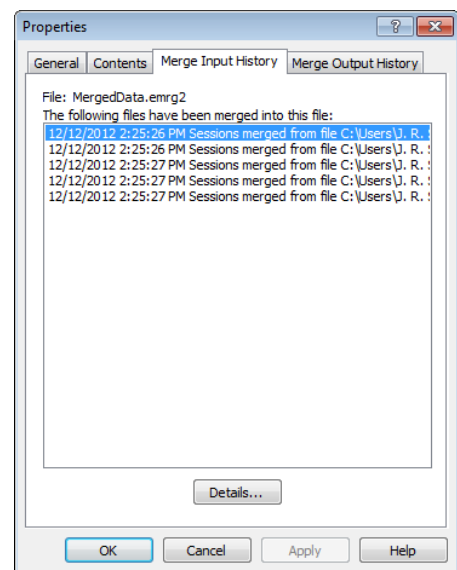
Figure 6. Merge Output History tab indicates the files into which the session has been merged.

6.3 Stage 3: Data Handling Using E-DataAid

Stage 3, Step 1: Introduction
E-DataAid is E-Prime 2.0's data editing and analysis application. Single E-Prime 2.0 .edat2 and/or edat files generated by E-Run, or merged E-Prime 2.0 .emrg2 and/or emrg files generated by E-Merge may be opened for viewing, editing, and analysis with E-DataAid. When the E-Prime 2.0 data file is opened, E-DataAid displays the data in a spreadsheet format with levels and variables in column headings across the top and trials of data in the rows of the spreadsheet.

E-DataAid provides many of the common commands (e.g., find, replace, fill down) and functionality (e.g., moving, hiding, and unhiding columns) of typical spreadsheet applications. However, this is where the similarity to typical spreadsheet applications ends. E-DataAid was designed with four goals in mind:

1) Allow quick simplification of the spreadsheet, thus allowing the user to get at the "heart" of the data
2) Allow editing of the data while simultaneously preserving data integrity
3) Obtain quick descriptive statistics
4) Allow the export of the data to common statistical package formats

This section provides guidance, insight, and tips for using E-DataAid to maximize these four goals. Work through the *E-Prime 2.0 Getting Started G*uide (Tutorial 3, E-DataAid) and have a general knowledge of working in a Windows application (preferably a spreadsheet application, such as Excel) before reading this section.

Stage 3, Step 2: Reduce the Data Set
When E-DataAid is first opened, the number of variables logged by the E-Run application can seem overwhelming. In addition to logging the attributes and dependent measures specified at each level of the experiment, E-Run logs many other variables for each List and Procedure object used in the experiment. This is by design, and is an important feature for verifying the correctness of the experiment's structure. However, during the data-handling phase, these "extra" variables may lose their usefulness, and even hamper the user's ability to view the data. Therefore, whether the goal is to view, edit, analyze, and/or export the data, reduction of the visible data set is recommended.

E-DataAid has a number of special, built-in features that allow the user to easily simplify the data (i.e., reduce the visible data set). This section discusses three ways to reduce the data set: 1) Collapsing levels, 2) Arranging columns, and 3) Filtering rows. It also illustrates how to restore the spreadsheet to its "original", or default format.

Collapse Levels
E-Prime 2.0 experiment data has a hierarchical format. That is, experiments start with a top level of data, which is the session level. From there, experiments branch off into lower levels, such as block, trial, sub-trial, etc. The lower levels share data collected at the parent level. For example, in the image below, the Trial level shares the Block number and PracticeMode data from the Block level. Likewise, the Block level shares data from the Session level (e.g., Participant and Session information).

	ExperimentName	Subject	Session	Date	Group	RandomSeed	Time	Block	PracticeMode	Trial	CorrectAnswer	NameGender	Prime
1	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	1	1	male	sports
2	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	2	2	female	sports
3	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	3	1	male	laundr
4	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	4	1	male	bald
5	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	5	1	male	flower
6	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	6	2	female	flower
7	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	7	2	female	bald
8	Tutorial	1	1	09-1	1	-747118593	13:37	1	no	8	2	female	laundr

Because the data is in a flat spreadsheet format, it is necessary for the spreadsheet to repeat higher-level data across consecutive rows of the spreadsheet. For example, the value of PracticeMode changes only at the block level. Thus, the value of PracticeMode is repeated for each instance of the lower level (i.e., Trial). The image above illustrates only a single block of data in which the value of PracticeMode was set to "no". Thus, the value of PracticeMode is simply repeated for each instance at the lower Trial level.

When examining the lowest level of data, the repetition of higher level data is not an issue. However, to examine one of the higher levels of the experiment, it would be extremely convenient to view only the unique occurrences for that level, or in other words, "collapse" the lower levels of the spreadsheet. This may be done using E-DataAid's Collapse Levels command on the Tools menu, or clicking on the Collapse Levels tool button. Activating this command displays the Collapse Levels dialog.
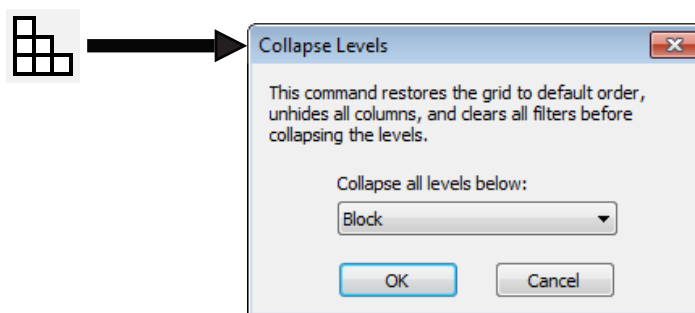
Select the Block level in this dialog and click the OK button to collapse the spreadsheet at the Block level. This will display only the unique occurrences at this level and all higher levels (i.e., Session level). Lower levels are hidden from view. The image below illustrates the MergedData.emrg2 file, obtained from *Standard (Page 159)* in this chapter, collapsed at the Block level.

	ExperimentName	Subject	Session	Group	RandomSeed	SessionDate	SessionTime	Block	BlockList	BlockList.Cycle	BlockList.Sample	Prac
1	Tutorial	1	1	1	-561982034	07-10-2007	11:22:59	1	1	1	1	no
9	Tutorial	2	1	1	-385525238	07-10-2007	11:23:48	1	1	1	1	no
17	Tutorial	3	1	1	-223890453	07-10-2007	11:24:34	1	1	1	1	no
25	Tutorial	4	1	1	-71483576	07-10-2007	11:25:16	1	1	1	1	no
33	Tutorial	5	1	1	85229878	07-10-2007	11:26:00	1	1	1	1	no

The Tutorial experiment ran a single block of trials per participant, and the MergedData.emrg2 file contains data for 5 participants. Thus, the spreadsheet, when collapsed at the block level, displays the unique block level values for each of the 5 participants. Collapsing the spreadsheet according to the lowest data level (in this example, Trial) is equivalent to restoring it to its default format.

Arrange Columns

E-DataAid allows columns to be moved, hidden, and unhidden. Traditionally in a spreadsheet application, a column is moved by first selecting it, then clicking the column header and, while holding down the left mouse button, dragging and dropping it to a new location in the spreadsheet. Similarly, a column is hidden or unhidden by first selecting it and then resizing it, or using the Hide or Unhide commands on the application's View menu. E-DataAid supports all of these methods. However, these methods can be very cumbersome in a large spreadsheet. Therefore, E-DataAid's Arrange Columns command on the Tools menu provides an easier way to move, hide, and unhide columns in the spreadsheet. The Arrange Columns command allows selective viewing of a subset of the output variables in the experiment (e.g., show all dependent measures but hide all other variables). The Arrange Columns command, available from the Tools menu or by clicking the Arrange Columns tool button, displays the Arrange Columns dialog.
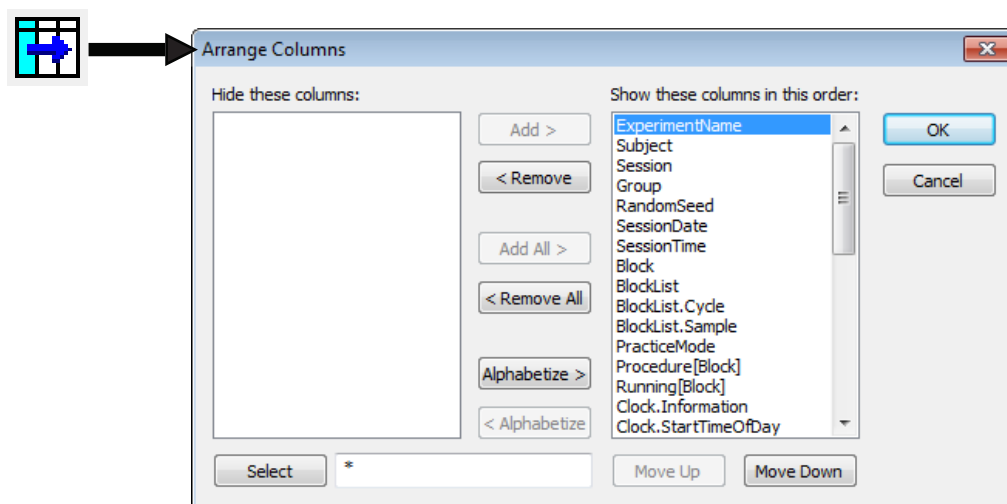
This dialog displays the hidden columns in the list on the left side of the dialog, and shows the displayed columns on the right side of the dialog. By moving columns between the two lists, the view status of the columns can be set. For example, select all columns beginning with "BlockList" in the list on the right, and click the Remove button (Figure 7). The selected columns will be moved to the list on the left (i.e., "Hide these columns"), as in Figure 8.
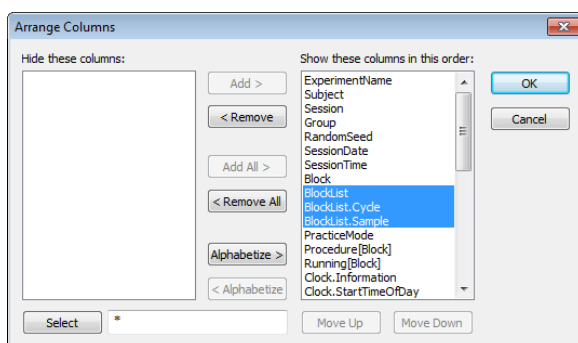
Figure 7. Columns to be removed are selected.
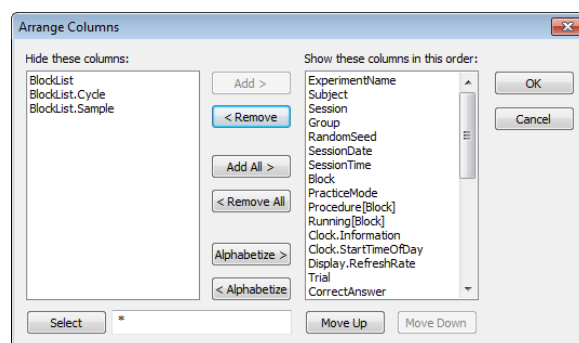
Figure 8. The selected columns are moved to the "Hide these columns" side of the dialog by clicking the Remove button.

When the OK button is clicked, the columns in the left list will be removed from the display. The columns to be displayed may be reordered by using the Move Up or Move Down buttons below the list.

A useful feature for selecting columns within the Arrange Columns dialog is the Select button (bottom left). The field to the right of the Select button is used to enter a filter criterion with which to select column names. Filters can include the "*" wildcard symbol, and multiple filters may be entered, separated by a semicolon. Click the Select button to automatically select, or highlight, these columns in the dialog. For example, it is frequently useful to hide all columns that pertain to the List and Procedure objects. Therefore, a convenient filter to use in the Arrange Columns dialog would be "*List*; Running*; Procedure*." Enter this filter and click the Select button to select all columns matching this filter. Click the Remove button to move the selected columns to the "Hide these columns" list.
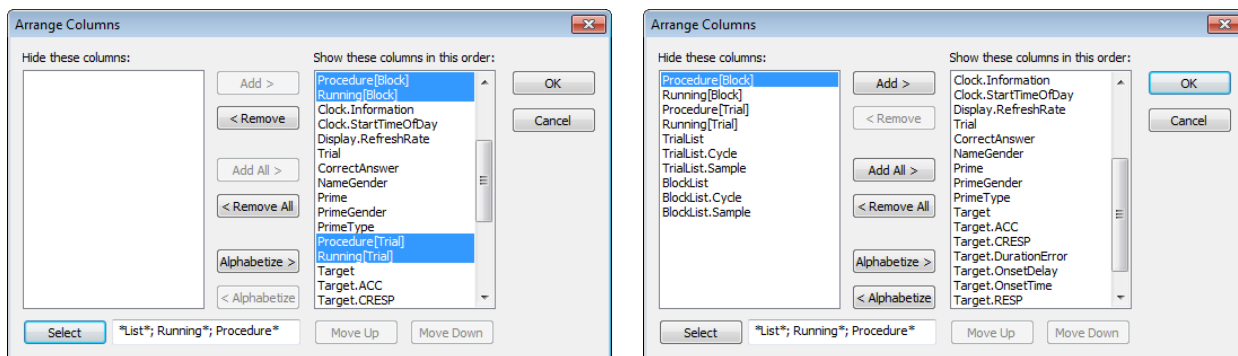
The resulting display is a more manageable and concise spreadsheet, allowing the user to focus on the most important data.

Another useful method of narrowing the display of data in the spreadsheet is to first Remove all of the columns to the "Hide these columns" list, and then to replace (i.e., move back to the Show column) only those columns of interest. For example, to look at only the data related to the input collected by the Target object, click the Remove All button, enter the "Target*" filter, click the Select button to select all variables related to the Target object, and click the Add button. The resulting display will narrow the spreadsheet to only data relevant to the target.

Filter Rows
Most likely, there will be data that the user does not want to include in the spreadsheet. For example, it may not be desirable to view or analyze practice data, incorrect trials, or response times outside of a specific range (e.g., RT < 100). E-DataAid offers the ability to filter the data based on specific criteria using the Filter command. When filtered, the spreadsheet will display only data matching the filter criterion. Data not matching the criterion is hidden from view. E-DataAid does not include hidden rows when analyzing, copying, or exporting data. Filtering may be performed using E-DataAid's Filter command on the Tools menu, or by clicking the Filter tool button. Activating this command displays the Filter dialog.
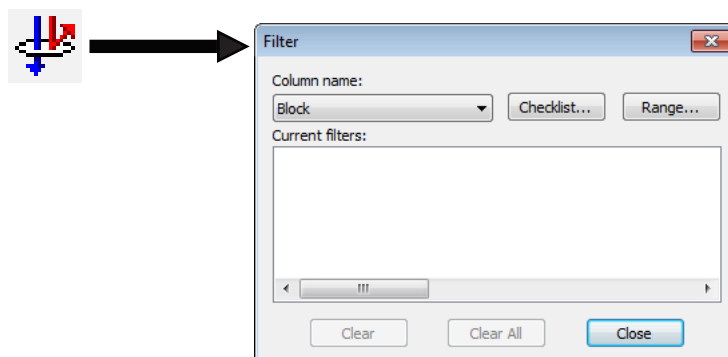
A filter may also be applied to a specific column by selecting a variable in the Column name dropdown list, and using the Checklist or Range buttons to specify the criterion for the filter.

Checklist Filter

The Checklist filter allows values to be individually checked for inclusion in the filter. For example, in order to include only correct responses, use a Checklist filter applied to the accuracy variable (e.g., Target.ACC), and select only the value "1" (i.e., indicating a correct response). The resulting spreadsheet will display only correct responses (i.e., Target.ACC = 1).
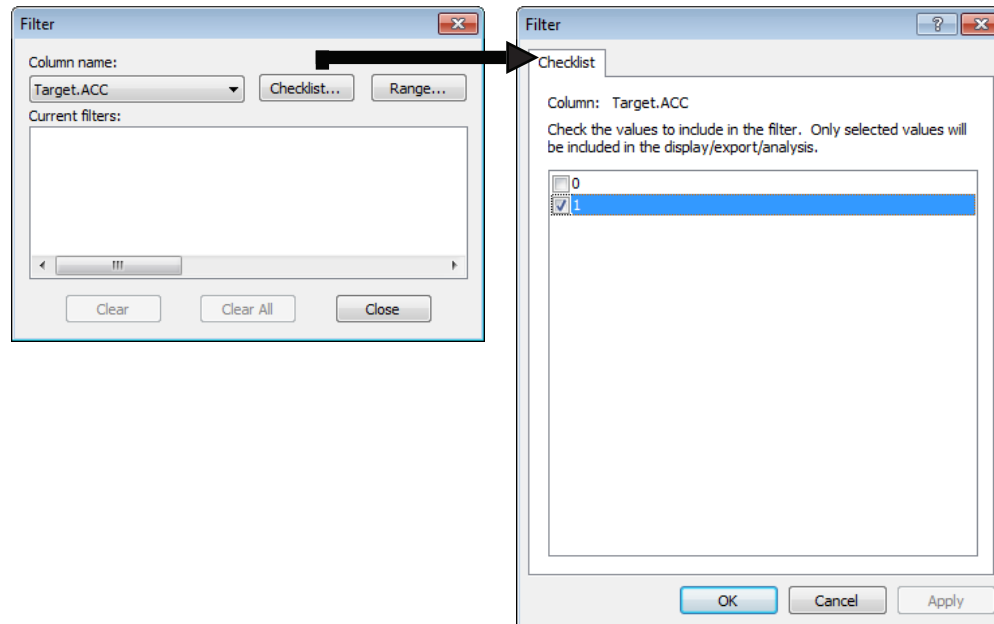
Range Filter

The Range filter permits the user to create expressions that define the range of values to include in the spreadsheet. For example, the user may want to filter the spreadsheet so that only response times between 400 and 1000 ms are displayed. To apply such a filter, select the variable containing the RT data (e.g., Target.RT) in the Column name list and click the Range button. As in the images below, set the First Range to include values greater than 400, change the operator from OR to AND, and set the Second Range to include values less than 1000. The resulting spreadsheet is filtered to display only response times in the specified range.
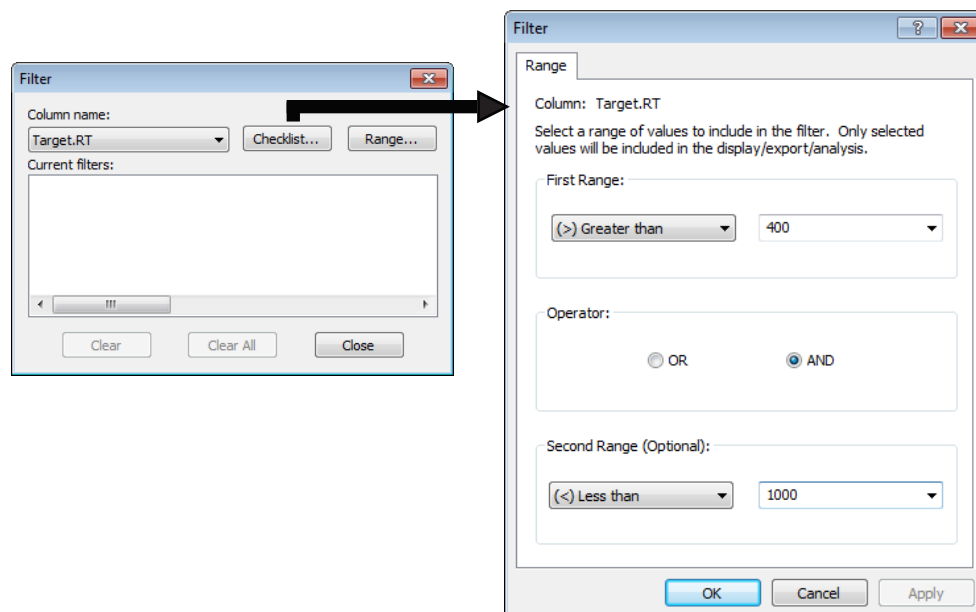
In large files with many rows of data, the Checklist and Range dialogs may take a long time to appear. This is because E-DataAid searches the spreadsheet for the first 32,767 unique values in the column and displays them in the Checklist and Range dialogs. In the case of the Checklist filter, the values are displayed in the checklist. In the case of the Range filter, the values are displayed in dropdown lists next to the fields in which the values are entered for each expression. With the Range filter, an option may be set to prevent the unique values from being displayed. This will greatly speed up the time it takes the application to display the Range dialog. To prevent unique values from being displayed in the Range filter, use E-DataAid's Options command on the View menu. Click on the Filters tab and uncheck the box "Display values in dropdown lists of Range filter."
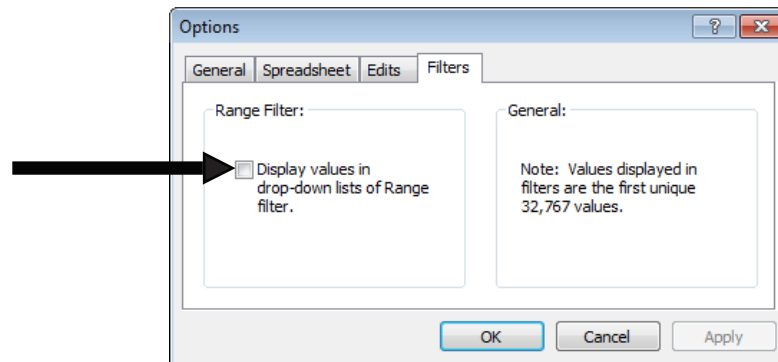
Visual Cues
The spreadsheet provides a number of visual cues to remind the user that filters have been applied to the displayed data. Rows are numbered non-consecutively.

Below the spreadsheet, the Filter Bar, labeled "Filters" displays the current filters. At the bottom of the display, the Status Bar displays the number of unhidden rows.
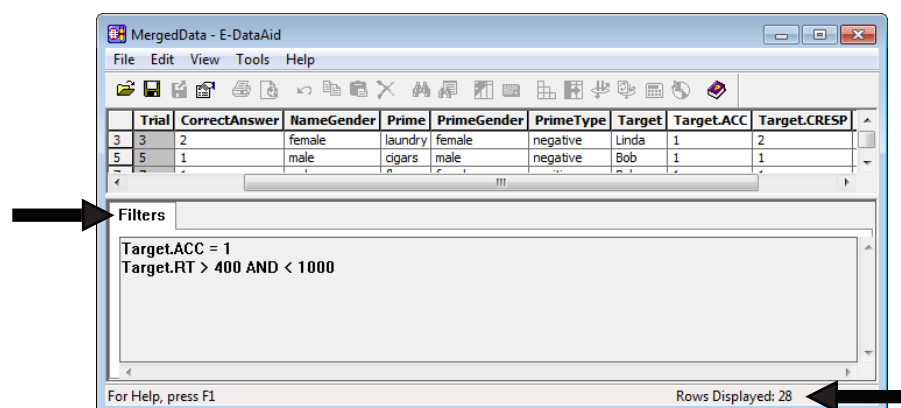
The Rows Displayed value is a quick method of determining that all of the data is present, and/or whether a filter has been applied. When a data file is opened, the Rows Displayed number will always equal the total number of rows in the data file because filters are not saved when the file is closed.

Clear Filters
A filter may be cleared using E-DataAid's Filter command on the Tools menu. Activating this command displays the Filter dialog, which lists the current filters applied to the spreadsheet. To clear a filter, simply select the filter in the list and click the Clear button. To clear all filters, use the Clear All button.
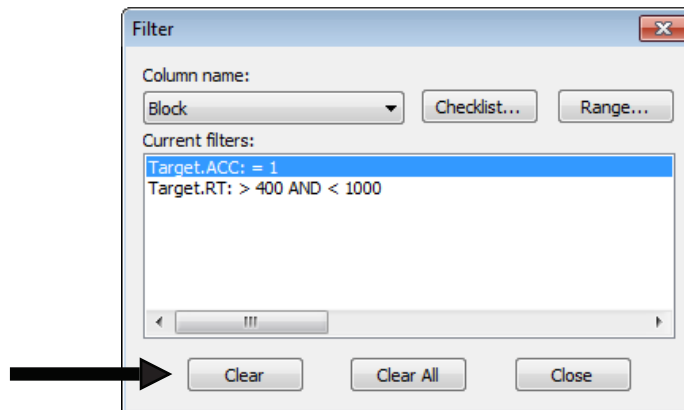
Restore the Spreadsheet
It is not uncommon to need to restore the spreadsheet to its default format (i.e., the data file's format when it is opened for the first time in E-DataAid). When a data file is opened, it will open in the format in which it was saved, with the exception of filters (filters are not saved when the file is closed). The default format is equal to collapsing the spreadsheet at the lowest level (i.e., all data is displayed).

E-DataAid's Restore Spreadsheet command on the View menu restores the spreadsheet to its default format. Activating this command clears all current filters, unhides all columns, and arranges the columns in default order. Default order is as follows:

• System variables (ExperimentName, Participant, Session)
• All session level variables in alphabetical order
• Log-level 2 (e.g., Block) and its variables in alphabetical order
• Log-level 3(e.g., Trial) and its variables in alphabetical order
• Log-level n down to the tenth log-level

Stage 3, Step 3: Understand the Audit Trail
In addition to being a spreadsheet application, E-DataAid also serves as an experimenter's notebook, by marking edits in a different colored text, writing annotations to the file for each modification, and allowing the user to add comments to the data file. With these features, E-DataAid permits the user to keep track of changes that have been made to a data file. Before editing the data, it is important to understand E-DataAid's auditing features.

Edits
To edit a value in a cell, select the cell by clicking in it, and type the new value. When a value is edited, the spreadsheet displays the edited value in red. In addition, when a value that is shared by many cells is changed, as is the case with subject number, all cells sharing that value are automatically updated to the new value when one of the cells is edited.

This feature not only saves time, because additional cells do not have to be updated, but it also preserves the integrity of the data. That is, it prevents inconsistencies in the data file in case the user fails to update all appropriate cells.

After the data file has been saved and closed, the edited data continues to appear in red whenever the data file is re-opened in E-DataAid. If the user chooses to "undo" the edit before closing the file, the data will revert to the original black color. Thus, upon opening a file, it is possible to determine immediately if any data has been edited.

Annotations

For each modification made to a data file, E-DataAid writes an annotation, which is saved with the file. The annotation describes the modification, and serves as a record of changes made to the file. Modifications include such actions as editing a value, renaming a variable, creating a new variable, adding a comment to the file, importing data, or merging data. The file's annotations may be viewed using E-DataAid's Display Annotations command on the View menu, or by clicking the Annotations tool button. Activating this command displays the Annotations dialog.
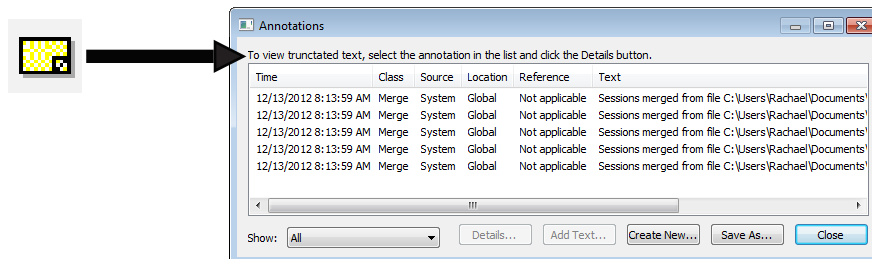
By default, annotations are listed in reverse chronological order. The annotations may be sorted by clicking on any of the column headers in the list. Refer to the *E-Prime 2.0 New Features/Reference Guide* (Chapter 5, Section 5.3.2) for a detailed description of the columns within the Annotations dialog.

The text of the annotation describes the specific modification made to the file. For example, if a session is merged into the file, the annotation includes the name of the file from which it was merged. If a value is edited, the annotation includes both the old and new values, as illustrated in the image below.
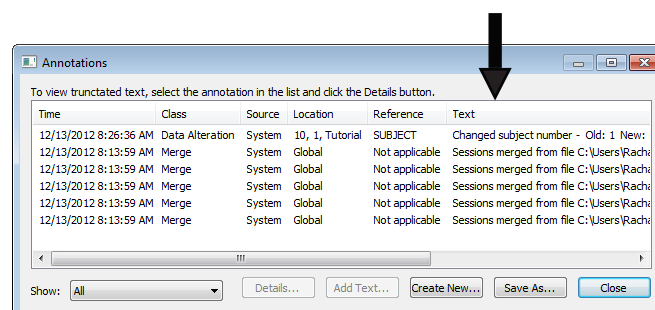
The list of annotations may be filtered to see only one class of annotations at a time by changing the setting in the Show field. For example, selecting "Data Alterations" will display only annotations describing modifications made to the data in the file. When Data Alterations is selected in the Show field, the display above is modified as follows:
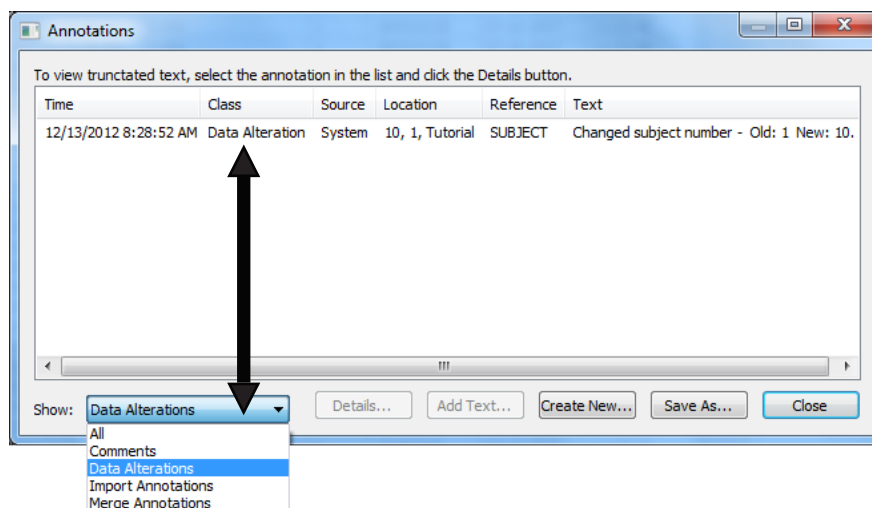
Comments

E-DataAid allows the user to add comments to the data file. For example, it may be beneficial to add comments concerning a specific participant, the procedure, something that may have influenced the session, etc. To add a comment to the file, first use E-DataAid's Display Annotations command on the View menu to display the Annotations dialog. Click the Create New button to display the Annotation Text dialog. Enter the text to be included as the comment and click the OK button. The comment will be added to the file in the form of a user annotation.
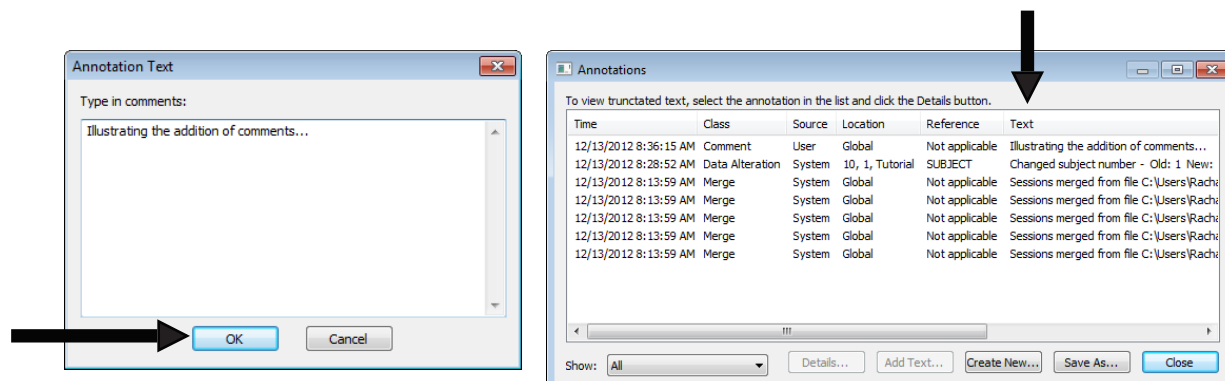
It is also possible to add text to an existing annotation by selecting an annotation in the list and clicking the Add Text button. In the Annotation Text dialog, type in the text to be added and click the OK button. If the annotation text appears truncated in the Annotation dialog, use the Details button to see the complete text.

Edit Data

E-DataAid provides data editing capabilities from editing a single cell to adding a new variable.

Edit Cells

Editing a single cell is as easy as navigating to that cell, entering the new value, and pressing the Enter key. To preserve the hierarchy of the data, E-DataAid does not allow the user to edit log-level values below the session level. Although log level variables may be renamed, the cells are gray to indicate that they are read-only. If any other cells are displayed in gray, this indicates that the experiment Administrator has placed security restrictions on the file and has set the cells as read-only.

Like other popular spreadsheet applications, E-DataAid provides common commands, such as Find, Replace, Fill Down, Copy, and Paste, to make the chore of editing many cells easier. Since these commands are not unique to E-DataAid, time is not spent describing them here. However, these commands are documented in the *E-Prime 2.0 New Features/Reference Guide*.

All edits to a cell appear in red, and for each edit, E-DataAid writes an annotation to the file. Even if done within a group operation, such as Replace All, E-DataAid will write an individual annotation to the file for each edited cell within the operation. For example, if a Replace All operation edits three session-level cells, E-DataAid will write three annotations to the file – one for each edit. (Notice that an edit that is made at a higher level is propagated down throughout the file. For example, each session-level edit appears in every cell in the worksheet that is associated with that session: when you replace the session number, for instance, every block and trial row within the session shows the edited value.) While this type of record keeping is necessary for a complete history of modifications, the number of annotations can quickly become very large, depending on the number of edits made to a file. To make the list of annotations more readable, refer to *Annotations (Page 177)* for information concerning filtering the annotations list.

Delete Data

In E-DataAid, a cell or group of cells is deleted by using E-DataAid's Delete command on the Edit menu. Cells with deleted data or no data are marked as NULL.

Target.RESP
NULL
NULL
NULL
NULL
1
1
1

E-DataAid requires the use of the Delete command to delete data. When data is deleted, the edits appear in red, and E-DataAid writes an annotation to the file for each deletion. This is to distinguish deleted data from the empty string (i.e., it is not uncommon for an experiment to use the empty string as one of its exemplars).

By design, E-DataAid does not allow the deletion of rows or columns of data. To remove columns or rows from the display, columns may be hidden and rows may be filtered, refer to *Stage 3, Step 2: Reduce the Data Set (Page 170)* for more information. If reduction of the data set is not adequate, the spreadsheet can be exported as an E-Prime 2.0 text file using E-DataAid's Export command. Once exported, the text file can be modified in another application and re-imported using E-DataAid's Import command. Refer to the *E-Prime 2.0 New Features/Reference Guide* (Chapter 5, E-DataAid) for information concerning importing files into E-Prime 2.0.

Rename a Variable

At some point, it may be necessary to change a variable's name in the data file. To rename a variable, select the column containing the variable and use E-DataAid's Rename Variable command on the Edit menu to display the Rename Variable dialog.
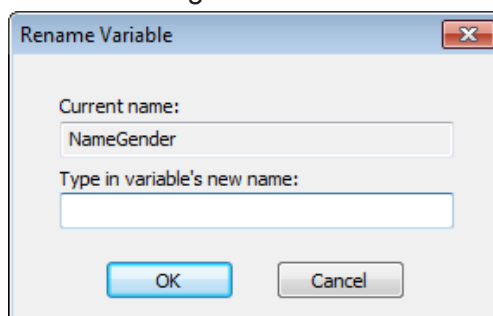
Enter the variable's new name. The new name must begin with a letter, must be composed of only letters, numbers, periods, or underscores, must not exceed 80 characters, and must not already be in use. After entering the new name, click the OK button to rename the variable. With the exception of log-level names, the variable's new name is considered an edit. Thus, the new name appears in red, and E-DataAid writes an annotation to the file describing the renaming.

Add a Variable

It may be necessary to add a variable for additional coding of the data (e.g., post-processing of verbal responses). Adding a variable to an E-DataAid file is akin to adding a column to the spreadsheet. To add a variable, use E-DataAid's Add Variable command on the Edit menu to display the Add Variable dialog. Enter the new variable's name. The new name must begin with a letter, must contain only letters, numbers, periods, or underscores, must not exceed 80 characters, and must not already be in use. Select the variable's level and data type (i.e., integer, float, or string). Click the OK button to add the variable. By default, E-DataAid places the new variable at the end (i.e., as the last column) of the spreadsheet.
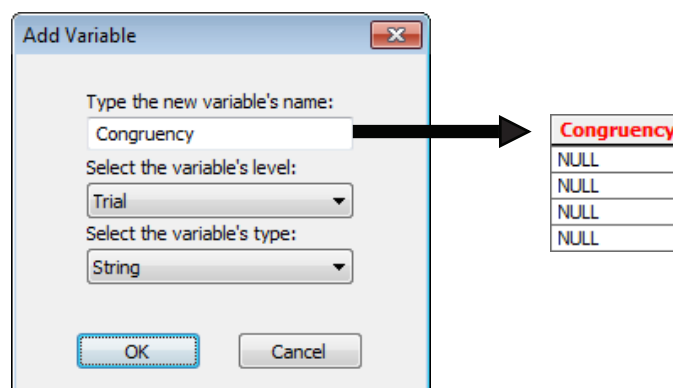
Adding a variable is considered a modification of the data file. Thus, the new variable's name (Congruency) appears in red and E-DataAid writes an annotation to the file in reference to the action. E-DataAid automatically fills the cells in the new column with NULL. To enter values for the cells in the new column, simply click in the first cell and type a value. Depending on the defined level of the new variable, E-DataAid will fill in the values when appropriate, refer to *Stage 3, Step 2: Object.Properties (Page 124)* for a description of the hierarchical nature of experiment data.

Undo Edits

E-DataAid allows the user to "undo," or rollback, any edits made to the file from the time the file is opened until it is closed. To undo the most recent edit, use E-DataAid's Undo command on the Edit menu. Undoing an edit also removes the annotation written to the file by E-DataAid.

E-DataAid's Undo command only reverses actions related to edits. The undo command does not act like a "general undo" to reverse other actions as in other applications, such as hiding or resizing columns. To undo an edit, click the Undo button, or use the Undo command in the Edit menu.
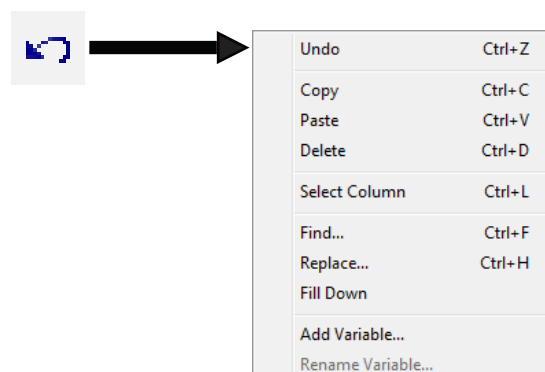
Stage 3, Step 4: Analysis

The Analyze command within E-DataAid allows the creation of tables and graphs of the data. To create an analysis, use E-DataAid's Analyze command on the Tools menu to display the Analyze dialog, or click on the Analyze tool button. Activating this command displays the Analyze dialog.
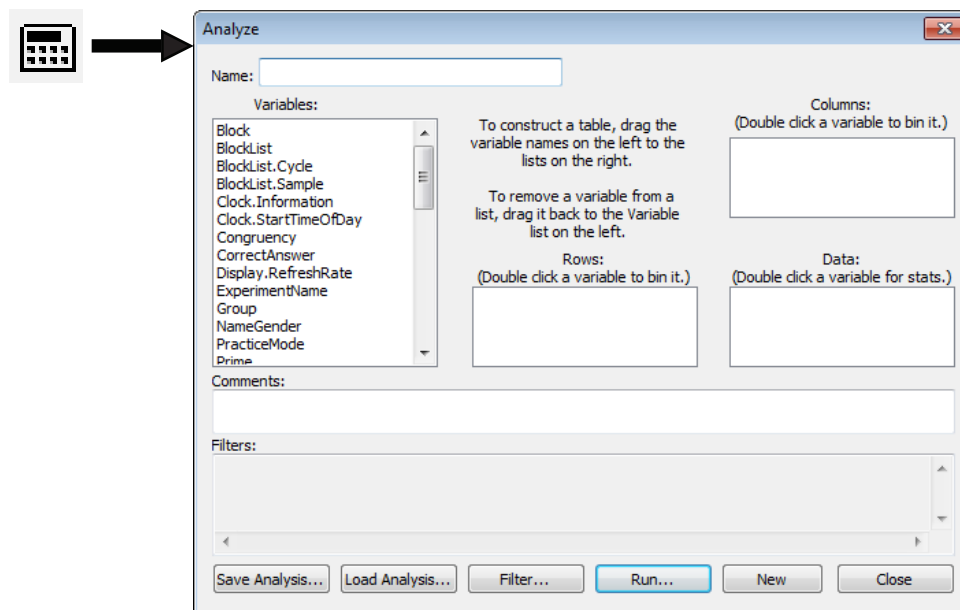
Variables are added to the analysis by dragging and dropping them from the Variables list to the appropriate lists on the dialog (i.e., Rows, Columns, and Data). Variables placed in the Rows list will have a row orientation in the table (i.e., their values will be listed down the rows of the table). Variables placed in the Columns list will have a column orientation in the table (i.e., their values will be listed across the columns of the table). Variables placed in the Data list define the data with which the table will be created (Figure 10). For example, in order to analyze the mean reaction time for male versus female names by participant in the Tutorial experiment, drag NameGender to the Columns list, drag Target.RT to the Data list, and drag Participant to the Rows list (Figure 9). Once defined, click the Run button to perform the analysis.
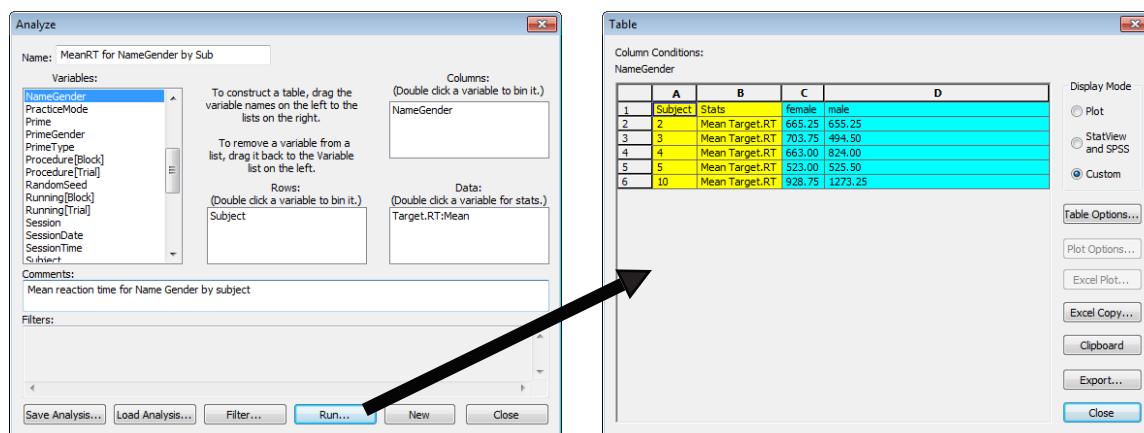
Figure 9. Analyze dialog allows selection of variables for analysis.

Figure 10. Table created from analysis specifications.

The settings for the current analysis may be cleared by clicking the New button on the Analyze dialog. Alternatively, simply remove the variables from a particular list location by clicking a variable and dragging it back to the Variables list. The Analyze dialog will also allow the user to rearrange variables, or move them from one list to another simply by clicking and dragging.

Change a Statistic
By default, when a variable is dragged from the Variables list to the Data list, the statistic analyzed for that variable is the mean. E-DataAid permits the user to specify 15 different statistics:

| Statistic	Definition
Count	Number of times that a value occurs for the variable (numeric or string, provided that the string is not the string for missing data).
CountNull	Number of times that the value for the variable is equal to the string for missing data.
CountNum	Number of times that the value for the variable is numeric.
Max	Maximum value for the variable.
Mean	Mean value for the variable.
Median	Median value for the variable.
Min	Minimum value for the variable.
StdDevP	Population standard deviation for the variable.
StdDevS	Sample standard deviation for the variable.
StdErrP	Population standard error for the variable.
StdErrS	Sample standard error for the variable.
SumSqrs	Sum of squares for the variable.
Total	Total sum of all values of the variable.
VarP	Population variance for the variable.
VarS	Sample variance for the variable.

Change the statistic for a variable by double clicking the variable in the Data list to display the Statistics dialog. In the Statistics dialog, choose a statistic from the dropdown list and click the OK button. When a statistic is chosen, the variable displays the statistic after its name in the Data field of the Analyze dialog.

To calculate more than one statistic for the same variable, add that variable to the Data list more than once. For example, to calculate the count, mean, and standard deviation for a variable, drag and drop the variable from the Variables list to the Data list three times. Change one occurrence to Count and one to StdDevP (one occurrence will already be Mean by default).
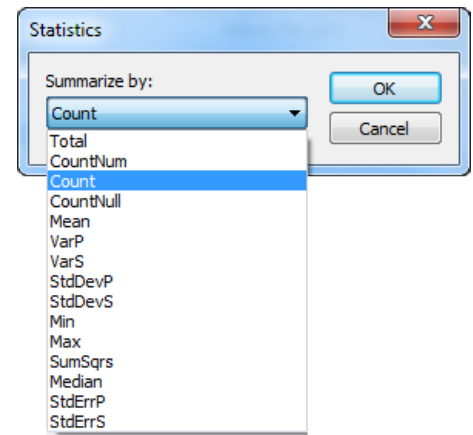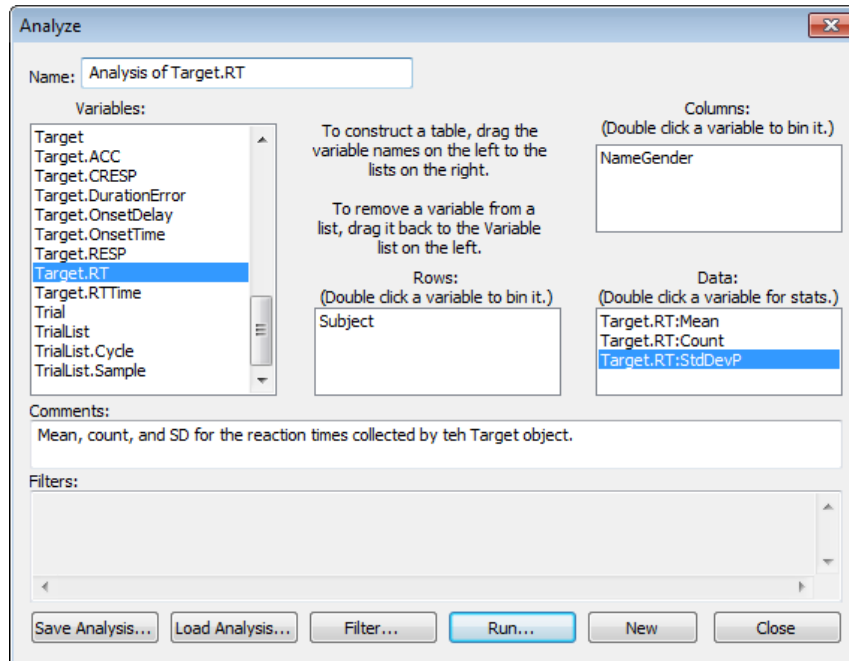
Bin a Variable

It is possible to group or, bin, a variable in an analysis. For example, when creating a distribution of response times, it may be beneficial to bin reaction times by groups of 100 ms. Any numeric variable in either the Rows or Columns list of the Analyze dialog may be binned by double clicking the variable to display the Bin dialog. In the Bin dialog, enter an integer from 1 to 32,767 and click the OK button. Binning the variable by 1 is equivalent to not binning it at all. A variable binned by more than 1 will display its bin number after its name.
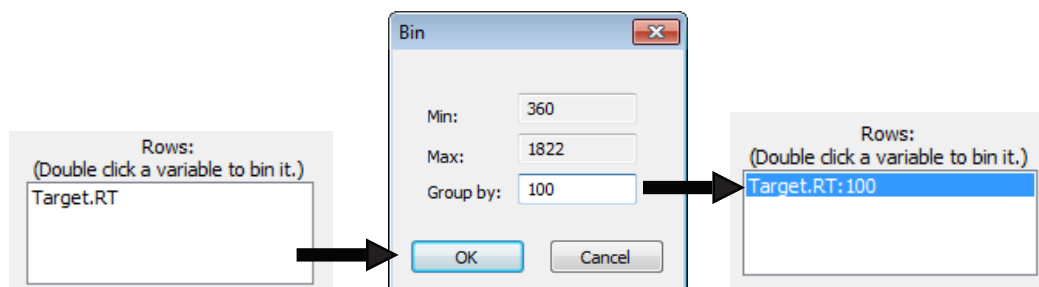
The resulting table will contain bins of data of the specified size for that variable.
For example, to count the number of correct responses occurring during the Tutorial experiment at various levels of the range of reaction times, do the following:

• Place the Target.RT variable in the Rows list
• Place the Target.ACC variable in the Data list
• Set a filter to include only correct responses (ACC = 1)
• Double click the Target.RT variable to bin the data
• Double click the Target.ACC variable to set the statistic to Count.

The results would be the following (when binning by 100):

The image above displays the count of the correct responses occurring at 100 ms levels within the RT data.

Run the Analysis
Once the analysis has been created, run it by clicking the Run button on the Analyze dialog. E-DataAid displays the results in table format as displayed above. The table may be copied to the clipboard, exported to a text file, plotted in Excel, or copied to Excel or export to StatView/SPSS.

Save an Analysis
Once the parameters for an analysis have been set, the analysis may be saved so that it need not be recreated. To save an analysis, click the Save Analysis button on the Analyze dialog. If the experimenter provided a name for the analysis, the name will appear in the Save As dialog; otherwise, a name must be entered.

Analyses are saved in text file format with the .anl extension. These text files can be read by E-DataAid using the Load Analysis button on the Analyze dialog. When E-DataAid saves an analysis, it saves the configuration for the analysis, the spreadsheet's filters, the table's options, and the plot options if applicable. When reloaded, E-DataAid reapplies these items. Saving an analysis does not save the results of the analysis. To reproduce the results of an analysis, the analysis must be reloaded and rerun.

Load an Analysis

After saving an analysis, it may be loaded for rerunning by clicking the Load Analysis button on the Analyze dialog. The application will prompt the user for the name of the analysis file. In addition to loading the configuration for the analysis, the application reapplies the saved filters, table options, and plot options, if applicable. Once loaded, the analysis may be performed by clicking the Run button on the Analyze dialog.

Batch Analysis

To run multiple pre-saved analyses, use E-DataAid's Batch Analysis command on the Tools menu. Within the Batch Analysis dialog, use the Add Files button to select the individual analyses to include in the operation.
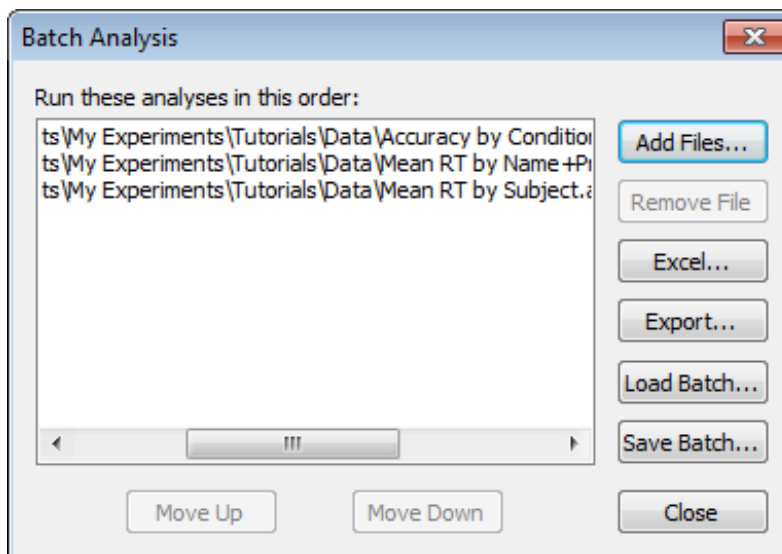
The Batch Analysis command allows the user to run a batch of analyses in rapid succession without individually loading and running each analysis file. After selecting all analyses, click the Excel button to create tables in Excel (shown below), or the Export button to export the tables to a text file. If the Excel option is chosen, the individual analyses are written to separate worksheets within the same workbook.

6.4 Stage 4: Example Analyses

Many different analyses are permitted by the Analyze command. Some examples are provided below. All of the examples use the merged data file (MergedData.emrg2) that was created in *Stage 2, Step 3: Merge (Page 159)*.
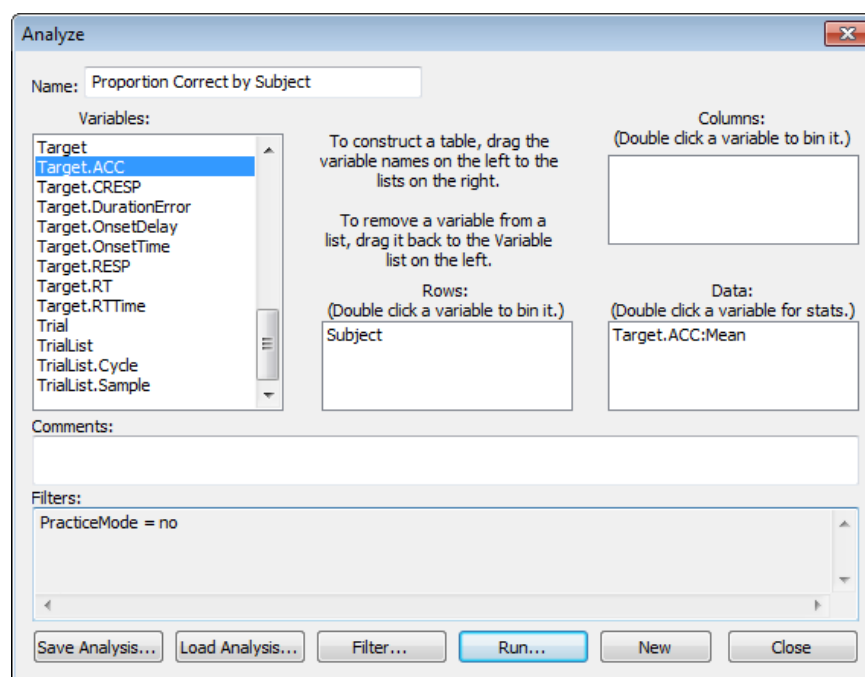
Stage 4, Step 1: Table of Means

The following analysis creates a table of means excluding practice trials and including only correct answers.
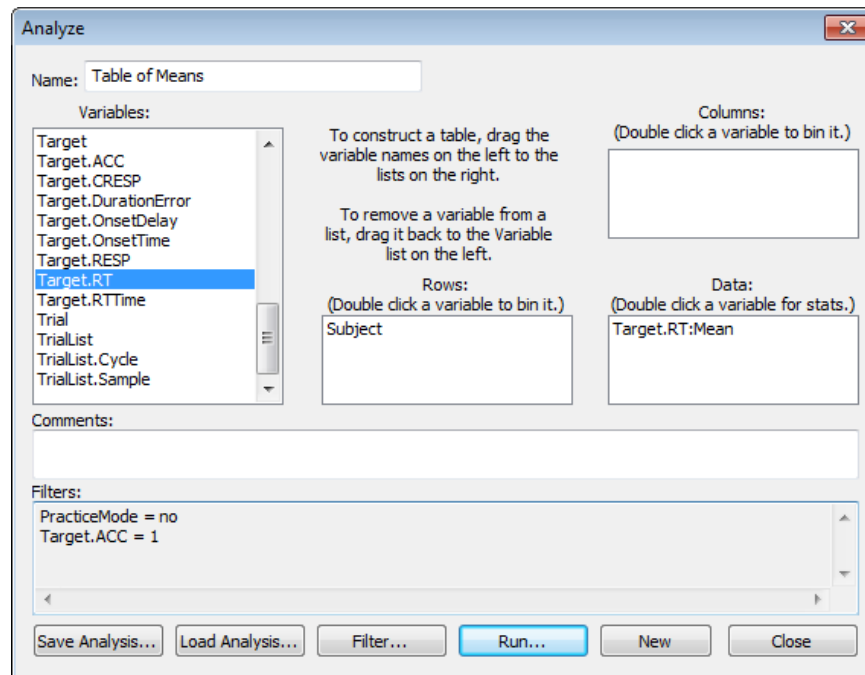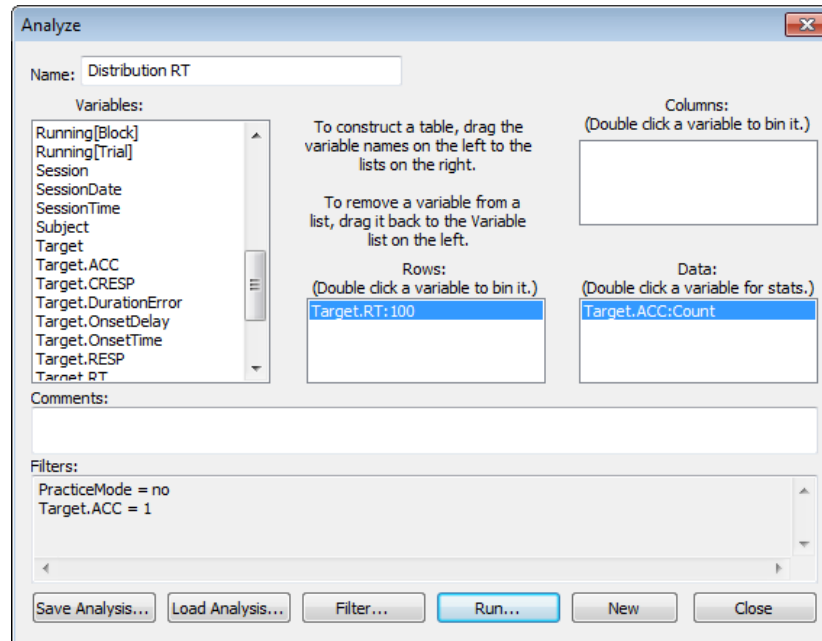

Stage 4, Step 2: Proportion Correct by Participant

The following analysis creates a table of the proportion correct by participant excluding practice trials.
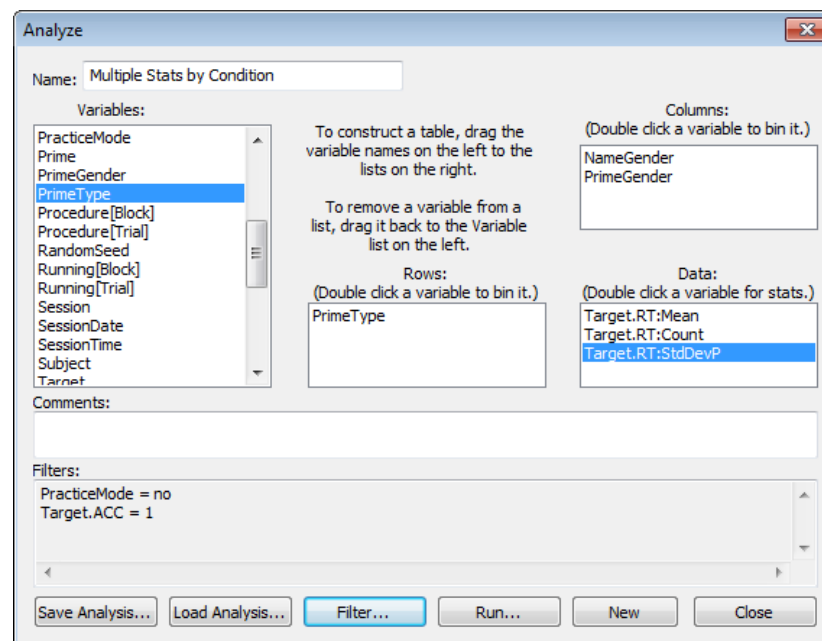
Stage 4, Step 3: Distribution of RT

The following analysis creates a distribution of response times in bins of 100 ms excluding practice trials and including only correct answers.
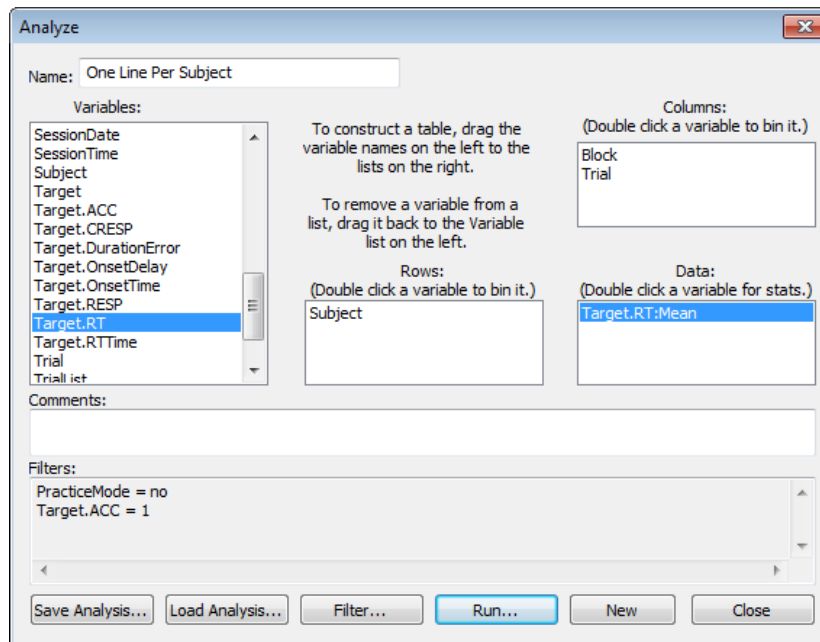

Stage 4, Step 4: Multiple Stats by Condition

The following analysis creates a table of count, mean, and standard deviation by conditions for real trials with correct answers only.

Stage 4, Step 5: One Line Per Participant

The following analysis creates a table of all response times on one line per participant for real trials only.


Stage 4, Step 6: Use the Results

Once a table has been created, it may be copied to the clipboard, exported to a text file for further analysis in another application, or plotted. E-DataAid allows the user to easily format a table, copy it, export it, and even plot it to Excel (Excel 2003, Excel 2007 or Excel 2010 must be installed on the working machine for this option).

Stage 4, Step 7: Format the Table

By default, when an analysis is run, the resulting table follows a default format. All row and column variables are listed in separate rows or columns, statistics are displayed down rows, all measures for a single variable are listed together, and measures are displayed to two decimal places of precision. However, recognizing that the user may need to format tables differently for different uses, E-DataAid allows a variety of format options. To format the table, click the Table Options button on the Table dialog to display the Table Options dialog.
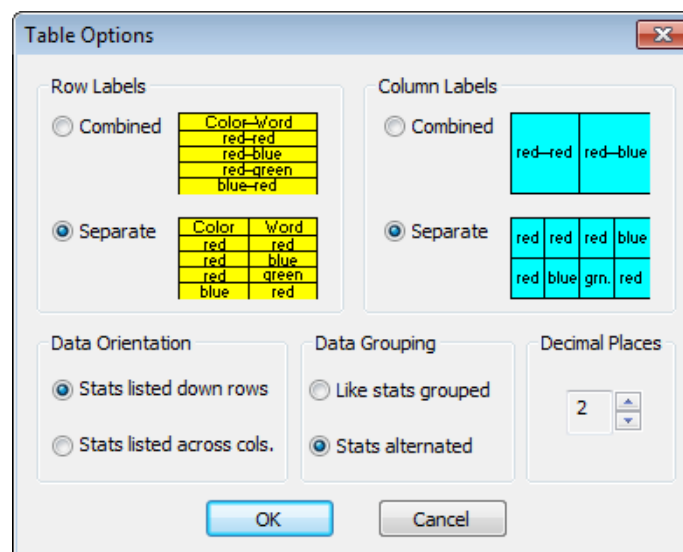

Each of the options may then be set to accommodate specific formatting requirements of various external or statistical packages. The following analysis creates a table of the proportion correct by participant excluding practice trials.

Stage 4, Step 8: Copy the Table

Once a table has been created, E-DataAid allows the table data to be copied to the clipboard, or directly to an Excel worksheet.

Clipboard

To copy a table to the clipboard, click the Clipboard button on the Table dialog. A dialog appears indicating if the copying operation was successful. Once copied to the clipboard, the table may be pasted into another application (e.g., Microsoft Word, PowerPoint).
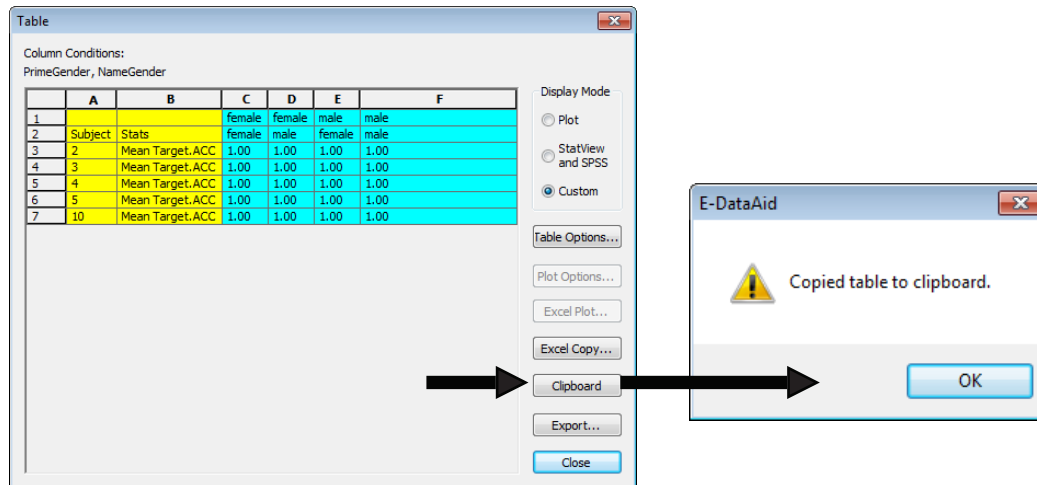
Excel Copy

If Excel is installed on the machine, the table may be copied directly to Excel by clicking the Excel Copy button on the Table dialog. If a workbook is currently open in Excel, E-DataAid will add a new worksheet to the workbook and copy the table to it. If Excel is not open, E-DataAid will open Excel and create a new workbook before performing the copy. The image below displays a table copied from E-DataAid to Excel.
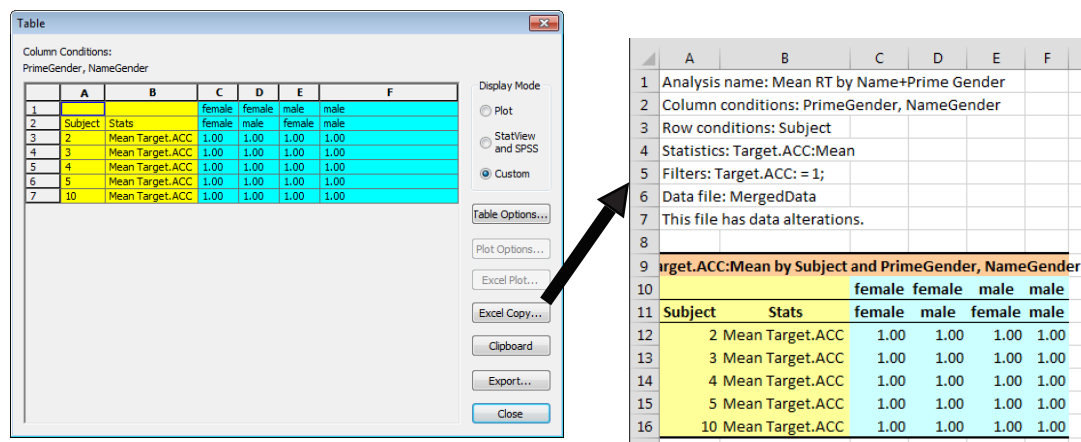
## Stage 4, Step 9: Export the Table

E-DataAid allows a table's data to be exported to a text file so that it can be imported by another application. This may be useful when performing further analysis on a table. Tables may be exported to Excel, StatView or SPSS, or some other application. Refer to the *E-Prime 2.0 New Features/Reference Guide* for a detailed description concerning each type of Export.

Stage 4, Step 10: Plot the Table

If Excel is installed on the machine, the table may be plotted directly to Excel. This feature is only available for tables that calculate one statistic for one variable. Before plotting the table, the table must be in the correct format. To correctly format the table for plotting to Excel, change the table's display mode to "Plot." The Display Mode setting is located to the right of the table . If the table cannot be plotted, this option will be disabled.
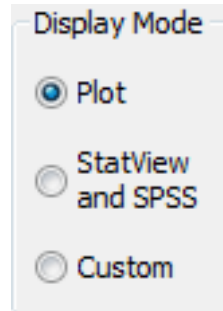
Once correctly formatted, the table may be plotted by clicking the Excel Copy button. If a workbook is currently open in Excel, E-DataAid will add a new worksheet to the workbook and plot the table to it. In addition to the plot, the application also copies the table to the worksheet. If Excel is not open, E-DataAid will open it and create a new workbook before performing the plot. The picture below displays a table automatically plotted in Excel.
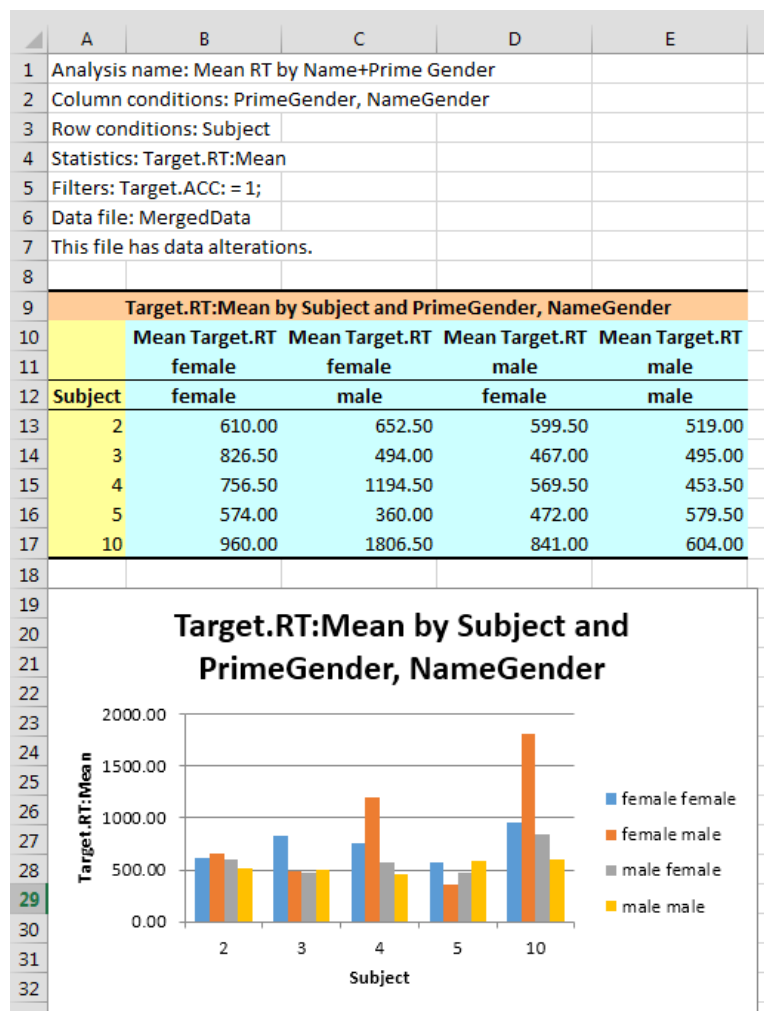
Before plotting, it may be necessary to change the plot settings. To change the plot settings, click the Plot Options button on the Table dialog to display the Plot Options dialog. This button is disabled unless the table's display mode is set to plot.
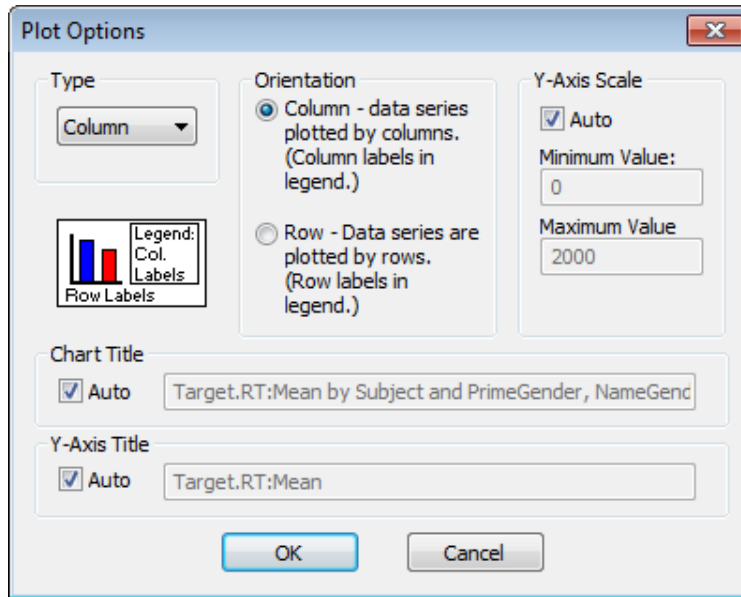
Select the plot type (column or line) as well as the plot orientation (column or row). A column orientation means that the application plots the data series by columns. In other words, the column labels make up the plot's legend and the row labels make up the x-axis labels. A row orientation means that the application plots the data series by rows. In other words, the row labels make up the plot's legend and the column labels make up the x-axis labels. The figures below compare and contrast four different plot combinations for the same table.
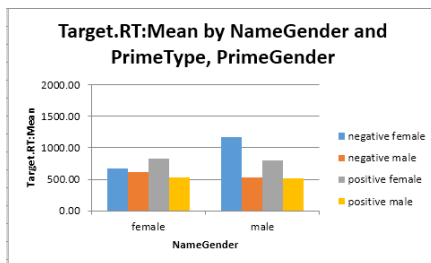

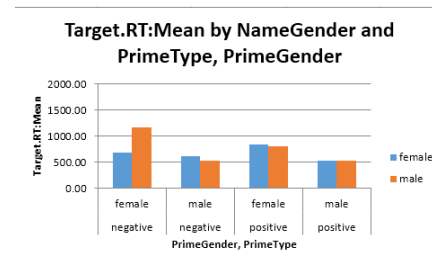*Figure 11. Column plot with column orientation.*

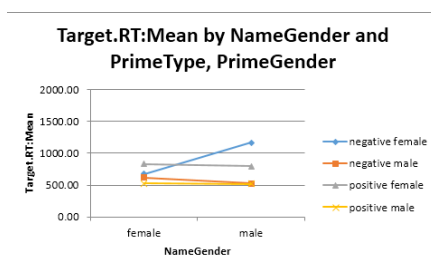*Figure 12. Column plot with row orientation.*

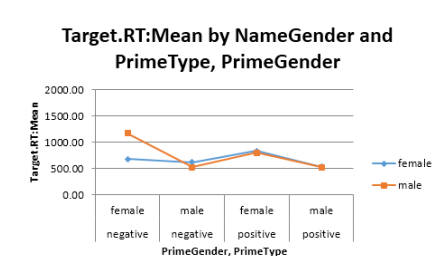

Figure 13. Line plot with column orientation.

Figure 14. Line plot with row orientation.

On the Plot Options dialog, the y-axis scale, chart title, and/or y-axis title may be set by unchecking the respective box labeled "Auto" and entering values into the adjacent fields. Note, once the plot has been exported to Excel, settings may be altered using the plot options in Excel.

6.5 Stage 5: Secure Data

E-DataAid provides light security features to prevent accidental viewing or editing of a data file. These security features are not foolproof, so be aware that a knowledgeable person intent on doing harm could potentially circumvent E-DataAid's security. Furthermore, E-DataAid's security features are designed specifically for an environment in which only one machine has E-DataAid installed on it (i.e., one machine is designated as the "analysis" machine). The remainder of this section assumes a single data analysis machine.

Applying or removing security from a data file involves the following steps:

1) Selecting an Administrator
2) Changing the Administrator's password
3) Opening the data file as an Administrator
4) Applying or removing security from the data file

Stage 5, Step 1: Select an Administrator

In E-DataAid, security is applied to or removed from a data file by opening the file as an Administrator. The Administrator has the responsibility of applying security to or removing security from data files. To open the file as an Administrator, a user must know the Administrator's password for the machine on which E-DataAid is installed. As soon as E-DataAid is installed, the Administrator should change the password from the default value.

Change the Password
When E-DataAid is installed, the Administrator's password is "admin" (without the quotes). As soon as E-DataAid is installed, the Administrator should change the password to something other than the default. To change the password, use E-DataAid's Change Password command on the File menu to display the Change Password dialog. A data file need not be open in order to change the password.

In the first field, type the current password (i.e., "admin,"", if not previously changed). The password will appear as asterisks when typed. In the second field, type the new password. Passwords are not case-sensitive and must be 4 to 8 alphanumeric characters. In the third field, type the new password again for verification. Click the OK button to change the password.

E-DataAid's password is machine dependent. This means that the password is stored on the machine, not in the data file. Thus, it is possible to use a different password on each machine on which E-DataAid is installed. A word of caution, however, about multiple passwords. If multiple Administrators are working on the same machine, all Administrators must share the same password. Or, if a single Administrator would like to be able to use multiple machines, all machines must use the same password.

Open as Administrator

To open a file as an Administrator, use E-DataAid's Admin Open command on the File menu instead of the usual Open command. Before allowing a file to be opened, E-DataAid will prompt the user for the Administrator's password.
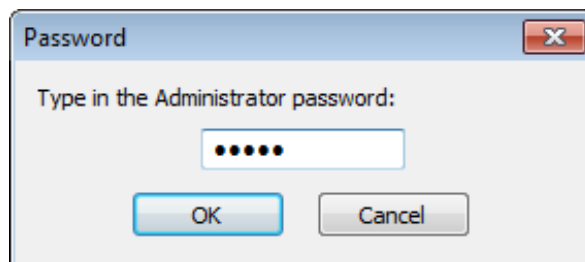
Type in the password and click the OK button. The Open File dialog box then appears, and E-DataAid allows a data file to be selected and opened.

Stage 5, Step 2: Set Security Restrictions

Once a data file has been opened via the Admin Open command, the Administrator can apply or remove security from the data file using E-DataAid's File Security command on the File menu. This command displays the Security dialog.
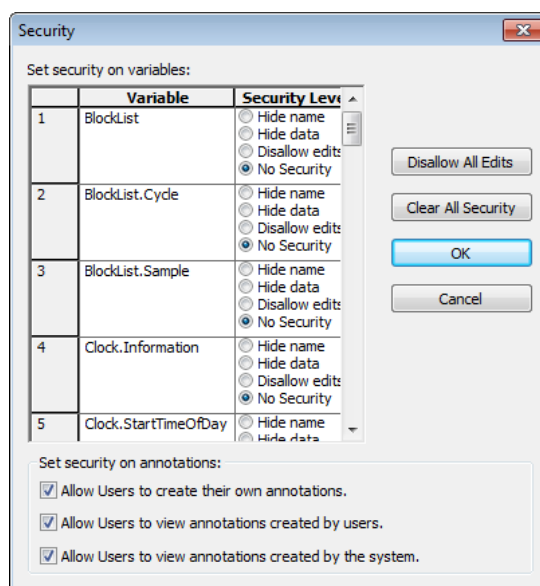
Most often, the Administrator will simply want to either make the file read-only or clear any security restrictions from the file. To make the file read-only, click the Disallow All Edits button on the Security dialog. To clear all security restrictions, click the Clear All Security button on the Security dialog. Click the OK button to apply the selected security settings, and save the data file. The data file must be saved, or the security settings will be lost when the file is closed.

While the file is opened as the Administrator, the data file will not look any different. To view the data file with the security restrictions visible, close the data file and reopen it with the Open command. Figure 15 on the left shows a read-only data file opened as an Administrator and Figure 16 on the right displays the same data file opened as a regular user.

Figure 15. Data file with security restrictions opened as Administrator.

Figure 16. Data file with security restrictions opened as user.

Security settings may be applied to specific variables. For each variable, the Administrator has the option of hiding the name, hiding the data, or disallowing edits. For example, if the Administrator sets the "Hide name" option for a variable, when the data file is opened with the Open command the spreadsheet will not display any columns for that variable . Even if the user tries to restore the spreadsheet, that variable will not be displayed in the spreadsheet. If the Administrator chooses the "Hide data" option for a variable, when the data file is opened with the Open command, the spreadsheet will display a column for that variable but the cells in that column will be blank. Likewise, the "Disallow edits" setting applied to a variable will result in a display containing a column for that variable with data in the cells, but the cells will be read-only. Similar security settings may be applied to annotations.

6.6 Stage 6: Export Data

E-DataAid includes the capability to perform limited descriptive statistics. E-DataAid also supports the ability to export data for analysis in other statistical packages. Both tables of summarized data and the entire spreadsheet itself can be exported. To export tables, use the Export button on the Table dialog, as described in *Stage 4, Step 9: Export the Table (Page 189)*. To export the spreadsheet, use E-DataAid's Export command on the File menu to display the Export dialog.

In the "Export to" list, select a format for exporting. The format options include StatView, SPSS, Excel, Compressed IFIS, E-Prime 2.0 Text, and Other. When exporting the spreadsheet, keep in mind that only the displayed data is exported. Any hidden columns or hidden rows will not be exported.

To export to analysis formats other than those specifically listed, select the Other option in the Export dialog and define the format using the available fields. The requirements of the package attempting to import the data will define appropriate field values for the format of the export. Note also that Excel is a standard export option. Most modern statistical programs will accept the tab delimited format of Excel files.

6.7 Stage 7: Import Data

E-DataAid provides the capability to import data that was previously exported from E-DataAid to an E-Prime 2.0 Text file. Refer to the *E-Prime 2.0 New Features/Reference Guide* (Section 5.3.10) for a detailed description of the import function.

Appendix A: Timing Test Results

Windows desktop computers can support millisecond precision timing if they are reasonably configured. It is prudent to run the timing test programs whenever new hardware or applications are added to the computer. The timing test results in this section show that E-Prime 2.0 can maintain millisecond precision on appropriately tuned desktop PC hardware for Pentium class machines running at 2GHz or faster. Experiments can take precision real-time input from the keyboard device or PST Serial Response Box, but we do not recommend that the mouse be used as an input device for experiments in which response time at the millisecond level is the primary dependent measure. Using a quality name brand PCI audio card, E-Prime 2.0 can achieve consistent low latency playback of digital audio (e.g., latency < 1 screen refresh). For experiments that require optimal and consistent playback latency, the recommended sound API varies with operating system. Please refer to the timing test summary data at the end of this section for a detailed review of the tests performed and the resulting data.

Applications frequently install either new background jobs that take processor cycles or new device drivers that may alter timing precision. Some hardware cards and software programs compromise the machine's ability to provide millisecond precision. The testing of the computer needs to be done when the computer is initially setup or when new hardware or software is installed, but it need not be done for every experiment. The rest of this section details how the testing should be performed. If the system has already passed the performance tests described and someone else is responsible for machine configuration testing, then this section may be skimmed, although the figures should be reviewed. Once the machine passes the performance tests, it is ready to serve as a data collection station for E-Prime 2.0.

People tend to install multiple background applications that may, at intervals, wake up and steal cycles from other applications (e.g., the experiment), resulting in timing errors. For example, the Windows Office toolbar periodically checks the state of the machine to see if some action needs to be performed (e.g., put up a reminder alarm on the screen). These types of programs use small amounts of time, but can block or delay other programs from executing and therefore can result in timing errors. Having multiple applications running encourages the operating system to swap out parts of the experiment (see *Chapter 4: Critical Timing*). A common problem is virus detection programs that scan all files and periodically scan the system for viruses. There is an internal timer in the virus monitoring applications that will, on some cycle, wake up and scan the system, halting activity for potentially hundreds of milliseconds. To avoid random time delays during the experiment, scanning must be disabled while running the experiment. E-Prime 2.0 runs in high priority mode and will block most, but not all such background jobs.

A good experimental computer should have a minimum number of concurrent applications running and no other applications loaded during a data collection run. Ideally, no other applications other than E-Run should be displayed on the Windows taskbar when real data is being collected. There should also be a minimum number of background applications and utilities running (this is typically determined by what programs are on the Start menu configuration for the computer but can also be affected by various non-obvious entries located in the Windows System Registry). Pressing Ctrl+Alt+Del and choosing TaskManager, will display a list of all programs currently running on a computer. Figure 1 is a display of a completely clean computer. This is the ideal scenario and may not be possible to achieve on all hardware setups because of various drivers that may be required for proper operation of the computer hardware. The Windows Explorer application will always be listed, as it is responsible for running the Windows desktop . If other applications or utilities are listed due to the configuration of the machine, be aware of the tasks each program performs. It is important to know if these programs are indeed necessary while running an experiment, since the programs may be taking processor time.
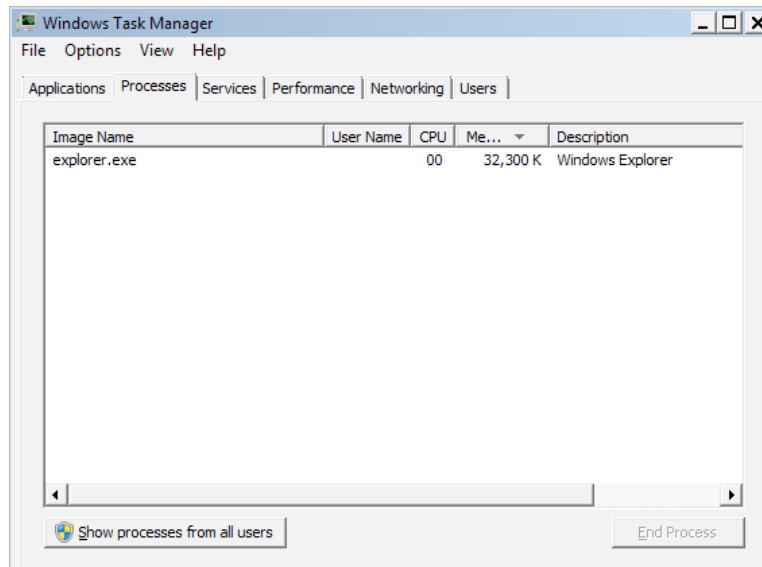
Figure 1. Ideal listing of operating programs before running an experiment. Note this is the ideal minimum. Other programs will be displayed depending on the configuration of the machine.

Clock Test

The E-Prime 2.0 Clock Test monitors the clock for a period of 10000 ms and checks to see if continuous readings of the clock ever failed to identify sequential clock ticks. E-Prime 2.0 cannot completely stop the operating system from suspending an executing experiment. However, E-Prime 2.0 can determine if the reading of the clock was halted long enough to skip a clock tick (e.g., if the values 2001, 2002, 2005, 2006 were returned on 4 successive clock reads, E-Prime 2.0 sees the 3 ms tick between 2002 and 2005), indicating that the operating system would, in a similar situation, have delayed the experiment. E-Prime 2.0 uses a microsecond precision clock for all internal time assessment. If E-Prime 2.0 is unable to read the crystal clock when it needs to (e.g., to timestamp the onset of a stimulus), the precision of the timing is compromised (see *Chapter 4: Critical Timing*).

Having a well-configured machine is critical for achieving accurate timing during the run of an experiment. Figure 2 illustrates the importance of the configuration of the hardware and how dramatically it can impact millisecond precision.
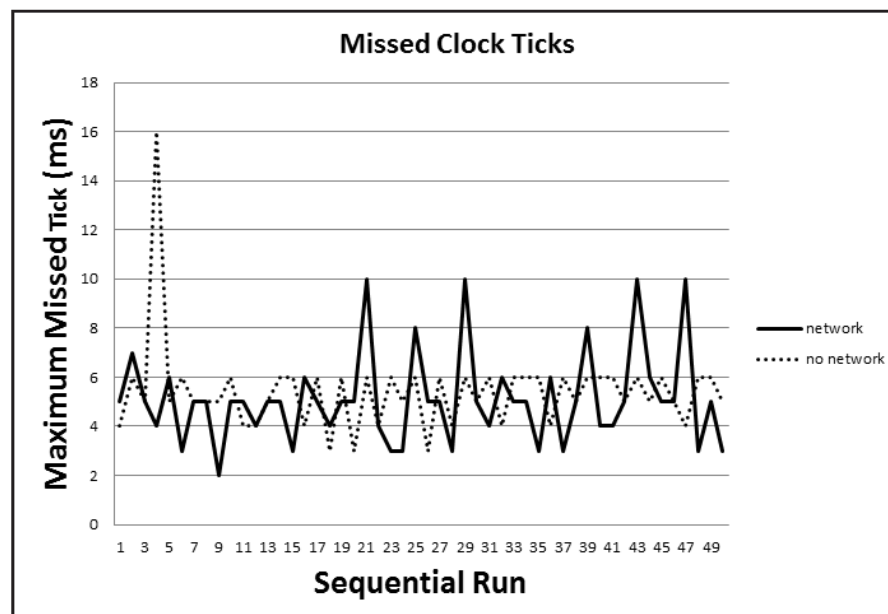
Figure 2. Missed ticks test of 10,000 ticks on a computer with and without the network connected.

The test was run on a Pentium 1.8 GHz computer with a network card. Although this computer is capable of delivering millisecond accuracy, it cannot do so with the installed network card and version of the network software. Better network cards and drivers on other machines can provide millisecond precision while supporting network activity. One of the reasons for doing long duration tests (e.g., 6 hours) is to see if a system is sensitive to the behavior of other devices on the network.

Figures 3 displays the timing error pattern during the six-hour time test. A perfect run would be a straight line, indicating that there was no loss of ticks during the run. Three computers were tested. Of the three computers, the fastest two (3.4 and 2.9 GHz respectively, both mutli-threaded) showed 0 missed ticks. The timing test run on the third machine (1.8 GHz, single core, running XP) shows a repeating pattern of missed ticks, settling around 10% after a larger rate on the first 2 runs ranging from 2% to 4% (Figure 3).
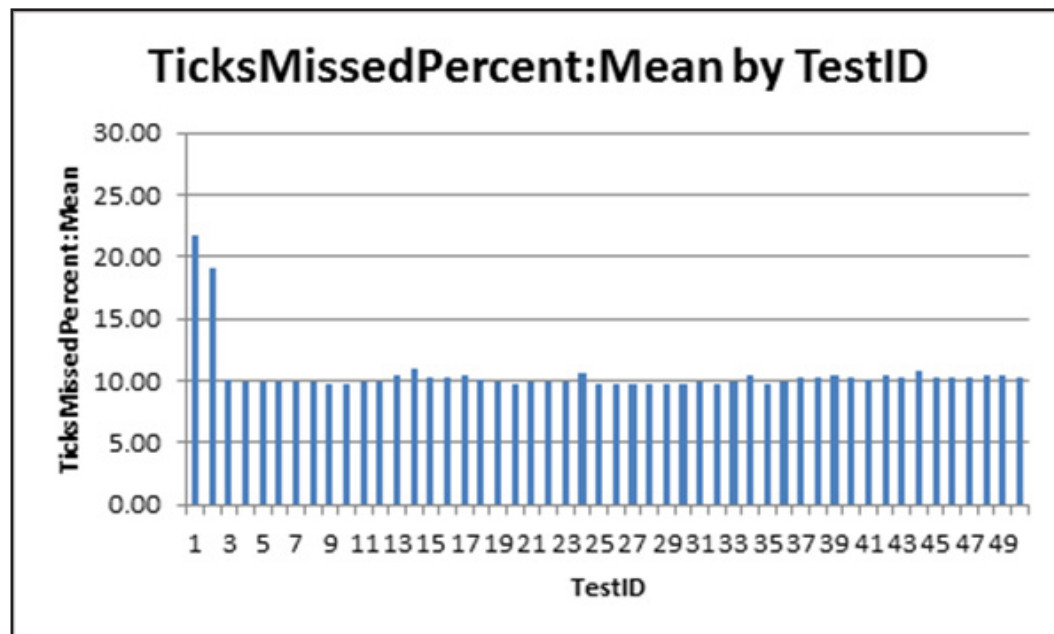
Figure 3. A time loss pattern.

The impact on the data of missed ticks is determined by the percentage of missed ticks. We recommend that the percentage of missed ticks be less than 0.1%. Note, a high rate of missed ticks early in the run is likely caused by the operating system stabilizing the running of the program.

The clock bin count test (Figure 4) gives a distribution of the delay times in table and graph form. It shows how many counts occurred for each time interval. This can be used as a diagnostic to suggest what programs are taking cycles. Below is the distribution of tick sizes for a HP laptop. There were a significant number of ticks in the first bin. Potentially, a program like the power conservation program is producing interrupts.
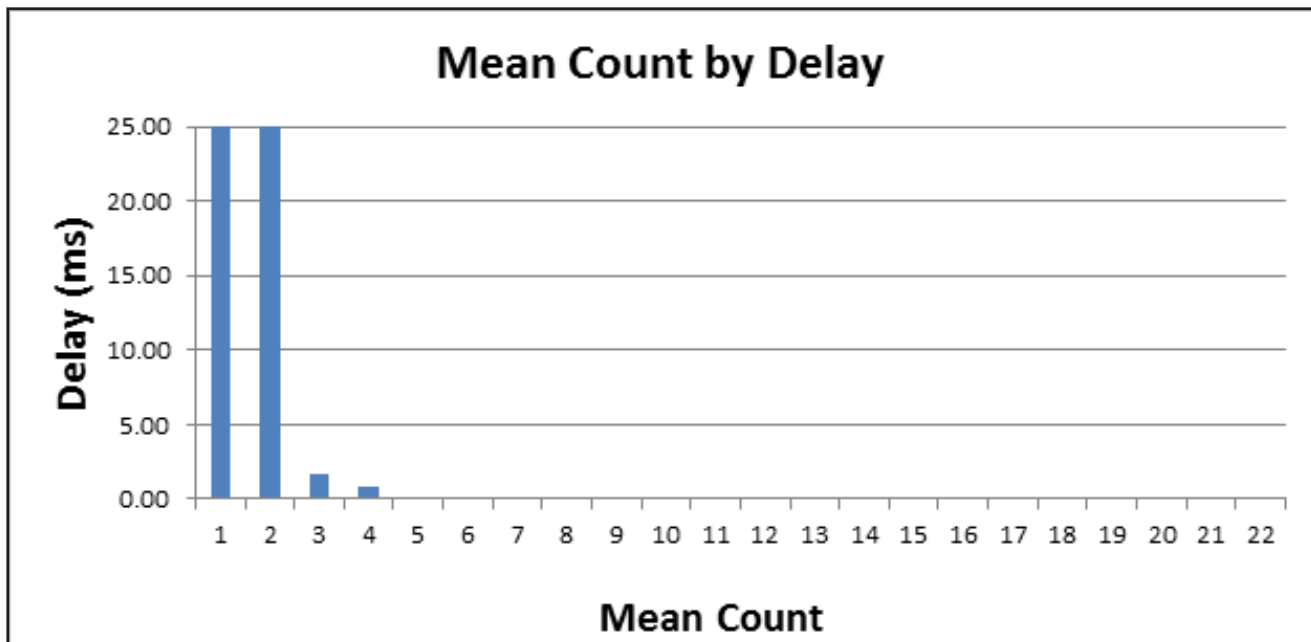
Figure 4. Distribution of tick times from a laptop computer, showing frequent interrupts.

Configuring computers is a complex process and may require the assistance of a technician who is experienced in installing and removing software applications and/or hardware. The main strategy is to run a test. If timing is a problem, remove programs, and possibly hardware, until the system can provide the precision needed for professional research. E-Prime 2.0 provides the tools to assist the researcher in detecting configuration problems that disrupt timing prior to the implementation of experiments.

E-Prime 2.0 Timing Tests

For research experiments, it is absolutely critical that the timing be tested rigorously. PST has developed an automated timing test station, connected to a HP ZD7000 2GHZ single core laptop that can evaluate timing accuracy on both Windows (Intel, AMD) computers. Under many test scenarios, the test station can simulate responses made from an interactive participant.

Timing Station (TS) and Experiment Station (ES)

The Black Box Toolkit (BBTK, http://www.blackboxtoolkit.com/) served as the timing station for the remainder of the test described in this section. The Experiment Station (ES), running the software and experiment scripts to be tested, consists of any Pentium 4 class PC (1.8 GHz+) running Windows XP or Windows 7 and 8. All tests that are conducted on the ES machine are written as E-Prime 2.0 experiments. InLine script was used to call ReadPort/WritePort or Task Events using the ParallelPortDevice commands in order to control the TTL lines wired to the BBTK. The data that is collected on the BBTK is merged with data from the EDAT (E-Prime 2.0 data file) and imported into Microsoft Excel for further processing and analysis.

Tests were conducted on a wide range of machine configurations. Sample results provided in this text are from a representative 'low-to-mid-range' class machines* with the following specific configuration.

- Intel Pentium 1.8 GHz Processor or AMD 2.9 GHz Processor
- 1 GB Ram
- Windows XP or Windows 7 and 8 Operating System
- Not connected to a network

Results for other classes of machines will be periodically updated on the Psychology Software Tools web site.

Test Equipment Details

- Microsoft 2 Button PS/2 Mouse (modified for use) –the Microsoft mouse was modified on the left switch to parallel in a high-speed reed relay (see #3 above).

- USB Keyboard (modified for use) – the keyboard was modified with transistor and digital i/o gate circuitry to allow a parallel port bit to trigger a button press.

- Microsoft USB IntelliMouse (modified for use) – the mouse was modified with transistor and digital i/o gate circuitry to allow a parallel port bit to trigger a mouse button click.

- PST Serial Response Box Model 200a – the SRBOX was not modified; transistor and digital i/o gate circuitry was added to the expansion connector to allow a parallel port bit to trigger a button press.

Test Methods

1) Clock Bin Test – In this test, the ES machine continuously reads the E-Prime 2.0 real-time clock for a preset period of time (e.g., 10 seconds). It then subtracts the difference between consecutive clock-read values to determine if any type of delay (e.g., perhaps caused by the operating system or some other running application) occurred between the clock reads. If no delay occurred, the values should either be the same (i.e., multiple reads during the same millisecond) or different by just 1 ms. Any difference greater than 1 ms is a potential timing error. The delays are grouped into bins based on the duration of the delay. External hardware is used to ensure that the test ran for the expected amount of time (e.g., to ensure that the real-time clock was never paused or halted).

2a) Fixed Duration Clock Test – In this test, the ES machine toggles a single bit connected to the BBTK in order to produce either a fixed or varying duration square wave. The BBTK timestamps each bit transition observed, and reports the results. A sample table of results for the fixed duration tests (run at 1, 10, 100, and 1000 ms durations) is shown below.

ACTUAL DURATION BY EXPECTED FIXED DURATION Expected Fixed Delay (ms)						
	1	10	100	1000	10000	Grand Total
Average of L1 Duration	1.536	10.467	100.714	1000.687	10000.753	100.577
StdDev of L1 Duration2	0.360	0.256	0.668	0.164	0.206	589.138
Min of L1 Duration3	1.120	9.776	100.252	1000.309	10000.547	1.120
Max of L1 Duration	6.007	16.007	115.648	1001.084	10001.141	10001.141
Count of L1 Duration4	1000	1000	1000	100	10	3110

ABSOLUTE TIMING ERROR BY FIXED INTERVAL Fixed Interval (ms)						
	1	10	100	1000	10000	Grand Total
Average of ABS Error	0.536	0.468	0.714	0.687	0.753	0.577
StdDev of ABS Error	0.360	0.255	0.668	0.164	0.206	0.467
Min of ABS Error	0.120	0.046	0.252	0.309	0.547	0.046
Max of ABS Error	5.007	6.007	15.648	1.084	1.141	15.648
Count of ABS Error2	1000	1000	1000	100	10	3110

2b) Varying Duration Clock Test – This test is the same as #2a, but the durations used are incrementing delays of 1, 2, and 3 ms as well as delays incrementing by prime numbers. Sample results are shown below.

ABSOLUTE TIMING ERROR BY VARYING INTERVAL						
Fixed Interval (ms)						
	+1	**+2**	**+3**	**PRIMES**	**Grand Total**	**Grand Total**
Average of ABS ERROR	0.824	0.699	0.473	0.665	0.695	0.134
StdDev of ABS ERROR	0.355	0.235	0.201	0.200	0.317	0.100
Min of ABS ERROR	0.003	0.241	0.065	0.349	0.003	0.001
Max of ABS ERROR	4.772	2.957	2.002	0.970	4.772	1.528
N (Number of trials)	471	338	278	25	1112	3110

2c) Refresh Rate Test – This test verifies that the ES can reliably detect and sync to the video vertical retrace signal to get the exact number of refreshes requested by the experiment. In the sample test below, the ES displayed stimuli while a photo-diode optical sensor detected each display. The photo-diode then signaled the BBTK upon detection of a retrace, and the BBTK timed the period between each signal (i.e., the refresh duration).

ABSOLUTE ERROR IN REFRESH DETECTION		
(ms)		
	Low-Range Test Computer, 60 Hz	**Mid-Range Test Computer, 120 Hz**
Average of ABS ERROR	0.988	0.679
StdDev of ABS ERROR	0.20	0.19
Min of ABS ERROR	0.3866	0
Max of ABS ERROR	1.290	1.318
N (Number of trials)	594	1189

3a) Keyboard Response Test – This test creates a simulated participant response on the keyboard. The BBTK plays the role of a participant as well as that of a recording device. The BBTK monitors the display with a photo-diode sensor. When it detects a stimulus, it delays a preset duration of 100 ms and then sends a TTL signal to a modified keyboard, thereby simulating a key press. Two different computer configurations were tested, a low and a medium range computer. The sample data below shows that the keyboard produced a fairly constant delay averaging about 12 ms with a standard deviation of <3 ms. Considering that normal human response times generally have a standard deviation of over 100 ms, the added variability is negligible. Also, since the delay is likely to be equal across all conditions and is represented in the base reaction time, it is unlikely to influence experimental results.

⚠ **NOTE:** *Not all keyboards (including those of identical model) will not have the same latency characteristics.*

ACTUAL KEYBOARD DELAY BY TEST ES COMPUTER (low or mid-range) ES Computer Type		
	Low-Range Test Computer	**Mid-Range Test Computer**
Average of ACTUAL DELAY	12.408	11.571
StdDev of ACTUAL DELAY	2.397	2.398
Min of ACTUAL DELAY	8.000	7.000
Max of ACTUAL DELAY	17.000	16.000
N (Number of trials)	49	49

3b) Mouse Response Test – This test creates a simulated participant response on the mouse device. The BBTK plays the role of a participant, as well as that of a recording device. The BBTK monitors the parallel port for the start of each trial. When it detects a stimulus, it delays for a preset duration of 100 ms, and then sends a TTL signal to a modified keyboard, thereby simulating a key press. The sample data shows that a common mouse is not typically a valid device to use for precision reaction time experiments (e.g., both the average delay and standard deviation is generally too high to be considered valid for RT experiments). The mouse can be used as an input device, but should only be used for RT experiments in which millisecond timing precision is not required.

ACTUAL MOUSE DELAY BY TEST ES COMPUTER (low or mid-range) ES Computer Type		
	Low-Range Test Computer	**Mid-Range Test Computer**
Average of ACTUAL DELAY	22.67	17.92
StdDev of ACTUAL DELAY	2.73	2.32
Min of ACTUAL DELAY	15	14
Max of ACTUAL DELAY	27	22
N (Number of trials)	49	49

3c) PST Serial Response Box Test – This test creates a simulated participant response on the PST Serial Response Box device. The BBTK plays the role of a participant as well as that of a recording device. The BBTK monitors the parallel port for the start of each trial. When the start of trial detects a stimulus, a simulated response is made on the Serial Response Box, as if the participant had pressed a key. The sample data below shows that PST Serial Response Box device can produce much more accurate response times relative to a modified input device (e.g., a delay averaging about 2.245 ms with a standard deviation of <0.75 ms when running in native mode). Note that the negative response time shown as a minimum is possible with the Serial Response Box since the software attempts to subtract out the average data transmission delay, and as such, a very small negative response time is possible. If buffer flushing is enabled in E-Prime 2.0 (the default on all devices), these negative responses would be filtered out of the data stream as an anticipatory reaction.

ACTUAL SRBOX DELAY BY CONNECTION TYPE		
	Native Connection	**USB Connection**
Average of ACTUAL DELAY	2.245	2.612
StdDev of ACTUAL DELAY	0.723	0.702
Min of ACTUAL DELAY	1.000	1.000
Max of ACTUAL DELAY	4.000	4.000
N (Number of trials)	49	49

4) Audio Output Latency – This test checks the delay between when an audio output is initiated to the time until a signal is detected at the output of the audio card. During this test, the BBTK audio microphone detector unit is placed near the speakers of the ES station. When an audio signal is detected, a pulse is generated on a TTL line connected to the BBTK. The ES prepares an audio stimulus for output, sends a start signal to the BBTK, and then immediately initiates playback. The BBTK watches for the audio detect signal from the interface hardware, timestamps the event, and reports the time between the start and detect signals.

Our testing shows that audio performance can vary widely depending on the audio card and the combination of operating system and API being used. Desktop audio systems are primarily designed to meet the needs of the computer gaming market. This market has very soft real-time constraints (e.g., delays on the order of 1-3 screen refreshes are not typically a concern or even noticed by a game player). The tests indicate that with the appropriate sound card, you can achieve low and consistent playback latency.

As a result of our tests for Windows 7 and 8, PST generally recommends on board sound with the default High Definition Audio Device driver and the CoreAudio API. In Windows Vista, we recommend an add-on PCI or PCIe sound card with the manufacturer's driver and the CoreAudio API. For Windows XP, on board sound was found to be comparable to some add-on cards. Please see the KB 1307 - INFO: Sound Latency: Not all sound cards provide optimal millisecond timing.

Appendix B: Considerations in Computerized Research

-- Contributed by James St. James, Millikin University

While many of the persons using E-Prime 2.0 are intimately familiar with the intricacies of research, many are not, or are just beginning to learn. In this chapter, we present a brief overview of experimental research, including review of some basic terms and ideas. It is not the purpose of this document to serve as an introduction to the use of E-Prime 2.0. Rather, its purpose is to aid the user of E-Prime 2.0 in the conceptual development of experiments, and to consider a number of broad classes of issues regarding experimentation in psychology. If you feel that you are quite familiar with research methodology as it pertains to psychological research, feel free to skip this chapter. This chapter is particularly useful for those in need of a 'refresher course' in research methods.

Experimental Design Considerations

We begin with a consideration of some basic principles of research design. Then we consider a number of details of the single-trial, reaction time paradigm that is the basis of much of current psychological research. Skip any sections that cover familiar material. Parts of what we include below will seem too obvious to some, but we hope to aid the person using E-Prime 2.0 who will benefit from a reminder of basic terms, or who is just beginning the process of learning to do research. Our emphasis here is on experimental research, and our examples lie there, but observational and correlational research requires consideration of many of the same points.

Because what follows is not a complete textbook of research methods, the references cited are largely to general sources. The reader is encouraged to go to those sources for more complete coverage and primary references. Many textbooks of research methods in psychology are available that will treat the general topics below in more detail. Most do not include considerations specific to single-trial reaction-time procedures, which we detail.

Definitions

Because they are so widely used in our discussion, we begin by defining dependent and independent variables and controls.

Dependent and Independent Variables

In designing and setting up an experiment using E-Prime 2.0, independent and dependent variables will have to be named. Dependent variables (DV's) are measures of outcome, such as reaction time and accuracy. Independent variables (IV's) are the aspects of an experiment that are manipulated by the experimenter. Note that, in an experiment, the value of the outcome measure is assumed to depend upon, or be caused by, the condition under which the participant was tested—the level of the independent variable. Hence, it is a dependent variable.

Independent variables have two or more levels, which define the conditions under which the participant is tested. Examples would be type of stimulus, timing of stimuli, or any other aspect of the experiment that will be manipulated. The independent variables may be manipulated by randomly assigning participants to conditions (levels of the IV), or by applying each condition to each participant, in a random or counterbalanced order. In discussing statistical analysis of experiments, independent variables are also sometimes called factors. An experiment with more than one IV is said to use a factorial design.

Controls

Confounding variables are aspects of the experimental situation that are correlated with the IV's that the experiment is intended to study, and that may be producing (or hiding) differences among different levels of the IV's. An example may help. Suppose that a researcher wishes to compare two methods of teaching basic statistics. She teaches two sections of the course, and so decides to teach her 8:00 am class by Method A and her 10:00 am class by Method B. Suppose she finds that students who learned

by Method B have significantly higher mean scores on the common final exam than those taught by Method A. Can she conclude that Method B is better? Hardly. Perhaps students are less alert in 8:00 classes than in 10:00 classes. However, suppose that she finds that there is no difference between the classes on the final exam. Can she conclude that the method of teaching doesn't matter? Again, hardly. In this case, perhaps Method A is actually superior, but the 8:00 class was only half awake, and the only reason they did as well as those taught by Method B was that Method A was sufficiently better to overcome the problem of inattentiveness. In this example, time of day is confounded with method of teaching. (Confounding method of teaching with time of day is not the only problem with this design. The lack of random assignment of students to classes is also a problem.)

Controls include any aspects of the experiment that are intended to remove the influence of confounding variables. Controls are usually intended to remove variability caused by confounds, by making them constants, not variables. In the example above, that would involve controlling the time of day at which the classes were taught. Another example: In research involving visual displays, be concerned about the effective size of the stimuli, which would vary if different participants sat at different distances from the computer monitor. In that case, a relevant control would be to use a viewing hood or chin rest to make the distance from the screen the same for all participants. A third example is: make sure that equal numbers of males and females are in the group of participants tested at each level of the independent variable, if it is suspected that there might be sex differences in performance. By having equal numbers of males and females, any effect of sex would be the same for each level of the IV, and any differences in average performance for the two levels would not be due to having more males in one group or more females in another.

Note that in the case of assigning equal numbers of males and females to each level of an IV, sex has actually been added as a blocking variable. If recording the participants' sex in the data file, later analyze the data separately for males and females to explicitly check for sex differences. Blocking variables should always be included as "independent variables" in a data file. An advantage to matching groups on a blocking variable is that it serves to control that confound and to permit examination of its influence.

Order effects are an important class of confounds, especially in experiments in which each participant serves at each level of the IV. Here, experiencing one level of the IV may change performance at another level. Examples would be when experience in one condition provides practice that improves performance in another condition, or when experience of the first condition induces a strategy that affects performance on the other. Two general solutions are available: counterbalancing and randomization. Complete counterbalancing guarantees that each condition precedes or follows each of the others equally often. (For experimental designs with many levels of an IV, complete counterbalancing is usually not possible, due to the number of participants required. In this case, a Latin square design can approximate complete counterbalancing.) An alternative is to randomize the order of presentation of the experimental conditions for each participant. Over a fairly large number of participants, this will approximate counterbalancing. Note that with either counterbalancing or randomization, recording the order of the conditions in the data file will permit later comparison explicitly on the performance of participants receiving different orders of the experimental conditions.[1]

---

[1] We realize that it is now fashionable to refer to the persons from whom we obtain data as "participants" (Publication Manual of the American Psychological Association, 4th ed., 1994). We continue to use the term "participant," because the whole point of doing an experiment is that you, the experimenter, manipulate the independent variable. It is precisely because the person has agreed to temporarily suspend control and let you decide the level of the IV to which they will be exposed, or the order of the levels, that makes the study an experiment. Of course, the participant may remove himself or herself from participation at any time, but for as long as they participate, participants have allowed you to participant them to the conditions you choose. "Participant" suggests a level of free choice that is not a part of an experiment (beyond the freedom to choose whether or not to participate and to withdraw).

Before Beginning

Before beginning to design an experiment, carefully consider the broader research question that is trying to be answered. While the use of computers with software such as E-Prime 2.0 makes it easier to run experiments, there is still a lot of cost involved in paying participants, in time testing participants and analyzing data. For that reason, time spent "up front" on careful theoretical considerations will avoid wasted effort and increase the chance of getting an interpretable and publishable result. In this section, we consider a number of issues that need to be addressed before and during the detailed process of experimental design.

What are the questions that need to be answered?

Before beginning to design an experiment, have a clear formulation of the questions trying to be answered. Specify a hypothesis, or a statement about the expected effects of an independent variable on the dependent variable (e.g., reaction time will decrease as the flanking letters are moved farther from the target letter). The hypothesis may come from an explicit theory, may represent an extension of previous research, or may come from personal observation. In exploratory research, the questions may concern the nature of a phenomenon—what are the conditions under which the phenomenon (e.g., a visual illusion) occurs? Here, the concern is not with testing a theory, but with delineating a phenomenon. In confirmatory research, the research questions concern the explicit test of a theory about the nature of a phenomenon. If the experimental result is predicted in advance by the theory, that tends to confirm the theory. However, if an experimental result contradicts a prediction of the theory, it suggests that the theory is at least incomplete, and possibly incorrect. (A thorough discussion of the confirmation and falsification of theories lies far beyond the scope of this chapter. See Elmes, Kantowitz, & Roediger, 1992.)

How can the research questions be answered?

Whether the research is exploratory or confirmatory, the questions to be answered must be as specific as possible, so that what would count as evidence is clear. It is important that the questions be posed in such a manner that some kind of experimental task can be devised that can answer the questions. Comparisons are at the heart of any scientific question—it is expected that a dependent variable will, in fact, vary as the level of the independent variable(s) is changed. In confirmatory research, there is a specific prediction of at least the direction (possibly the degree) of the differences in DV's as IV's vary. For example, a theory might predict that RT would increase as the intensity of some IV changes. In exploratory research, there is no precise prediction about how the DV will change, but there is the expectation that changes in the IV's studied will produce changes in the DV. If they do not, not much exploration has taken place.

How will data be analyzed?

The experiment and the data analysis should be co-designed. It is extremely important that the methods of data analysis be known in advance of collecting the data. There are several reasons why this is so. Since the point of the research is to make comparisons of DV's at varying levels of IV's, it should be clear in advance what comparisons would be made and how they will be made statistically. This can avoid nasty surprises later, such as discovering that a crucial variable was not recorded, or (worse yet) that no appropriate statistical test is available. There is some additional discussion of statistical analysis of single-trial RT data below.

Before collecting the data, it is useful to draw a graph of the data, to be clear as to what patterns of RT's would support or disconfirm the hypothesis. If there are other possible assumptions about the effects of the IV(s), graph those as well. Such a graph will help clarify predictions. Plotting the expected means along with expected standard error bars (perhaps derived from pilot testing) can give a good perspective on what is expected to be seen, and what differences might be significant. As a rule of thumb, a difference between the means of two standard errors is likely to be significant. A statistical power analysis is useful as well, to help judge the likelihood of getting a statistically significant difference between means based on the size of the differences expected, the variability of the data, and the sample size.

How will the experimental tasks be presented?

A careful review of the pertinent literature is a natural starting place for any research, usually focusing on previous theoretical and empirical developments. However, a review of Methods sections of experiments using similar tasks is also likely to be rewarding. Such a review may alert to considerations not thought of, saving much pilot testing. If there is literature or research using similar tasks, it might be worthwhile to discuss the design with the authors and take advantage of any insights not made a part of the formal report of Methods.

A host of considerations comes into play in the detailed design of an experiment and the analysis of the data it produces. While some are specific to a restricted research domain, others are more general. The discussion below of the minutia of single-trial, reaction-time research highlights many of those considerations.

Implementing a Computerized Experiment

Once you have thoroughly thought out the question you wish to answer and how you plan on answering it, you are ready to begin designing the experiment. Do not rush the previous planning stage. It is critical to have properly prepared before attempting to design or implement an experiment.

Constructing the experiment

Work from the inside out (or the bottom up). The best way to construct an experiment is to get a few trials going before typing in full stimulus lists and instructions. We recommend leaving instruction screens blank and specify a minimal list of stimuli; usually one from each level of a single IV is sufficient. Once certain that the basic trial is running and that the data are stored correctly, add the other IV's, additional stimuli, instructions, and other details. It is fairly often the case that in setting up an experiment, it becomes clear that further variables need to be specified in stimulus lists. Going back to a long list of trials and adding those to each can be frustrating. Thorough testing with a few trials of each type will usually catch such errors while they are easy to correct. As an example, suppose that you have to type in 200 words to serve as stimuli, and designate each word by frequency and length. If you then decided to add concreteness as an additional IV, you must enter the concreteness rating for each of the 200 words. However, if you first test the experiment with only four words, and discover that an additional IV is needed, only four levels must be fixed.

Pilot testing

Once the experiment is set up, perform a couple levels of pilot testing. The first level is to sit through the whole experiment alone. You may notice errors you did not spot before, or realize that the experiment is too long and should be run over multiple sessions. Do not expect participants to undergo an experimental procedure that you are not willing to sit through yourself. This is especially important if someone else sets up the experiment according to your specifications. As the Cold War arms-control negotiators used to say, "Trust, but verify." The second level of pilot testing should be to have two or three people complete the experiment. These should be laboratory assistants, colleagues, or others who might spot potential problems. Especially if using students as pilot participants, let them know that reporting anything that seems like a problem is necessary.

Once the pilot data are collected, perform a full data analysis. Although it isn't likely that so few participants will give the statistical power needed for "significance," you can satisfy yourself that the relevant variables are recorded and that you know how to proceed with the analysis. An important aspect of analysis is to know how to structure the data for the program that is being used for analysis. Note that most statistical programs will read in a tab-, comma-, or space-delimited ASCII (or DOS) file, which should have the data for each participant on a single line. With reaction-time research, it is common to use the mean RT's for each condition as the data for analysis, rather than single-trial data. That can be produced using the Analyze feature of the E-DataAid application within E-Prime 2.0.

Formal data collection

Once formal data collection has begun with actual research participants, it is a good idea to debrief at least the first few participants rather extensively when they finish the experiment. Check that they understood the instructions. Ask whether they noticed anything that seemed unusual, and ask them about strategies they may have adopted. Participants sometimes read into an experiment all sorts of demand characteristics that the experimenter never intended. Do not assume that the participants are going to tell about aspects of the experiment that bothered or puzzled them. Therefore, explicitly ask whether there seemed to be anything "wrong." Note also that the colleagues or laboratory assistants who may have served as pilot participants bring a special expertise to bear, so they may spot problems a naïve participant would not. However, they may also, for the very same reason, overlook problems in instructions and procedures that will bother the naïve participant.

Also review both single-participant and mean data as the first few participants complete the experiment. Look for extremes of variability. In a single-trial, reaction time paradigm, look at a table of mean RT's by trial type, standard deviations and error rates. Extreme standard deviations or error rates may indicate that a particular trial type is not being presented as expected, or that participants are not reacting as instructions suggested.

Familiarizing participants with the situation

Especially with computerized experiments, it is sometimes necessary to spend time making sure participants are comfortable, because they need to do the experimental task without being distracted by the surroundings. Many researchers rely on undergraduates as participants, and can assume a familiarity with computers. However, in an elderly population, the participants may not be familiar with computers. In research with psychiatric patients, a strange situation may significantly alter the participant's ability to comprehend and focus on the experimental task. Giving a session of practice just to overcome the threat of a new, strange environment may be well worth the extra time and trouble. Research with children introduces other problems, such as understanding instructions. The use of a response box with just a few keys might be helpful in some of these situations by reducing the distractions inherent in a computer keyboard with its 100+ keys.

If data collection will take place on a computer other than the one used for setting up the experiment, be sure the pilot testing is done on the computer to be used for the final experiment. Problems that can arise include differences in display sizes, as well as problems of switching to a different graphics adapter.

Where will data collection take place?

Give some prior consideration to the setting for data collection. Most often, this is done with one participant at a time, in a laboratory setting. Sometimes, however, the data collection may take place in a hospital or clinic, or another setting.

Considerations in regard to the location of the laboratory include:

1) Limiting outside noise and interruptions. If tests must be done in a noisy environment, it may help to use a white-noise generator being played over a speaker or headphones to block most extraneous sounds. If several participants are tested at once, it helps to have dividers or separate booths, since this discourages the participants from chatting among themselves.

2) Control of lighting. Glare on the computer monitor is sometimes a problem, especially in a relatively dimly lit room. This can be a major problem when using brief, data-limited displays. Adjust the position of the monitor to eliminate glare. Also adjust the brightness and contrast so that the display is clear and sharp. Note that a display seen in a brightly lit room may look too bright (and fuzzy) when the lights are dimmed.

3) Control of access to the computer. It is a good idea to place the computer itself where the participant cannot reach the controls (i.e., so that they do not reboot the machine, pop out a floppy disk, or adjust the monitor settings).

4) Comfort. If participants must sit for lengthy experimental sessions, be sure to have a comfortable chair. A few minutes spent adjusting the chair for a tall or short participant may reduce their discomfort considerably. Ambient temperature should also be comfortable.

5) Testing multiple participants. If several computers are available, consider testing several participants at once. Verbal instructions can be used if all participants start at the same time, but if they do not, maybe have all instructions on the screen. If so, be sure to test those instructions thoroughly beforehand—instructions that were thought to be perfectly clear may not be for the participant population. Another consideration for multiple-participant testing arises when using auditory presentation of stimuli or tones to signal error trials. Participants can easily be confused about where the sounds are coming from; however, headphones can usually avoid that problem.

Is a keyboard the right input device?

Typically, keyboards are used for response collection, usually limiting the allowable keys to those used for responses. However, in many situations, a keyboard may cause problems. Participants can easily be confused about which keys are being used. If working in a darkened room, locating the right keys can be difficult. If participants have to look at the keyboard to locate the keys they need, it can be disastrous in recording reaction times. Especially with children, the temptation to play with the keyboard may be too great. A good alternative is to use a response box with only a limited selection of keys, such as the PST Serial Response Box available from Psychology Software Tools. Custom response boxes can also be made, using the Custom Expansion Kit with the PST Serial Response Box.

The Single-Trial, Reaction-Time Paradigm

An experiment using the single-trial, reaction-time paradigm consists of one or more blocks, or sets, of trials. Each trial consists of the presentation of at least one stimulus, and the collection of the time required for the participant to respond. The trials vary (within or between blocks), with each trial type representing a single level of an IV (or the unique combination of levels of two or more IV's). The principal DV is RT, though accuracy (usually percent error or percent correct) is also examined as a secondary DV. Both RT and accuracy are recorded as the DV's for each trial, but the analysis is usually based on the mean RT (or percent correct) for each trial type, averaged across all correct trials.

The concern for single-trial, reaction time experiments is how various independent variables affect RT—that is, how RT is changed when we deliberately manipulate the stimuli in some way. Inferences about cognition and perception are then made, based on the pattern of RT changes with changes in the independent variable(s). However, RT is also affected by many variables that are not of direct interest. These potentially confounding variables must be controlled in some way so that they do not influence the outcome.

Following a definition of RT, we present a discussion of the events that occur in a typical RT experiment. Then we discuss a number of confounds that can affect RT, and which must be considered in designing experiments employing RT as a dependent variable.

RT defined. For most research in psychology, RT is defined as the time from the onset of a stimulus to the time the participant responds. For computerized experiments, this is usually the time from stimulus onset until a key is pressed indicating a response.

It is important to note that RT can vary, depending on the particular response required. Suppose that there are two versions of an experiment, differing only in how the participants respond to indicate which of the two stimulus types has been seen. In one version, they must press the '1' and '2' keys on the computer keyboard to indicate which type of stimulus appeared. In the other version, they must press a lever to the left or right to indicate the stimulus. Overall RT might well be longer in the case of the lever-press, because the mechanical resistance is higher, or because the distance to be moved is farther, or because different muscles are employed in the two types of responses. In this case, caution is required in comparing the results of the two experiments. Differences in the obtained RT's might be due solely to mechanical factors, and not reflect any differences of interest. Care is needed, then, in comparing the outcomes of experiments using different responses. Whether a relatively fast key-press or a relatively slow lever-press was used will affect overall RT, but in either case, the difference in time to respond to the two types of stimuli may be about the same. In comparing experiments, then, the crucial issue is whether the same pattern of differences in RT's is seen, rather than whether overall RT differed.

While we have defined RT as the time from stimulus onset to a response, it is sometimes defined in other ways. In much research in kinesiology, for example, RT is defined in relation to the onset of a muscle potential (electromyographic signal), while the time from that first electrical activity in the muscle to when the response movement itself is completed is called Motor Time. Because RT is sometimes defined differently, and because it can depend on the nature of the response apparatus, it is important in RT research that the definition of RT and the nature of the response be made explicit, and reported in the Procedures section of the research report.

RT is also sometimes classified as simple RT or choice RT. In simple RT, a participant makes one response to a single stimulus. This requires only a judgement about the presence of a stimulus, and does not involve a decision about the nature of the stimulus. Choice RT is measured when more than one type of stimulus can occur, and the participant must indicate the stimulus type by his or her choice of responses. Because research on simple RT is rare, "RT" means choice RT unless noted otherwise.

General Considerations

In developing the general considerations for RT research, we examine issues concerning the events that take place on each trial, how blocks of trials may differ, and finally how these combine to form the experiment as a whole.

An example experiment

To permit concrete examples of the issues discussed, we begin by outlining an experiment that could be fairly easily implemented in E-Prime 2.0. The intent here is clarity, rather than scientific importance. Suppose that you wish to examine how RT to a stimulus is affected by changes in the location of the stimulus. Visual acuity is best for objects in foveal vision—the small, central part of the visual field, and drops rapidly for objects further away in peripheral vision. But does that affect RT? The following experiment would help answer that question.

The principal dependent variable is RT, with accuracy (percent correct) as a secondary dependent variable. The independent variables are the location of the stimulus and whether it is adjusted in size to compensate for poorer peripheral acuity. The stimulus is a letter, presented in a random location on the screen. Stimulus letters are centered on locations 0, 2, 4, 8, and 16, left and right of straight-ahead. Letter sizes are chosen to compensate for the distance from central vision (reference). The letters to be presented are C, G, O, and Q, with one response for C and O and another for G and Q. These letters were chosen because C and G share a lot of feature overlap, as do O and Q, so the discrimination is fairly difficult. Four different letters are used so that participants cannot rely on a single feature, such as the tail of the Q, for the discrimination.

What happens on each trial?

Typically, RT experiments consist of one or more series (blocks) of trials. While the specific stimulus may vary from trial to trial, certain aspects of the experiment are usually the same on each trial. There is often a fixation mark of some kind, to let the participant know where he or she should be looking when the trial starts. Initiation of a trial may be under the participant's control, allowing the participant to begin a trial whenever he or she is ready. Alternatively, initiation of a trial may be automatic, controlled by the experimenter or computer. In this case, a warning signal is typically given, to allow the participant to get ready for the trial. Sometimes the appearance of the fixation mark acts as the warning, and sometimes a tone or other signal is used. After a trial is initiated (by the participant or automatically), there is usually a brief delay before the stimulus appears. This delay is called the foreperiod, and may vary from trial to trial or be fixed (unvarying). The foreperiod is usually fixed for choice RT tasks.

At the end of the foreperiod, the stimulus is presented. In many experiments there is only a single event making up the overall stimulus. In others, there may be distracter elements displayed on the screen, or stimuli that serve as primes. In either event, timing of the reaction begins when the critical stimulus is displayed. The critical stimulus refers to the element in the display that determines the appropriate reaction (i.e., which key to press). This is sometimes called the "imperative" stimulus. The stimulus duration (how long it remains in view) will largely be controlled by the nature of the stimulus display. For example, if eye movements during the stimulus presentation could affect the experiment, a very brief (say, 100 ms) presentation is often used, since it takes about 200 ms after the stimulus appears for an eye movement to begin. If the stimulus duration is so short that the participant gets only a glance at the stimulus, the display is described as a data-limited display. Other situations involving data-limited displays are discussed below.

Another issue for defining a trial is that of how long to give the participant to respond. Typically, the participant must respond with a key-press within some limited time. The choice of that time depends on the sorts of RT's expected, with the time allowed being set so as to encompass any legitimate trials. If the task is an easy one, with RT on most trials being less than 500 ms, the time allowed for a response may be relatively brief (e.g., two seconds or so). If no response occurs in that time period, the trial is counted as an omission. Many harder tasks, however, have typical RT's of 1-2 seconds. In this case, the time allowed for a response should be increased accordingly.

Feedback about accuracy and/or RT is usually given following a response. Feedback about accuracy is usually provided, telling participants whether they were right or wrong in their choice of a response. It should be noted, though, that participants are generally aware of having made an incorrect response. The accuracy feedback emphasizes the importance of correct responding. Because the usual RT instructions emphasize speed of reactions, RT feedback is important, since it lets participants monitor their own performance. Many researchers prefer not to report RT on error trials, to avoid encouraging participants to respond so quickly that accuracy is reduced.

The inter-trial interval (ITI) is the time from the end of one trial to the beginning of the next. If the participant controls initiation of the next trial, the participant also controls the ITI. When it is important to control ITI, trial initiation must be controlled by the computer or experimenter.

In some experiments, there may be more than just a single stimulus presented on each trial, or there may be a prime and then a stimulus that calls for a response (sometimes called the imperative stimulus). For example, if the participant must judge whether two letters they see are the same or different, they might see one letter and then see the second some short time later. That delay before the second stimulus is the inter-stimulus interval (ISI). The ISI is time from the onset of the first stimulus to the onset of the second. Another term for this is stimulus onset asynchrony (SOA).

In the example experiment on visual acuity, a central fixation mark would be required so that measures of stimulus location would be accurate. Because the locations must be specified and the proper size letters chosen to compensate for distance from fixation, it would be necessary to control the distance from the participant to the screen, using a viewing hood or chin-rest. The distance to the screen, and the resulting display sizes (in degrees of visual angle—see below) should be included in the Methods section of the final report. To be sure that participants do not turn their eyes and re-fixate the letter in central vision, a data-limited display would be needed. A 150 ms display would control for this. Participants might employ a strategy of guessing the location, and thus not be looking at the fixation when the trials begin. This can be prevented by stressing in the instructions that participants should be looking directly at the fixation when they start each trial, and also by randomly presenting the stimuli to left or right of fixation. If participants adopt a guessing strategy, this will lead to them missing many stimuli completely, and the high error rates will clearly show that there is a problem.

Because of the brief displays and the need to guarantee that the participant is looking at the fixation, participant-initiated trials should be used, with a fixed foreperiod. RT's for this task should be fairly fast, so it would probably be appropriate to limit the allowed response time to 2 seconds or less. Accuracy feedback would be used, with RT reported on correct trials only.

What happens within a block of trials?
The entire series of trials making up an experiment is usually divided into blocks of trials. The division may simply reflect time constraints. In a long experiment, it is best to ensure that participants take occasional pauses, so it may be best to break the entire series into shorter blocks, with rest pauses between them. More importantly, the division of the experiment into blocks may be an integral part of the experiment itself. The rest of this section treats that situation.

Blocked versus random presentation
Suppose there are two or more different sorts of trials being presented in an experiment (two or more independent variables, with two or more levels of each). A question to consider is whether these different sorts of trials should be presented together in each block with the various types alternating in random order, or whether the series of trials should be blocked, with all trials of one type presented, followed by all trials of the other.

Compare two versions of the letter-identification experiment. One is the experiment described above, except that participants must respond by pressing one of four keys to indicate either which of four letters is present (four-choice RT). The other is the same, except that only two letters are used (two-choice RT). The two experiments differ only in whether the two types of trials (two- and four-choice) occur randomly (within a single block), or are blocked, with all of the two-choice trials occurring together, and all of the four-choice trials occurring together. In order to directly compare the four-choice RT to the two-choice RT, the two types of trials (two- and four-choice) could occur either randomly (within a single block), or blocked, with all of the two-choice trials occurring together, and all of the four-choice trials occurring together.

In general, we expect that RT will increase with the number of choices (Wickens, 1992). If participants completed one block of two-choice and one block of four-choice, that would probably be the outcome. But with random presentation, that may not be so. Why not? In this experiment, it is likely that random presentation would lead the participants to ignore whether the trial is two- or four-choice. That is, the participants seeing the stimuli in random order may not bother to pay attention to whether the trial involves two choices or four, but rather treat all of the trials as if they involved four possible choices. That would increase the mean RT for two-choice trials, while having no effect on four-choice trials. That is, the results of the experiment depend (in part) on the choice of blocked versus random presentation of the stimulus types.

In general, then, the choice of random or blocked presentation must depend on whether participants given random ordering of trials will adopt different strategies than those given blocked order. In the case of the experiment above, participants in the random-order experiment might adopt the strategy of ignoring whether there were two choices or four, and treat the two-choice trials as if they were four-choice trials. Thus, the blocked version gives us the better estimate of the actual time required for choosing between two and four response choices.

When blocked presentation is used, the issue of counterbalancing of treatment orders is raised. In the blocked version of the two- versus four-response experiment (two levels of one independent variable), half of the participants would do the two-choice trials first, while half would do the four-choice trials first. This counterbalancing is designed to remove (or at least balance) any effects of carry-over from one block of trials to the next.

Certain confounding variables are usually controlled by counterbalancing. One is the mapping of stimuli to responses. If interested in comparing the speed of reactions to the targets 'C' and 'O' to the speed for the targets 'G' and 'Q' in the two-response version of the experiment above, have half of the participants respond by pressing the '1' key for 'C' and 'O' and the '2' key for 'G' and 'Q'. Half would respond in the opposite way, pressing the '2' key for 'C' and 'O.' This controls for any possible difference in RT due to the different responses themselves, and is necessary because some muscular actions take longer than others.

If comparing detection of letters under the conditions in which the letters either were or were not adjusted in size, the comparison of interest is adjusted vs. constant size; therefore, since the '1'- and '2'-response trials will be averaged together, counterbalancing may not be necessary. In other experiments, however, it can be absolutely crucial. Consider, for example, a version of the letter-choice experiment in which two letters are presented and the participant must indicate the letters are the same by pressing one key or that they are different by pressing another. Since one aspect of that experiment is to compare "same" and "different" responses, it would be important to counterbalance the mapping of the response keys to same and different stimuli. Otherwise, a difference between RT to "same" and "different" might be interpreted as reflecting the difference in the stimuli, when it was really reflecting a difference in response time to press the '1' key versus the '2' key. The difference in RT really is due to a lack of proper counterbalancing. (Alternatively, a failure to counterbalance might lead to a finding of no difference, when there really was one.)

Ordering of trials within a block

When each type of trial is presented within a single block of trials, it is almost always the practice to randomize the order of trials. This is equivalent to writing down each trial on a card (including multiple cards for repetitions of the same stimulus) and then shuffling the cards. There is, however, a problem that can be caused by randomization. Suppose that there are two types of trials. In a single block, 100 trials of each type are presented, in a random order. It is likely that some fairly long sequences of a single trial type will occur, with a single type presented 7 or 8 times in a row. Because humans expect randomness to produce shorter sequences than it really does, participants tend to fall into the gambler's fallacy. If a single trial type occurs 6 times in a row, participants will often either decide that the other trial type is "overdue" and expect it, or they will decide that the type they have seen is more likely to occur and expect it again. In either case, if the expectation is correct, the participant will probably be very fast and accurate. If the expectation is wrong, the participant will be slow and error-prone. E-Prime 2.0 permits setting a maximum on the number of repetitions of a single trial type, as a way to avoid this problem.

What happens within the whole experiment?

An experiment is composed of one or more blocks of trials. If the experiment is particularly long, it may be broken down into sessions of one or more blocks each. In that case, counterbalancing of blocks across sessions may also be required. An experiment most often begins with instructions about the nature of the experiment, and some practice trials. When the experiment is concluded, some form of debriefing is often used to show the participant the purpose of the experiment and to permit questions about it. Instructions, practice, and debriefing are considered separately below.

Instructions

The purpose of the instructions, in any experiment, is to let the participant know what will be happening and what the correct responses are. In RT research, instructions should also emphasize that participants are to respond as quickly as possible while still remaining accurate. "Accurate" is typically considered 10% or fewer errors, though this would also depend on the specific experiment.

In long experiments, it is also advisable to instruct participants that they should take occasional breaks. If trials are initiated by the participants, these breaks are under the participants' control. Otherwise, it is a good idea to "build in" breaks by having blocks of trials that are fairly short (e.g., 5-10 minutes). Occasional breaks avoid having the participants just staring at the screen and pressing keys like zombies. This means that participants are less error-prone, and also that RT is less participant to added variability due to eye strain, mental fatigue, and the like.

Practice

Most experiments ask people to do unfamiliar tasks, and require them to indicate their responses by pressing keys that have no previous association with the stimulus. If asked to press the '1' key if a 'C' or 'O' appears and the '2' key if a 'G' or a 'Q' appears, participants must first learn to associate 1 with C and O and 2 with G and Q. At first, participants will be very slow and error-prone in their responses, simply because they have to carefully think about which key to press after they identify the target letter. After a while, participants no longer have to think about which key to press, and their responses become faster and more accurate. For this reason, usually give some practice on the task before actually beginning to collect data. The effect of this practice is to reduce the variability of RT during the experiment itself. The number of practice trials can be determined during pilot testing. It is also a good idea to stand and watch the participant during the practice trials, to be sure they understand the task. You may sometimes need to encourage them to slow down, if they are making many errors. Once they clearly understand the task, encourage them to try to speed up. Presenting the mean accuracy after each trial or block of trials can be useful.

In a short experiment, completed in a single session, one block of practice trials is usually all that is needed. If the experiment extends over several sessions, a brief block of practice trials is usually given at the beginning of each session and the first session is often treated as a practice. If the type of stimulus display or responses change from block to block, it might also be necessary to have practice before each block of trials.

Debriefing

When an experiment is over, it is usual to debrief the participant. The debriefing typically is a simple matter of telling the participant what pattern of RT's is expected to be found and why. That is, the debriefing is used to explain to the participant what the experiment was about. Participants may also be shown their individual results. A second reason for a debriefing is to get comments from the participants about their own experience. While such comments may not be part of the data proper, they can sometimes reveal the use of strategies that the experimenter had not considered, or may even point out flaws in the design. Remember that participants have spent some of their time during the experiment trying to figure out "what is going on." In doing so, they may notice things about the experiment that the experimenter never noticed—including problems.

How many trials?

Why not just have the participant respond once to each type of display, and take that single RT as the "score" for that condition? This would certainly be faster, since few trials would be needed. The problem with using this procedure, however, is that it ignores the large variability in RT that is due to factors other than the independent variables. RT varies from trial to trial, even if the stimulus does not. That variability comes from momentary changes in attention and muscular preparation, among other things. Note that participants cannot pay attention evenly and uniformly for any length of time. Even when you are listening to a fascinating lecture, you will find your attention wandering from time to time. The same thing happens in RT experiments, when the participant sits doing trial after trial. Occasionally, participants will start a trial when their attention is not focused on the task. When this

happens, a very long RT usually results. Occasionally, participants will start a trial when their attention is not focused on the task. When this happens, a very long RT usually results. Long RT's due to inattentiveness would be expected to occur about equally often for all stimulus types, so averaging a few such trials with many others does not create a problem.

Another way to look at the problem of number of trials per condition is to realize that the RT on each trial provides an estimate of that participant's "true" RT for that condition. Each individual estimate is not very reliable, for the reasons given above. Therefore, averaging a number of estimates (RT's on many trials) provides a better (more reliable) estimate of "true" RT. Recall that the confidence interval estimate of a population mean becomes more and more precise as the sample size increases. Similarly, the estimate of true RT becomes better and better as sample size increases--though in this instance, sample size refers to the number of trials per participant, rather than the number of participants. By employing the formula for the confidence interval, determine the number of trials needed to have a certain level of accuracy. In practice, 15-30 trials per condition per participant seem to provide a satisfactory result. This is enough trials that a few aberrant trials will have little effect on the mean RT for that condition.

Between- Versus Within-Participants Designs

Another issue of importance to RT experiments is that of whether the independent variables should be manipulated between participants or within participants. Between-participants variables are ones where different participants are tested on each level of the variable. For the example of two- versus four-choice RT, that would mean that participants do either the two-choice version or the four-choice version, but not both. Within-participants variables are those where each participant is tested at each level of the variable. For the same example, this would mean that each participant does both two- and four-choice trials (in either random or blocked order).

Which method is preferred? We use a different example here, to simplify. Suppose an experimenter wanted to determine the effect of alcohol on RT's to a simple stimulus, and had 20 participants available. He or she could randomly assign 10 participants to perform the task drunk and 10 to perform it sober, then compare those mean RT's. This would be a between-participants design. But why not test each participant both sober and drunk? That way there are 20 participants in each condition. This would be a within-participants design. (Of course, she would want to counterbalance the order, and test some participants sober and then drunk, and others drunk and then sober.) It should be clear that an analysis based on 20 participants per group is more powerful than one based on only 10 participants per group. (Note that the type of statistical analysis would change slightly, since a within-participants design violates the assumption of independent samples. In this case, comparing two means, the t-test for independent samples would be used with the between-participant design, and the t-test for dependent ("correlated", "matched-pairs") samples with the within-participant design. If there were several levels of dosage used, the appropriate test would be the standard ANOVA for the between-participants design, and the repeated-measures ANOVA for the within-participants design.)

The main thing to note about the example above is that a within-participants design is clearly better, if it is appropriate to use it, because it effectively increases sample size. But there are severe limitations to its use as well. A within-participants design works fine in this example because if the experimenter tests participants drunk, then tests them sober a few days later, she can be fairly sure that the only systematic difference in the participants is in whether or not they are sober. Similarly, when comparing RT to two versus four stimuli, making a choice between two stimuli probably does not have a later effect on making a choice between four stimuli (or vice-versa)—at least if the trials were blocked. But in many situations making the assumption that there is no carry-over from one condition to another is not justified. For example, to compare RT to naming meaningless shapes following two different types of training, a between-participants design is needed because if a participant learns something by one method that learning cannot be "erased." If participants performed faster following the second round of learning, is it because that method of learning is better? Or is the difference simply due to the added learning? Another situation in which a between-participants design is required is when the variable is "attached" to the person, and cannot be experimentally manipulated. Variables of this kind include sex, race, ethnic background, and religion.

In general, then, within-participants designs are to be preferred if it is reasonable to assume that there are no carry-over effects of one level of an independent variable on performance at other levels of that independent variable. If that assumption is not reasonable, a between-participants design should be used. Note that this is similar to the issue of random order of trials versus blocking by trial type—if encountering one level of a variable might induce a strategy that carries over to another level, the levels should be blocked, when using a within-participants design. If blocking will not solve the problem, a between-participants design will be necessary.

Another way of thinking about when to use a within-participants design is to consider whether the effect of the experimental "treatment" or manipulation wears off. If the passage of time will erase the effects of the manipulation, a within-participants design may be appropriate. An old joke illustrates the point. A lady walking down the street saw a man lying drunk in the gutter. "Sir," she said, in obvious disgust, "You are drunk!" Opening one eye the man replied, "Yes, madam, and you are ugly. And tomorrow, I shall be sober." Some treatments wear off, and thus are candidates for within-participant manipulation.

There are also some experiments that employ both within- and between-participants independent variables. These are usually referred to as mixed designs. For example, to compare the patterns of RT's for males and females in the letter-identification experiment, sex would be added as another independent variable, in addition to location and whether the letter size was adjusted to compensate for distance from central vision. Location and adjustment would be within-participants variables. But sex (male vs. female) would be a between-participants variable, since no participant could be in both groups.

Other Considerations in RT Research

A number of other factors that must be considered in designing research employing RT as a dependent variable are discussed below. Wickens (1992, Chapter 8) provides a more detailed account of most of these same issues.

Speed-accuracy trade-off

In research employing RT as the dependent variable, the interest is usually in showing that RT differs for different levels of the IV(s). A serious problem can arise, however, if the conditions associated with faster RT also have higher error rates. Such a situation is called a speed-accuracy trade-off, because the participants may be sacrificing (trading) lower accuracy for greater speed. That is, they may be faster on those trials because they are pushing themselves for speed, but ignoring the higher error rate that often goes with that effort. Consider the comparison of RT's in the letter-identification task.

Suppose that no differences in RT were found with increased distance from foveal vision, in contrast to the expected finding of an increase in RT to identify letters seen less clearly. If the error rates were seen to be increasing with incremental difference, this would suggest that participants were trading accuracy for speed—in order to maintain the same speed of response under more difficult conditions, the participants were permitting the error rates to climb.

Fortunately, in most RT research a speed-accuracy trade-off does not occur. In fact, most of the time the fastest conditions will have the lowest error rates, while the longest RT's will come in conditions with the highest error rates. In this case, difficult stimuli lead to both slow and sloppy responses. In any case, it is a wise practice to examine error rates for evidence of a speed-accuracy trade-off. To avoid this problem, instructions to the participants usually stress that they must be as fast as they can in each condition but without sacrificing accuracy. That is, the error rates should be uniformly low for all conditions.

Stimulus-response compatibility

In most RT research, the connection between the stimulus and the response is arbitrary. Participants may be instructed to press '<' for an S and '>' for an H, or '>' for an S and '<' for an H. But occasionally the mapping is not arbitrary. Consider the same experiment, but using L and R as stimuli, instead of S and H. If participants had to press '<' for an R and '>' for an L, for example, they might be both slower and more error-prone than otherwise, because of the association of L with "left" and R with "right." Making a "left" response to an R might well produce some response competition, resulting in a slowing of RT. Basically, any time a stimulus implies a certain direction of response (such as L and R implying left and right responses), there are potential problems of S-R compatibility.

Probability of a stimulus

In most experiments with RT as a dependent variable, each type of stimulus is presented equally often. In this way, participants are discouraged from guessing, since each stimulus is equally likely on each trial. Sometimes, however, one stimulus may be presented more often than another and can have major effects on RT (and error rate). In general, the most common stimulus is responded to more quickly and more accurately. Why is this so? Suppose that in the experiment on recognizing S and H the participants were presented an H 80% of the time, and an S 20%. Participants would quickly realize this, and would expect an H most of the time. On any trial, if the target is an H, there is likely to be a faster response. But if the target is an S, the participants must overcome their expectancy, and preparation for an H. The result is a slower response, and a higher probability of error.

Because of these considerations, it is best to always have the different trial types equally likely whenever randomization is used. Unequal stimulus probabilities are best avoided, unless they form part of the research itself.

Number of different responses

RT increases as the number of possible responses increases. This relationship has long been known, and was quantified in the early 1950's, when Hick and Hyman, working independently, each noted that RT increases linearly with the logarithm (base 2) of the number of alternatives. That means that additional alternatives will increase RT, but the effect of that increase is smaller as the number of responses becomes larger. This effect is not usually of much concern, but must be kept in mind when comparing the results of several experiments (i.e., if they used different numbers of response alternatives, the RT's cannot be directly compared).

Intensity and contrast

At least for low levels of illumination, the more intense the stimulus, the faster the RT. Once the stimulus reaches an intensity where it is clearly visible, however, further increases will have little effect. Similarly, increasing contrast (the difference in intensity between the stimulus and the background) will decrease RT, up to a point where the stimulus is clearly visible. Either low intensity or low contrast would produce a data-limited display. A very brief stimulus is another example of a data-limited display.

One common problem in controlling intensity and contrast is ambient light (the light present in the room). A display that may seem very weak under ordinary room lighting may seem quite bright when room lights are off and windows covered. In experiments employing brief, data-limited stimulus displays, it is important that ambient light be carefully controlled.

In addition to lowering apparent intensity and contrast, ambient light may result in glare or reflections on the display screen of the computer. In this case, lights must be shielded or the computer moved to prevent such interference.

Stimulus location

The location of the stimulus can have a powerful effect on both RT and error rates. Visual acuity drops quickly as stimuli are moved away from the fovea—the narrow area of vision straight ahead that is about 2° wide. A person with 20/20 vision in the fovea will typically have about 20/80 vision 2.5° from straight-ahead. At 10° from straight ahead most people have worse than 20/300 vision. To put this in perspective, at a viewing distance of 57 cm (22.5"), each centimeter is about 1° of visual angle, so a letter displayed 2.5 cm (about 1") from fixation will be seen quite poorly.

For these reasons, retinal locus (where on the retina the image of the stimulus falls) must be controlled by randomization or counterbalancing if the stimuli are not all presented in the same location. If one type of stimulus is presented in the fovea, and another in the periphery, differences in RT might occur (or fail to occur). However they could be due to differences in the location of the stimuli, rather than to differences in the stimuli themselves.

Note that the relative size of the stimuli is a function of distance from the eye. If the relative size of stimuli is a concern, then the location of the participant's head relative to the screen must also be controlled. This is often done by use of a chin rest or viewing hood to keep the participant's head relatively stable. In this case, the viewing distance should be specified in the Method section of the final report. Sizes of stimuli are also reported, in degrees of visual angle, rather than millimeters or inches.

Calculation of stimulus sizes in degrees of visual angle can be done using the law of cosines. A good approximation is obtained by the formula

Size in degrees of visual angle = 57.3W/D

…where W = width of display and D = viewing distance, with W and D in the same units of measurement.

Statistical Analysis of RT Data

While this brief review of single-trial RT research cannot include an extensive discussion of data analysis, a few points deserve comment.

The typical analysis of single-trial RT data employs the analysis of variance (ANOVA) to compare mean RT's under various treatment conditions as defined by the levels of the independent variables. For within-participants variables, a repeated-measures ANOVA is employed. Sometimes, both within- and between-participants factors occur in the same experiment, resulting in a mixed ANOVA. For the example experiment on letter identification, there are two independent variables that define types of trials for the analysis. One is the location of the stimulus, which has six levels (0, 1, 2, 4, 8, and 16°). The other is whether or not it was adjusted in size to correct for poor acuity, which has two levels (adjusted or not). For this analysis, the mean RT for each of the twelve conditions would be calculated for each participant, and those values would serve as the data. The ANOVA then compares the means of those values based on all participants to determine statistical significance[2].

In addition to an analysis of RT's, there should be a parallel analysis of error rates, expressed either as percent correct or as percent error. (Since percent error is just 100 minus the percent correct, these analyses yield the same result.) In general, error rates should parallel RT's—faster conditions have lower error rates. If faster RT's are associated with higher error rates, a speed-accuracy trade-off should be suspected, and interpretation of RT differences should only be made with extreme caution.

---

[2]A discussion of the controversy surrounding the merits of traditional null hypothesis testing is beyond the scope of this discussion. See several articles in the January, 1997 issue of Psychological Science and Chow (1998) for discussions of this topic.

In almost all instances, RT analyses are based on correct trials only. It is good practice to examine the overall error rate for each participant. While what constitutes an acceptable rate will differ with different experiments, it is common practice to delete the data from any participant whose error rates are clearly higher than the norm. In this case, it is likely that the participant either misunderstood the instructions or was simply unable to perform the task. If at all possible, the maximum error rate should be set in advance, so that there is no danger of deleting a participant's data because they do not conform to the expected results. Pilot testing should help in setting a maximum error rate.

Another issue for analysis of RT data involves outliers, or extremely deviant RT's that occur on occasional trials. These usually involve extremely slow RT's. Many researchers assume that such extreme RT's reflect momentary inattention or confusion, therefore they are properly omitted from the analysis, prior to calculating the mean RT by condition for individual participants. A common criterion is to omit any trials whose RT is more than three standard deviations from the mean for that condition. That can be done either based on the mean and standard deviation of RT for all participants, or for individual participants. The latter is clearly indicated if there are large differences in RT between participants. More sophisticated schemes for treating outliers have been suggested (Ratliff, 1993; Ulrich & Miller, 1994).

The repeated-measures ANOVA, which is almost always used for significance testing with RT data, makes an assumption of "compound symmetry," or equality of covariances across all pairs of conditions. That assumption is seldom met in real data. Most statistical packages compute adjusted values of p based on either the Greenhouse-Geiser statistic or the newer, less conservative Huynh-Feldt statistic. In general, these corrected values of p should be used in assessing statistical significance.

Appendix C: Sample Experiments

The E-Prime 2.0 installation includes sample experiments illustrating some of the essential elements with E-Studio:

BasicRT – A basic RT task presenting text.
PictureRT – A basic RT task presenting bitmaps.
SoundRT – A basic RT task presenting text and audio stimuli simultaneously.
NestingRT – A basic RT task using the nesting feature of the List object.
SlideRT – A basic RT task using the Slide object for stimulus presentation.
NestingXRT – A modification of the NestingRT experiment illustrating extended input.
MovieRT– A basic RT task presenting movies.
MultipleDisplayRT– A modification to BasicRT illustrating the use of multiple displays
These samples (*.es2 and associated files) may be found in the Samples folder in the E-Prime 2.0 installation folder tree (default installation ...My Experiments\Samples).

Basic Reaction Time: Text

The BasicRT experiment illustrates a simple reaction time experiment presenting a fixation and a text stimulus, and collecting a response to the stimulus. The BasicRT experiment illustrates the use of the List object, the TextDisplay object, the Procedure object, and the FeedbackDisplay object.
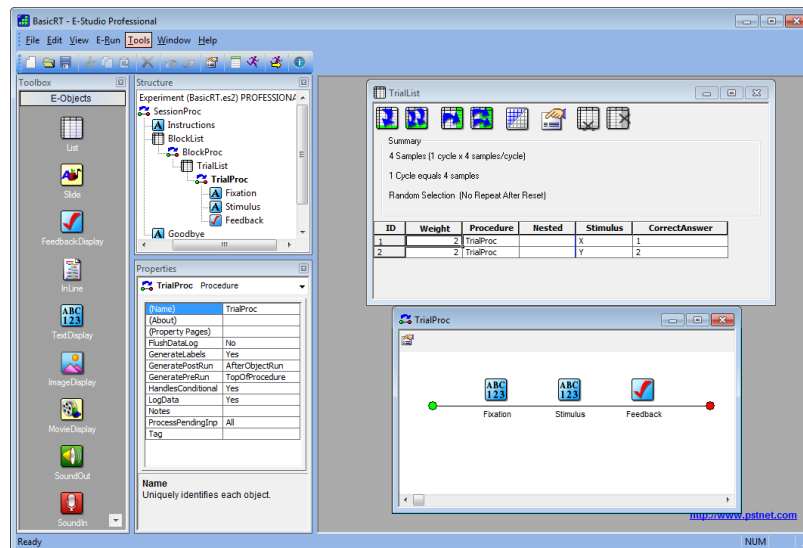
The TrialList defines the variables to be used within the experiment as attributes, the levels for the attributes, and the number of repetitions for each of the levels (i.e., set in the Weight attribute). The properties for the TrialList are set to randomly select a value for Stimulus out of the available exemplars, and to reset after all of the exemplars have been chosen.
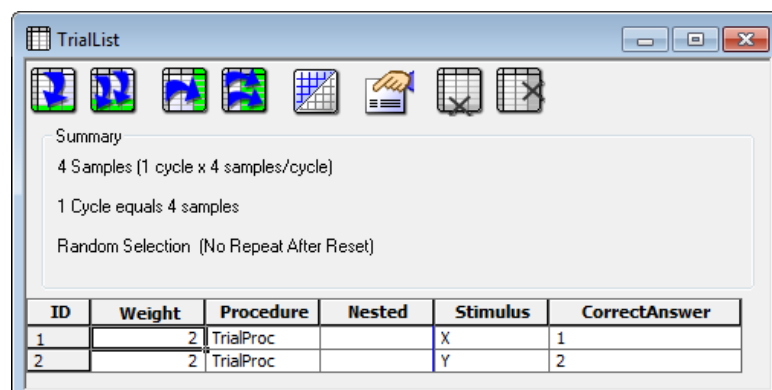

ID	Weight	Procedure	Nested	Stimulus	CorrectAnswer
1	2	TrialProc		X	1
2	2	TrialProc		Y	2

Figure 1. TrialList defines the Stimulus and CorrectAnswer attributes.

The trial Procedure (TrialProc) consists of a TextDisplay object (i.e., Fixation) used to present a fixation prior to the stimulus display, a TextDisplay object (i.e., Stimulus) used to present an "X" or a "Y", and a FeedbackDisplay object (i.e., Feedback) to present the appropriate feedback (i.e., correct, incorrect, etc.).
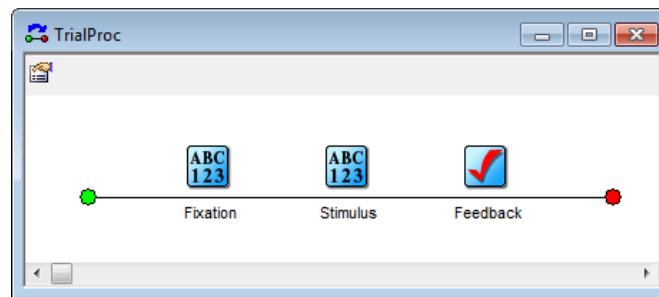
At runtime, the Stimulus object refers to the TrialList in order to resolve the value for the Stimulus attribute and the CorrectAnswer attribute.
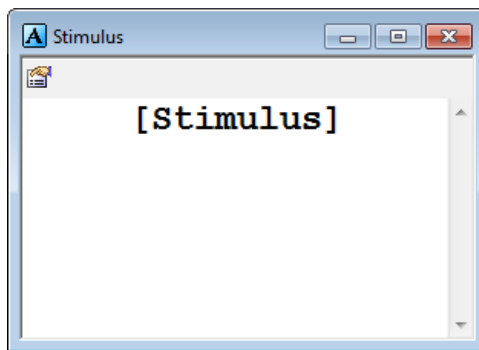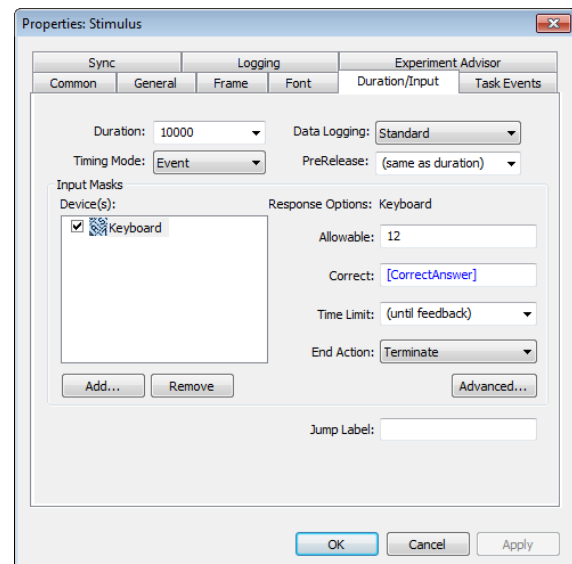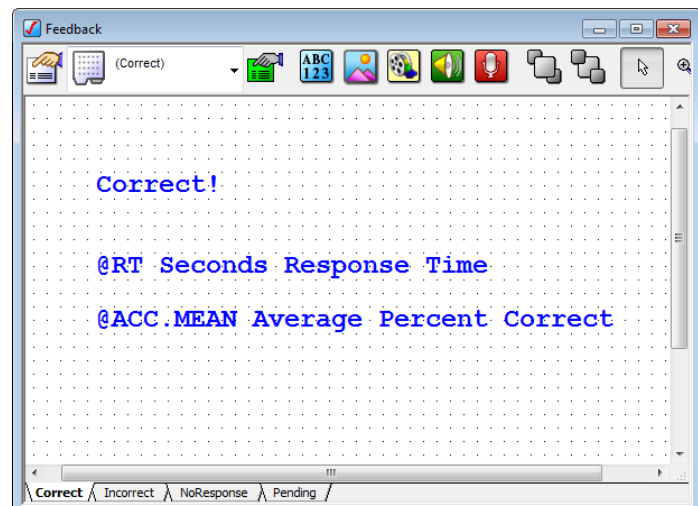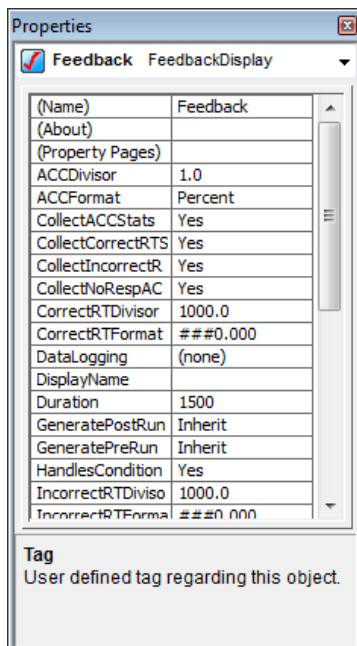
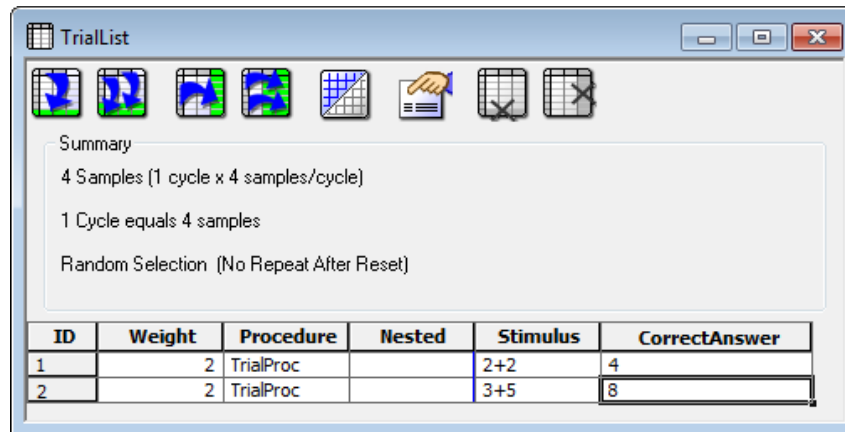*Figure 2. The bracket notation used to refer to the Stimulus attribute in the TrialList object.*

The FeedbackDisplay object named "Feedback" in the TrialProc organizes the feedback to be presented in relation to the input collected by the Stimulus object.
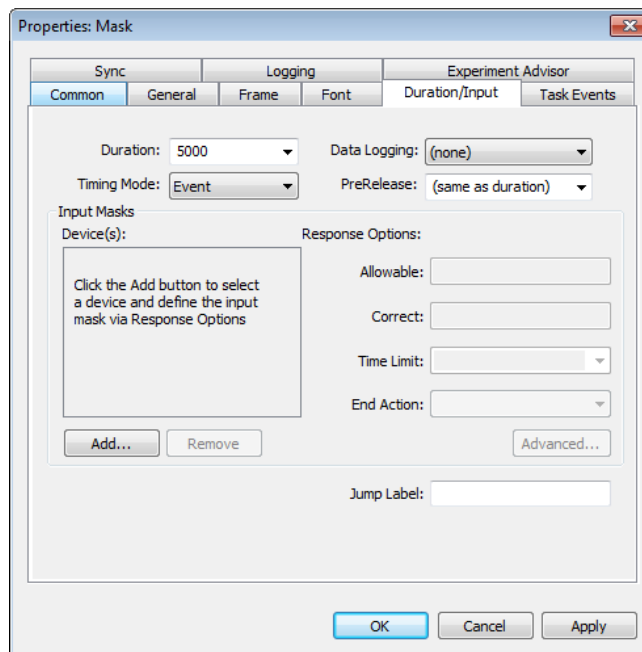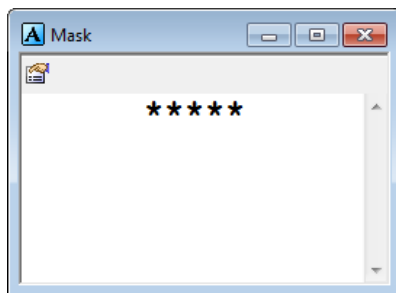
Change the stimulus

The value of the Stimulus attribute may be changed by opening the TrialList object and modifying the values in the Stimulus column. For example, the Stimulus might be changed to enter a mathematical equation (e.g., 2 + 2). The value for CorrectAnswer may then be changed to reflect the correct answer to the equation.

TrialList

Summary

4 Samples (1 cycle x 4 samples/cycle)

1 Cycle equals 4 samples

Random Selection (No Repeat After Reset)

| ID	Weight	Procedure	Nested	Stimulus	CorrectAnswer
1	2	TrialProc		2+2	4
2	2	TrialProc		3+5	8

Add a mask

To add a mask to be drawn over the Stimulus, insert a TextDisplay object in the TrialProc timeline after the Stimulus object. Enter the Mask text and set the duration in the properties for the new object.

Mask

★ ★ ★ ★ ★

Properties: Mask

Sync		Logging		Experiment Advisor	
Common	General	Frame	Font	Duration/Input	Task Events

Duration: 5000   Data Logging: (none)

Timing Mode: Event   PreRelease: (same as duration)

Input Masks

Device(s):   Response Options:

Click the Add button to select a device and define the input mask via Response Options

Allowable:

Correct:

Time Limit:

End Action:

Add...   Remove   Advanced...

Jump Label:

OK   Cancel   Apply

In order to collect input during both the Stimulus and Mask presentations, extended input must be enabled. Refer to the Basic Reaction Time Example: Extended Input section of this appendix for more information.

Add a prime

To add a prime to be presented prior to the stimulus, insert a TextDisplay object in the TrialProc timeline before the Stimulus object. Enter the Prime text and set the duration in the Properties for the new object. As with the stimulus presentation, it is likely that the Prime will vary on each trial. Therefore, it would be useful to create a new attribute in the TrialList to define the values used for Prime presentation.
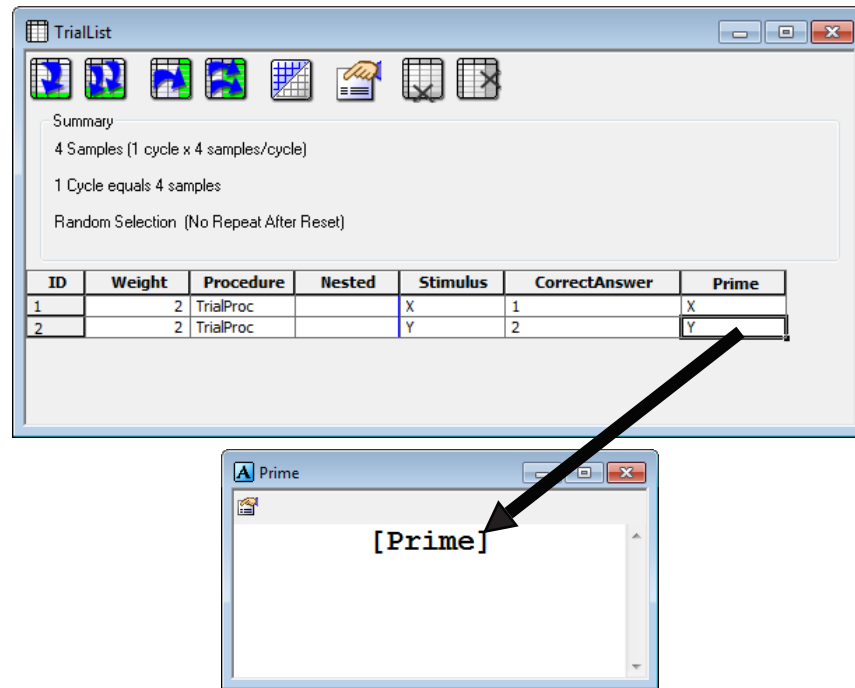
Basic Reaction Time: Pictures

The PictureRT experiment illustrates a simple reaction time experiment, presenting a fixation and a stimulus picture, and collecting a response to the stimulus. The PictureRT experiment illustrates the use of the List object, the ImageDisplay object, the Procedure object, and the FeedbackDisplay object.
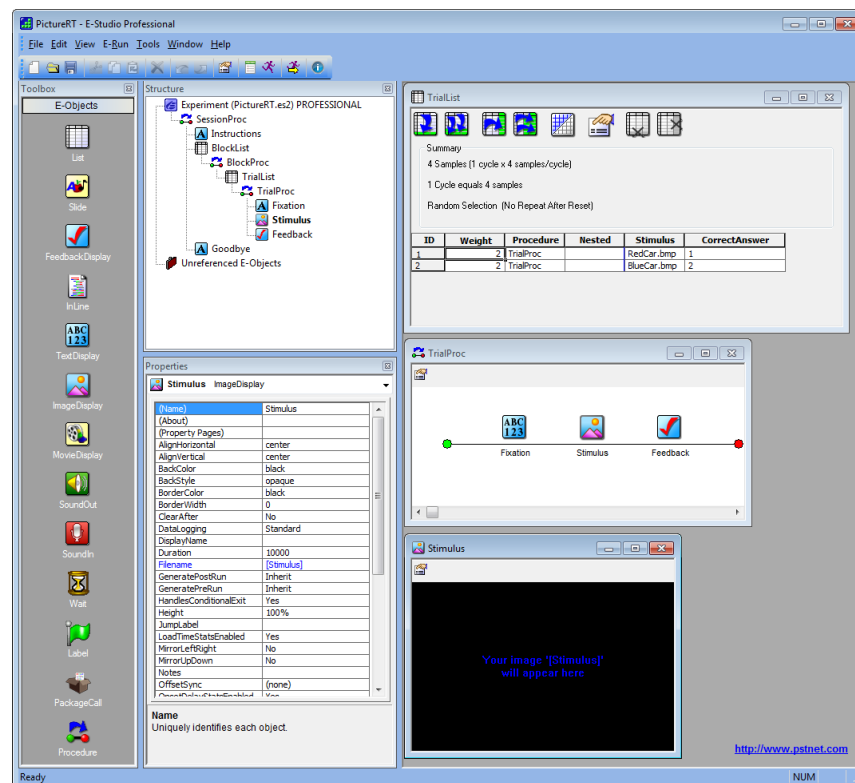

The TrialList object defines the variables to be used within the experiment as attributes, the levels for the attributes, and the number of repetitions for each of the levels (i.e., set in the Weight attribute). The properties for the TrialList are set to randomly select a value for Stimulus out of the available exemplars, and to reset the list after all of the exemplars have been chosen.
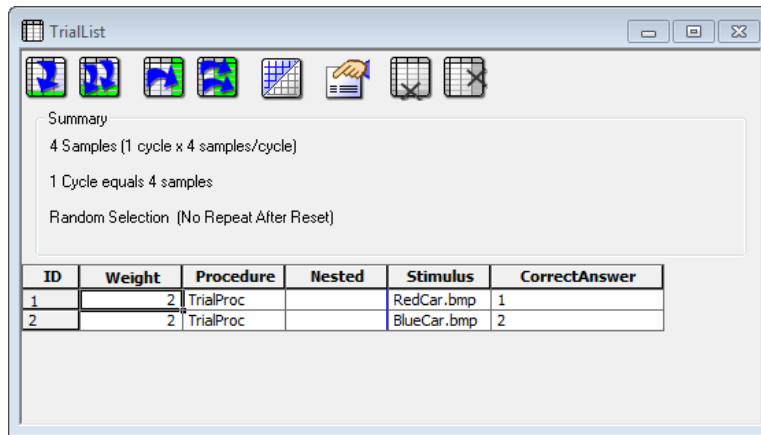
The trial Procedure (TrialProc) consists of a TextDisplay object (i.e., Fixation) used to present a fixation prior to the stimulus display, an ImageDisplay object (i.e., Stimulus) used to present a picture of a red car or a blue car, and a FeedbackDisplay object (i.e., Feedback) used to present the appropriate feedback (i.e., correct, incorrect, etc.).
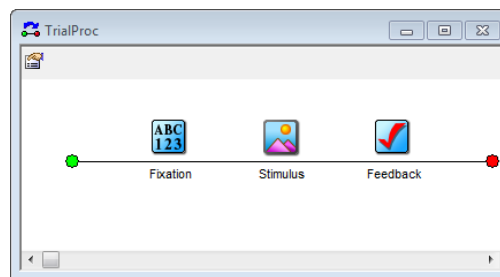
At runtime, the Stimulus object refers to the TrialList object in order to resolve the value for Stimulus and CorrectAnswer.
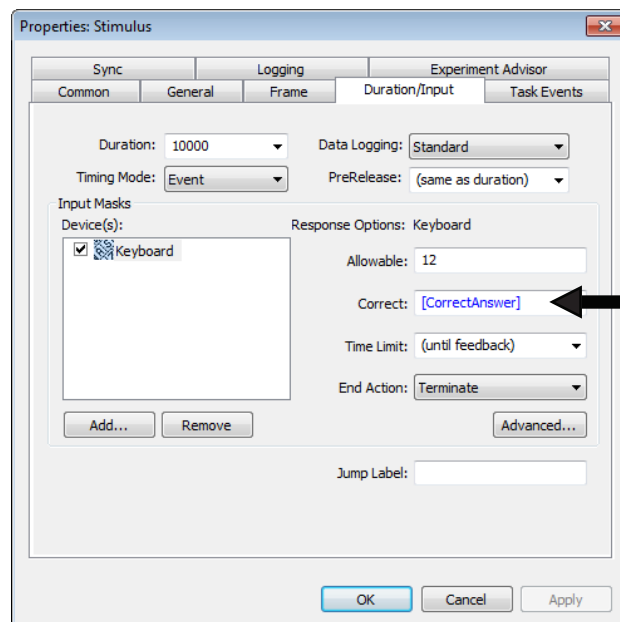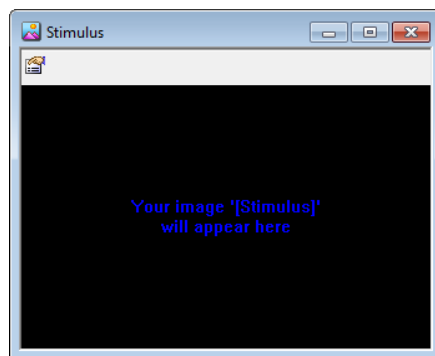
The Feedback object organizes the feedback to be presented relevant to the input collected by the Stimulus object.
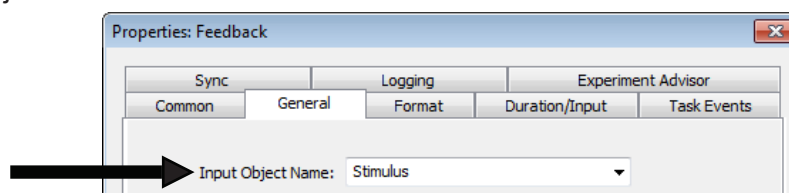
Change the stimulus

The value of the Stimulus attribute may be changed by opening the TrialList and modifying the values in the Stimulus column (names of the image files).

Present multiple pictures, or a combination of text and pictures

The ImageDisplay object is ideal for single picture presentation. Combinations of pictures and text may be displayed simultaneously by using the Slide object. The image below illustrates a Slide object composed of two SlideImage sub-objects and a SlideText sub-object. Refer to the Basic Reaction Time Example: Slide section of this appendix for more information.
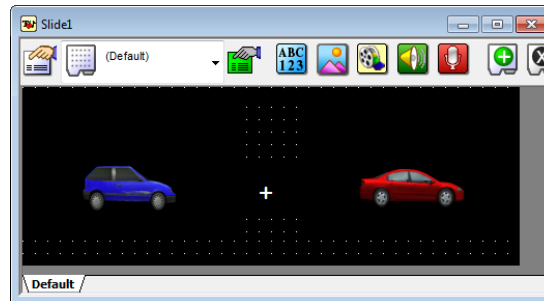
Basic Reaction Time: Sound

The SoundRT experiment illustrates a simple reaction time experiment presenting a fixation, using a Slide object to present audio and text stimuli simultaneously, and collecting a response to the stimuli. The audio file to be played varies on each trial. The names of the files are defined in the SoundFile attribute in the TrialList. Refer to "Advanced Tutorial 2" in the *E-Prime Getting Started Guide* for the full structure. The SlideRT experiment also illustrates the use of the SlideSoundOut and SlideText sub-objects on the Slide.

The SoundFile attribute is used in the Filename field of the properties for the SlideSoundOut sub-object. At runtime, the SlideSoundOut refers to the TrialList object in order to resolve the value for SoundFile.
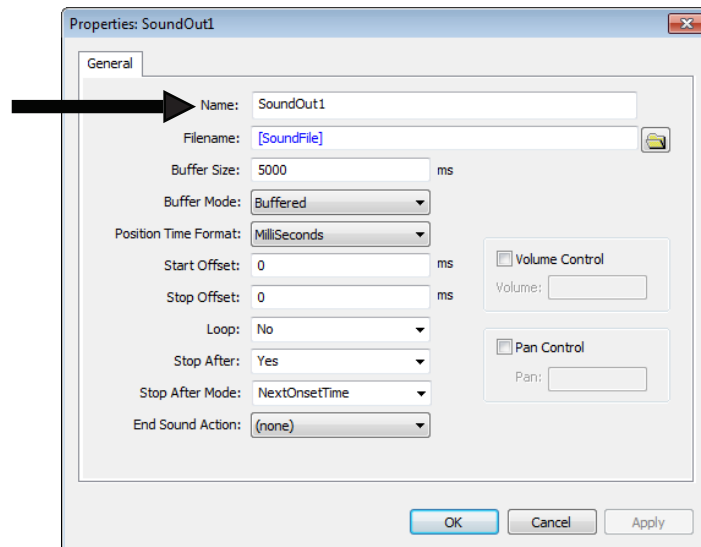
The Sound device is enabled via the Devices tab in the Properties pages for the Experiment Object. The properties for the Sound device may be edited by selecting the Sound device, and clicking the Edit button. The format for the creation of the audio buffer must be set globally via the Experiment Object. All audio files must be saved in the set format.
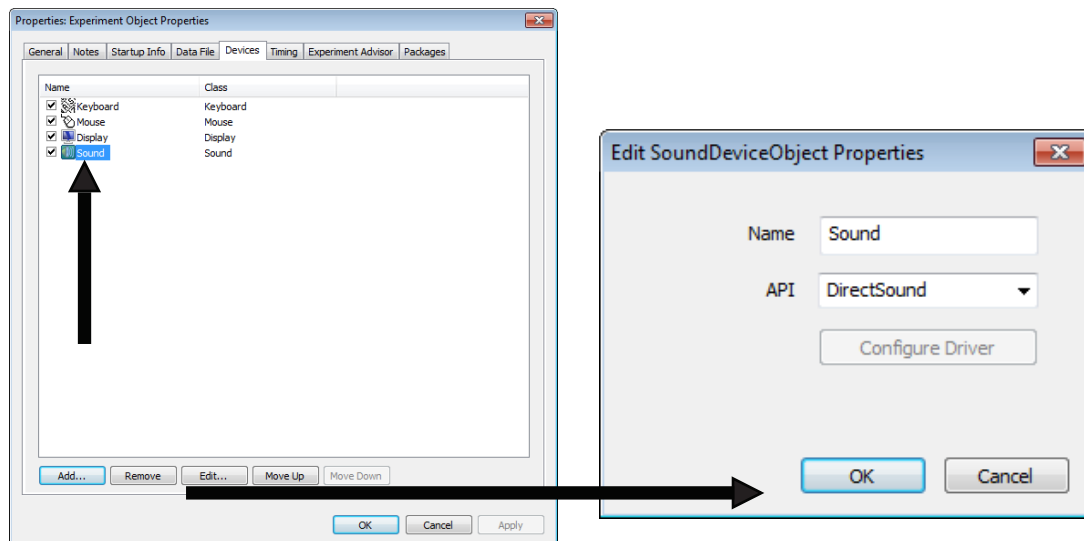
Change the stimulus/sound

The visual or audio stimuli may be changed by opening the TrialList and modifying the values for the Stimulus and SoundFile attributes.

Stop/Continue playback after response

To stop playback upon input from the participant, choose "Terminate" in the EndAction field in the SlideObject (i.e., Stimulus) Property pages. Then select "Yes" in the StopAfter field of the property pages of the SoundOut. To continue the playback after a response, choose the "No" option in the StopAfter field of the SlideSoundOut.

Basic Reaction Time: nested lists

The NestingRT experiment illustrates a simple reaction time experiment presenting a fixation and a text stimulus, and collecting a response to the stimulus. The NestingRT experiment illustrates the use of nested lists objects and the colon syntax to sample stimuli from more than one List during the same trial, and sample multiple stimuli from the same List during a single trial.

The NestingRT experiment illustrates a simple reaction time experiment presenting a fixation and a text stimulus, and collecting a response to the stimulus. The NestingRT experiment illustrates the use of nested list objects and the colon syntax to sample stimuli from more than one List during the same trial, and sample multiple stimuli from the same List during a single trial.
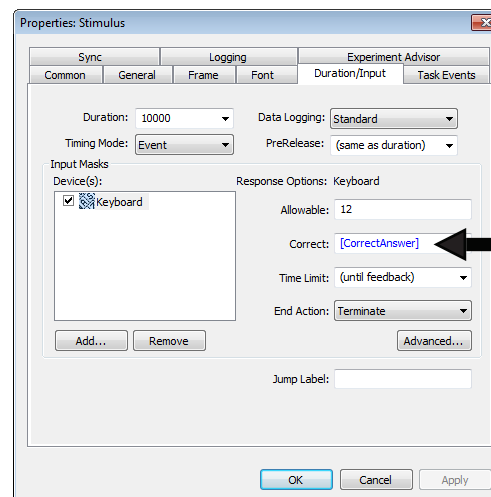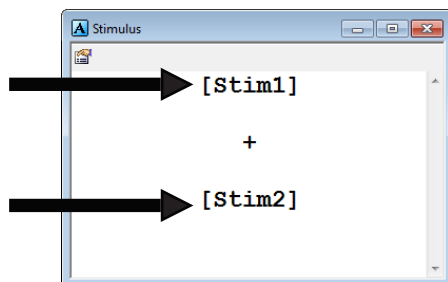
The Nested attribute in the TrialList defines the List objects from which the exemplars will be sampled in order to resolve the values for Word and NonWord in the Stim1 and Stim2 columns. The colon syntax (e.g., [Word:0], [Word:1], etc.) directs E-Prime 2.0 to sample a specific number of times from the appropriate List. The colon syntax default value is 0, which samples one time from the designated List. The integer value following the colon indicates the number of additional samples from the List (e.g., [Word:1] would result in two exemplars being sampled from the List.

ID	Weight	Procedure	Nested	Stim1	Stim2	CorrectAnswer
1	1	TrialProc	WordList	[Word:0]	[Word:1]	1
2	1	TrialProc	WordList, NonWordList	[Word:0]	[NonWord:0]	2
3	1	TrialProc	WordList, NonWordList	[NonWord:0]	[Word:0]	2
4	1	TrialProc	NonWordList	[NonWord:0]	[NonWord:1]	2

Change the stimulus

The value of the Stim1 and Stim2 Attributes may be changed by modifying the Nested lists (WordList, NonWordList).

Populate Lists from text files

Rather than entering values directly into a List object, stimuli may be entered into tab delimited text files and read into the List at runtime. This is accomplished by setting the LoadMethod property for the List object to "File", and specifying the name of the text file as the Filename property for the List object. The file must be in tab delimited, ASCII file format, and should contain the attribute header data (including the system attributes Weight, Procedure, and Nested).

Basic Reaction Time: Extended Input

The NestingXRT experiment is a modification of the NestingRT experiment illustrating extended input. A fixation is displayed, followed by two brief stimulus presentations (i.e., 500 ms) and a mask. Finally, the mask is cleared and a response is collected in response to whether or not the stimuli were both words. The NestingXRT experiment illustrates the use of nested list objects and the colon syntax to sample stimuli from more than one List during the same trial, and sample multiple stimuli from the same List during a single trial. Refer to NestingRT (above) for a description of nesting and the colon syntax. NestingXRT also illustrates the collection of input extended over several objects (i.e., two stimulus presentations and a mask presentation).

The TrialList defines the variables to be used within the experiment as attributes, and the levels for the attributes. The NestingXRT experiment uses the Stim1 and Stim2 attributes to display words and nonwords simultaneously. The CorrectAnswer attribute is used to score the input collected in response to the Stimulus display, and provide feedback via the Feedback object.
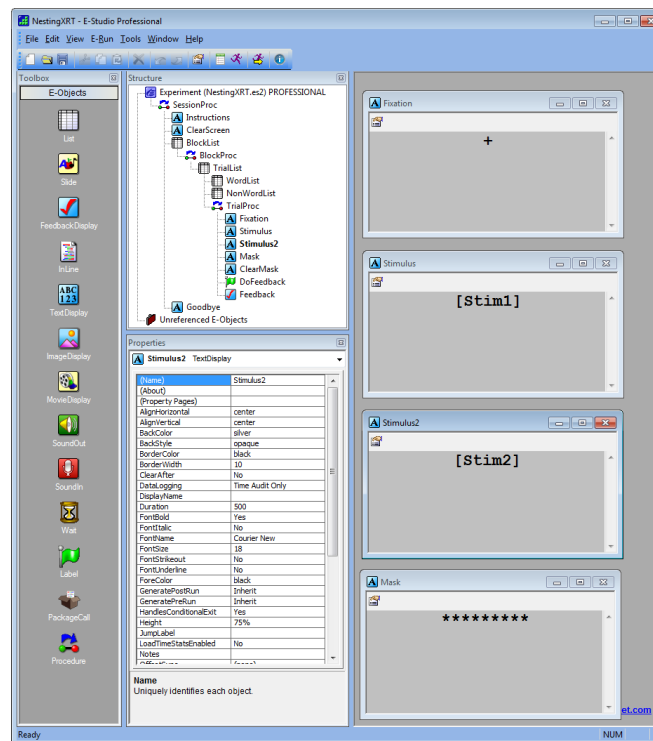
Input is enabled by the Stimulus object, and continues through the presentations of the second stimulus and the mask (i.e., the Stimulus2, Mask, and ClearMask objects). While the Duration field determines the duration of the Run method for the object, the Time Limit field sets the amount of time allowed for input. The Stimulus object's Run duration is set to 500 ms. Thus, the Stimulus object will run for 500 ms before processing moves on to the Stimulus2 object. The Time Limit setting of 3000 ms allows up to 3 seconds for a response (i.e., to continue collecting input through the entire trial presentation).
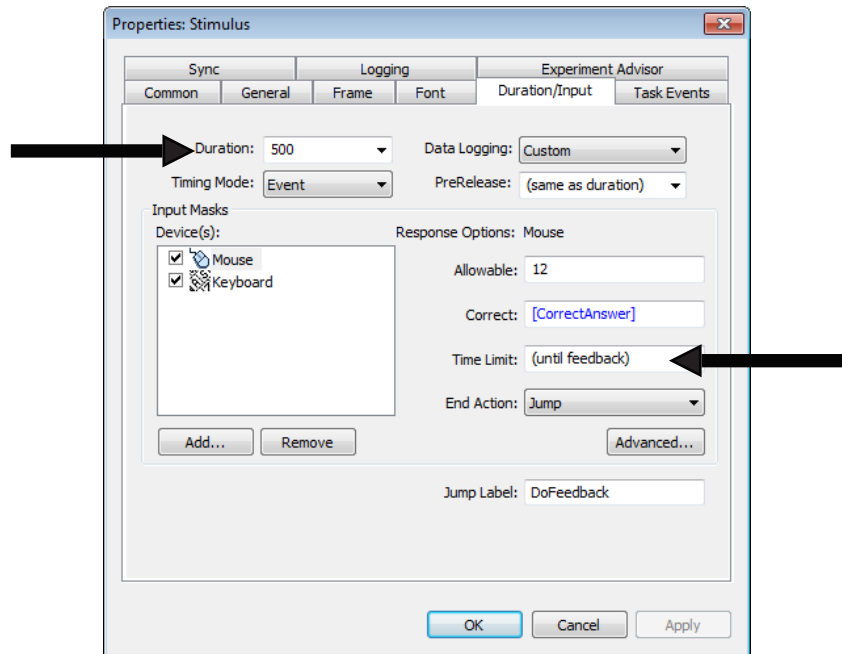
The setting for the End Action field determines the action to be taken upon input from the participant. The Jump option indicates that the program execution will jump to the Label object named DoFeedback in the current Procedure (see Structure window above) when input is received, skipping the other objects in the procedure if they have not yet been run (e.g., Mask).

Change the stimulus

The value of the Stim1 and Stim2 Attributes may be changed by modifying the nested list objects (WordList, NonWordList).

Add a stimulus

To add a stimulus presentation to the Procedure, double-click the TrialProc object to open it in the Workspace. Click and drag a TextDisplay object (or other appropriate display object) to the Procedure, and place the new object where the additional stimulus is to appear in the timeline of events. Set the properties for the new object.

Vary the Mask

To vary the mask presented on each trial (e.g., vary the number of asterisks based on word length), enter an attribute on the TrialList to hold the value of the mask. Then, on the Mask object, reference this new attribute (e.g., [Mask]) in the Text field, rather than the current mask ("*********"). Alternatively, the Mask attribute may be entered as an attribute on the nested list objects.
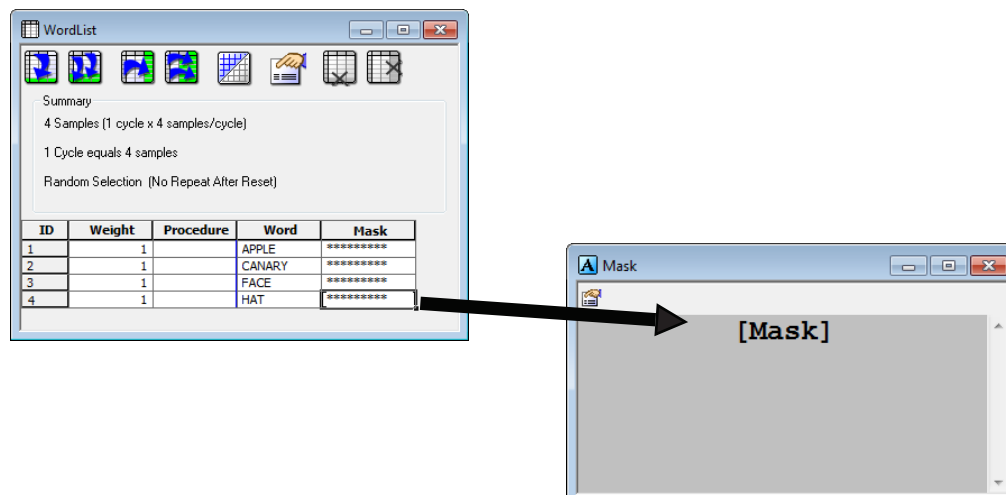
Basic Reaction Time: Slide

The SlideRT experiment illustrates a simple reaction time experiment, presenting a fixation and a stimulus, and collecting a response to the stimulus. The stimulus display consists of an arrow and an asterisk, and the task is to determine whether or not the arrow direction corresponds to the asterisk position. The SlideRT experiment illustrates the use of the List object, the Slide object, the Procedure object, and the FeedbackDisplay object. The SlideRT experiment also illustrates the use of the SlideImage and SlideText sub-objects on the Slide, as well as how to use SlideState objects.
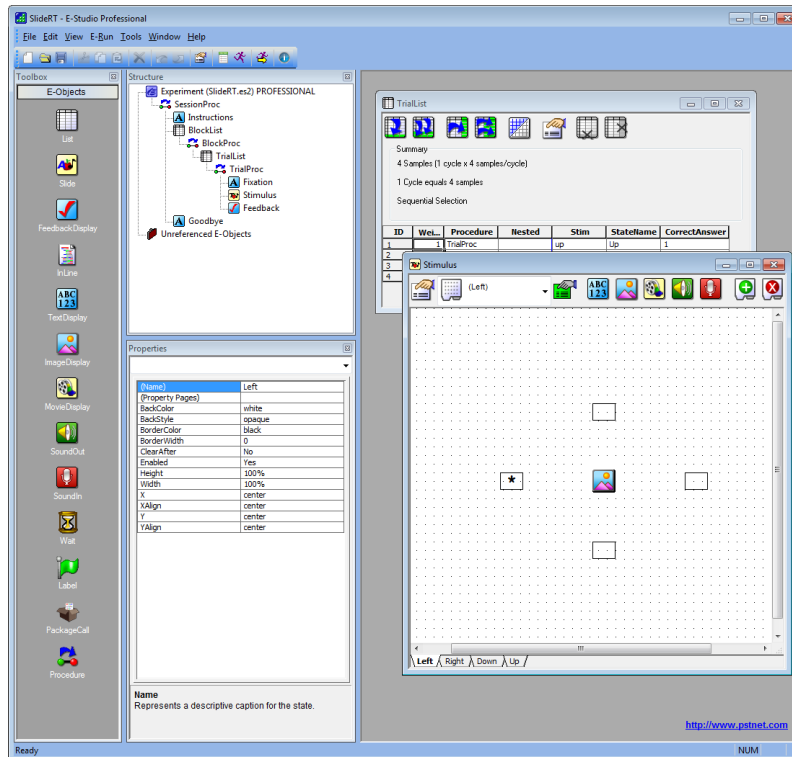
The TrialList defines the variables to be used within the experiment as attributes, the levels for the attributes, and the number of repetitions for the levels (i.e., the Weight attribute). The SlideRT experiment uses the StateName attribute to choose from four different SlideState objects on each trial. Each SlideState is created by using the Add State tool on the Slide tool bar, and is represented as a different tab within the Slide object.
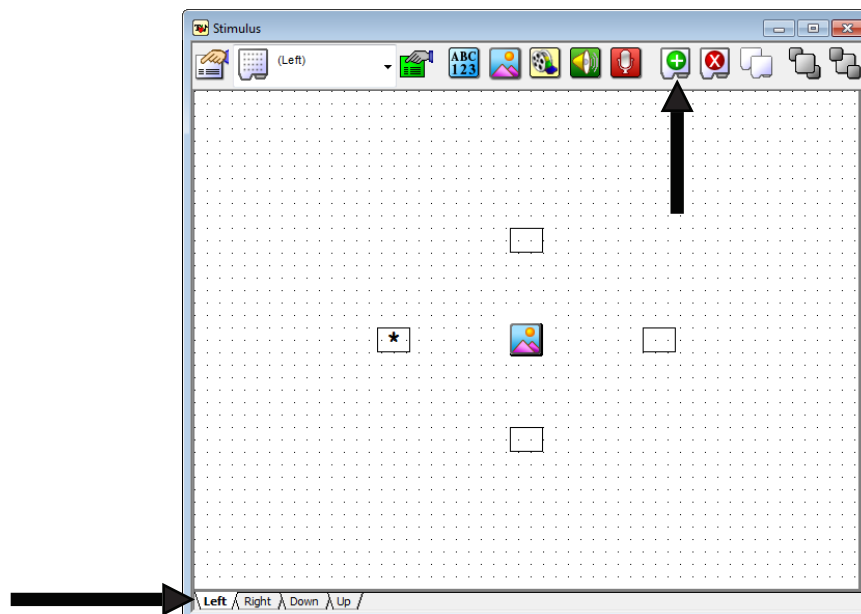
Each of the SlideState objects shows an asterisk in one of the possible positions. Each has a unique name, and is listed as level in the StateName attribute in the TrialList. In the Property pages for the Slide object, the Slide calls one of the four SlideState objects via its ActiveState property. The ActiveState property determines which SlideState will be presented on each trial.
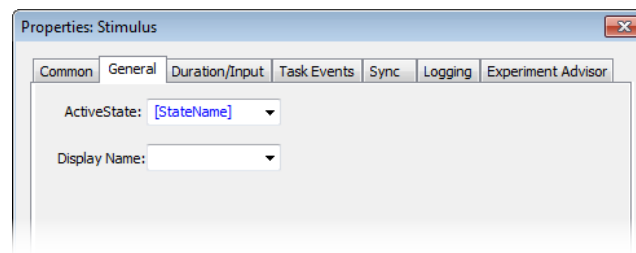
The Stimulus attribute determines the image to be displayed on each trial (i.e., the arrow direction). The CorrectAnswer attribute ("1" or "2") is used to score the input collected in response to the Stimulus display, and provide feedback via the Feedback object.

Sub-objects are entered on a Slide using the SlideText and SlideImage tool buttons. Click the appropriate sub-object tool button and click the Slide grid area to create the sub-object.
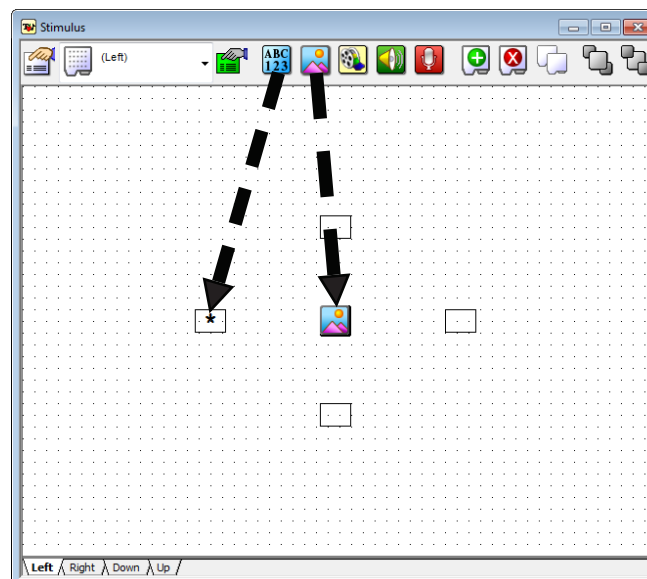
Each Slide state and sub-object maintains its own property settings. The settings may be accessed by selecting the appropriate state or sub-object from the dropdown menu on the Slide, and clicking the Properties button (Figure 1). In the properties for the sub-object selected in the image below, the value of the Up attribute is resolved at runtime by referring to the TrialList object. The value of the Up attribute will either be an asterisk (*) or nothing (i.e., will be blank).
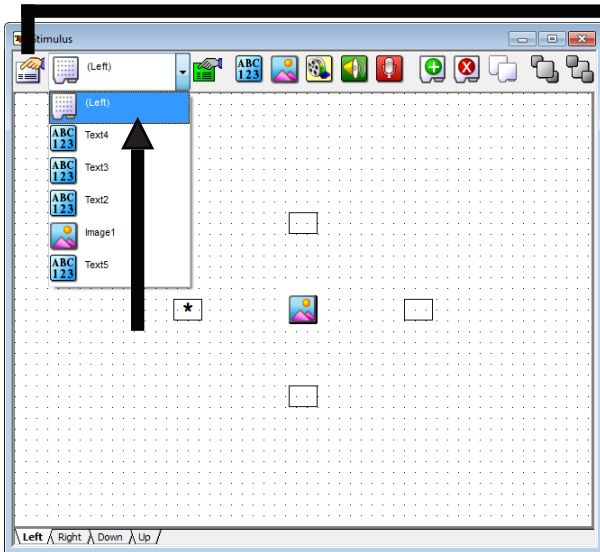
Figure 1. Selecting the sub-object on the Slide.
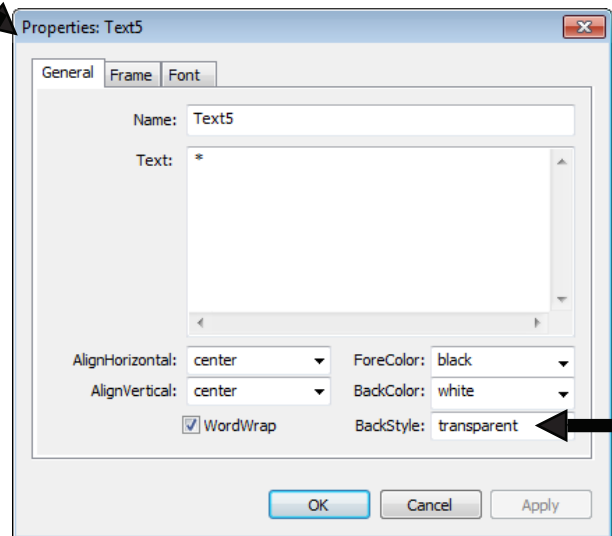
Figure 2. Set the BackStyle to transparent.

Display Sub-Areas in Color

To set the background for a specific part of the Slide object, select the sub-object defining that area, set the BackStyle property to "opaque", and set the BackColor property to the desired color from the dropdown box. Each sub-object maintains its own properties, so each may be set to a different color, size, font, etc.

Appendix D: Display Presentation

The figure below illustrates the three basic steps involved in the perception of a stimulus on a computer screen. The inset graphs show the time course over a 100 ms period. The first step is writing the stimulus to the video memory. A display command in the source code of an experiment instructs the program to write information to the memory on the video controller card. The second step is actually putting the stimulus on the screen, which occurs in scanning or "refreshing" the screen. This cycle will be explained in detail below, but note now that it is the limiting factor in how fast a stimulus can be put on the screen. The third step is activating the participant's retina. Let's consider each of these steps in turn.

Steps of Display Generation

1) Writing Video Memory
0.01 ms

Computer

Video Memory

2) Scanning the Screen
14.28 ms

Monitor

3) Activating the Retina
80 ms

Eye
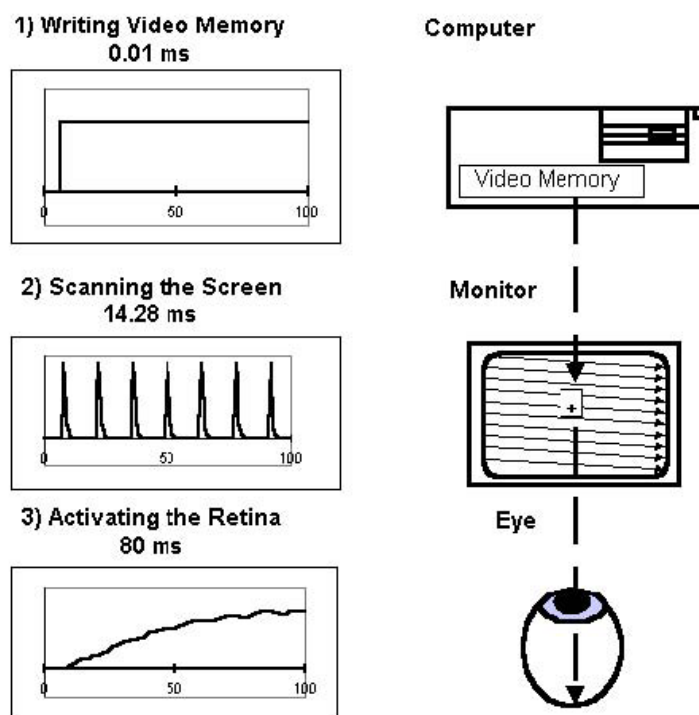
Figure 1. Steps involved in the perception of a stimulus displayed using a computer.

Step 1 - Writing to Video Memory

When a display command is executed, it writes data into a video controller card. This card has a certain amount of memory that can be thought of as a direct representation of the computer monitor (i.e., each memory location corresponds to a screen location). While the technical details of writing to video memory are not essential to this discussion, it is important to remember that this is a distinct event in the process. In the case of one or a few letters, the step is nearly instantaneous – on the order of tens of microseconds (0.00001 seconds).

If the stimulus involves extensive graphics (e.g., filling a circle), the process of writing to video memory can take a significant amount of time (milliseconds). The amount of time necessary for this step is dependent not only on the nature of the stimulus, but also on the computer and the display card.

233

Step 2 - Scanning the Screen

The second step is the physical activation of the stimulus on the screen. Older, CRT type computer monitors were based on raster technology, which works in the following fashion: The monitor has three raster guns, which emit a beam of electrons to activate red, green, and blue dots of color. The position of this beam can be adjusted to accommodate several different pixel resolutions. A pixel is the smallest unit of drawing on the monitor, and it is defined by the pixel resolution of the current graphics mode. For instance, if you are using the typical VGA graphics mode, the pixel resolution is 640x480, which means that there are 640 pixels per row and 480 pixels per column. SVGA is 800x600. The raster guns are deflected to draw each pixel on the screen sequentially, from left to right, top to bottom (see Figure 1, lines in Monitor), and the guns can only draw one set of pixels (red, green, blue) at a time. The raster guns shoot electrons at the screen, or front of the monitor. The electrons activate the phosphors to emit light. As soon as the guns are moved to the next pixel, the phosphors in the previous pixel begin to dim, or decay (see Figure 2 below). The decay period can vary from one monitor to the next, but for most color monitors, the intensity of the pixel will be 10% of its maximum within 2-5 ms. The video display card sequentially scans each memory location in order, turning on the gun's intensity based on the contents of the video memory set during Step 1.

The raster guns are continuously scanning the screen at a constant rate. This means that any given pixel on the screen is drawn with some constant frequency, known as the vertical refresh frequency. The vertical refresh frequency is the number of times per second that the entire screen will be drawn. The inverse of this measure, the refresh rate, is the amount of time necessary to draw one full screen. So, if a monitor has a 70Hz vertical refresh frequency, the refresh rate of that monitor is 1/70 of a second (= 14.2857 ms). Basically, once the scanning begins, it sweeps at 70Hz, independent of any writing to video memory operations.

Associated with the refresh rate is the vertical retrace, or top of screen event. This event occurs whenever the raster guns are moving from the bottom to the top of the screen to begin a new scan of the monitor. This event can be detected by the computer and can be used to synchronize the drawing of stimuli. By sensing the top of screen and only writing to video memory in a short period of time, the display will always occur within a fixed period of time after the top of screen event. For example, if text is displayed at the center of the screen and can be written in less time than half the refresh period (7 ms), the data will be in video memory before the display locations are scanned for the next cycle (7.14 ms after the top of screen event).

The effect of Liquid Crystal Display (LCD) monitors is basically identical to CRT monitors. Current active matrix LCD monitors work by having an electrical signal change the angle of the crystal of the LCD for each colored dot on the display. In contrast to a CRT display, where each dot is activated by an exponential function (see Figure 1 step 2), on an LCD the activation is a square wave signal (on for a given intensity during the time of the refresh). The LCD still has a refresh rate, which is the rate at which each dot is updated. Similar to the CRT, on the LCD the middle dot would be updated 7.14 ms after the first dot of the screen (on a 70Hz monitor). The effect on the eye of a LCD and CRT display are indistinguishable. See the KB 3133 - INFO: Monitor Recommendations and Timing Information for our latest test results.

Step 3 - Activating the Retina

The last step in the perception of stimuli on a computer monitor is the retinal response to the light emitted from the screen. The chemistry and neural circuitry of the retina are such that the visual system will integrate visual input over time. This is the reason that we cannot detect the raster guns scanning the monitor. The retina will average the input and produce the perception of a continuous display. For example, a stimulus which is to last for one refresh of the monitor will produce the following results: The pixels on the screen which make up the stimulus will be activated by the raster guns once, and then begin to decay. This decay process will take approximately 5 ms on a color display (see Figure 1, graph step 2) before the intensity is below perceptible levels. On LCD, the dot will be on for the duration of the refresh.

However, the eye integrates that impulse for the next 80 ms (see Figure 1, graph step 3). As an analogy, think of what occurs when you see a photographer's electronic flash go off. Typically, a very brief flash (microseconds) is presented and is perceived as a short (e.g., 100 ms) flash. If you flash the electronic flash 70 times a second, the retina would perceive the light as being continuous. During a period of about 80 ms, the eye is integrating the energy (e.g., a stimulus on for 5 ms at 10 Lux is seen as intense as a stimulus on for 10 ms at 5 Lux).

Example stimulus activation

Figure 2 provides an illustration[1] of the time course of the raster display on each scan, and the visual integration of the raster display. We want to display a '+' (text mode) in the center of the screen. Assume the monitor has a vertical retrace rate of 14.3 ms. We will examine the sequence of events, considering only the center pixel of the plus symbol. Execution of the display command occurs at time t = 0. After about ten microseconds, the '+' is written into the video memory. At some point during the next 0 to 14.3 ms, the raster guns will be aimed at the center pixel on the screen (see Raster Display peaks in Figure 2). With the stimulus in the center of the screen, the pixels for the '+' are displayed after 7.14 ms. As soon as the pixel has been activated, it will be refreshed every 14.3 ms. The retinal response begins as soon as the pixel is activated for the first time, but the response is delayed in the sense that it must build up over the next 80 ms to reach a steady state (see Raster Based Visual Activation Figure 2). At 200 ms the '+' is overwritten in video memory. When the raster guns reach that point (at 207 ms) where the '+' was, the pixel is not refreshed. The retinal activation begins to decay as soon as the pixel is no longer visible during the last refresh, and the visual activation slowly drops off over the next 80 ms.

There are four important aspects to note about the timing in the attached graph. First, there is a variable delay of up to one refresh period from the time that the stimulus is written to the video memory and the time that the center pixel is initially activated on the screen. The second feature is the exponential decay of the light intensity of the center pixel between refreshes. Third, the retinal response is slow and smooths the bumps in the pixel activation. Fourth, the decay of retinal activation is delayed, not instantaneous. If the stimulus is removed from video memory just after the pixel has been refreshed, the pixel will decay for another few milliseconds, and the retinal activation will persist for substantially longer unless it is masked by another stimulus.
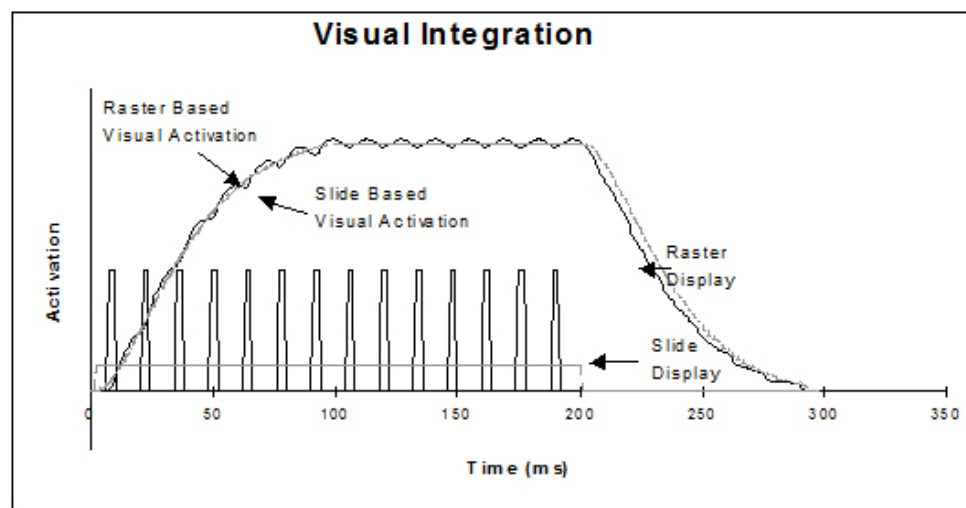
Figure 2. Time course of the raster display, and visual integration of the raster display on each scan.

[1]Figure 2 is based on simulation and empirical studies reported in Busey, T. A., & Loftus, G. R., 1994, Sensory and cognitive components of visual information acquisition, Psychology Review, 10, 446-469, assuming a decay constant of 20 ms.

Figure 2 also shows a comparison of presenting material with a slide shutter projector or tachistoscope relative to a computer monitor. The lines labeled Slide Display and Slide Based Visual Activation show the activity that would be produced if the total display intensity were continuous as opposed to pulsating (i.e., like the refreshes of a computer monitor). The total visual energy of the Slide and Raster activation have been matched. Note the similarity of the activation functions[2]. For the participant, there is no perceptible difference. For a detailed discussion of these issues, see Busey & Loftus, Psychology Review, 1994, 101, 446-469.

[2] In the simulation the slide display was delayed by 2 ms and the raster by 7 ms. This produces functions that have matching increases in activation within a few percent from 50 ms until the termination of the display. These differences are insignificant in terms of human perception (see Busey and Loftus 1994). In Figure 2, the decay of the activation for the Slide Display was somewhat slower than the Raster Display due to the last refresh of the Raster Display not being displayed as a result of clearing video memory 200 ms after the write to video memory but rather than 200 ms after the first display of the video.

Appendix E: Timing of Object Execution

The figures below indicate the timing of events that compose the execution of runnable objects within E-Prime 2.0. The first figure illustrates the events of a single object, while Figures 2 and 3 illustrate the relationships between multiple objects when using Event and Cumulative timing modes. Each time point that is noted is logged in the data file as a property of the object when Time Audit and Time Audit (Extended) logging is enabled via the Logging tab for an individual object. Refer to the Time Audit topic in the E-Basic Online Help for complete definitions of each property.

Figure 1. Events composing the execution of a single object
The timing for a single event is influenced by the steps necessary to both set up and clean up the object, as well as other considerations, such as synchronization with the refresh of the screen, the clearing of the display, or the timing mode used by previous objects.
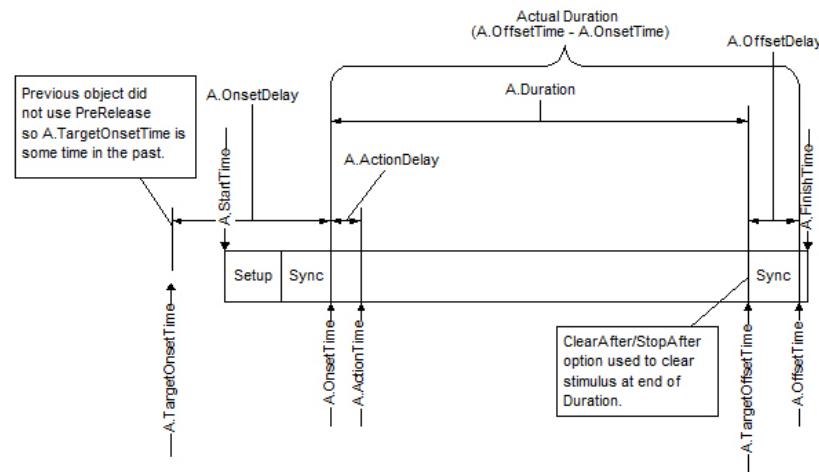
In general, each object first goes through a setup stage to prepare its stimulus. It then (optionally) synchronizes with a vertical refresh signal, enables any associated response inputs, and presents its stimulus. The object then waits for a precalculated duration (which is influenced by timing mode). If the object is to remove its stimulus at the end of the duration, it again (optionally) syncs with the vertical retrace, removes its stimulus (e.g., by clearing the screen), performs some internal housekeeping activities, and finishes its execution.

Figure 2. Event mode timing with no PreRelease
Duration is calculated from the actual onset time of the object. With Event-Event mode, the Duration of the object is maintained, and OnsetDelay is expected to be positive due to the processing time necessary to set up the next object.
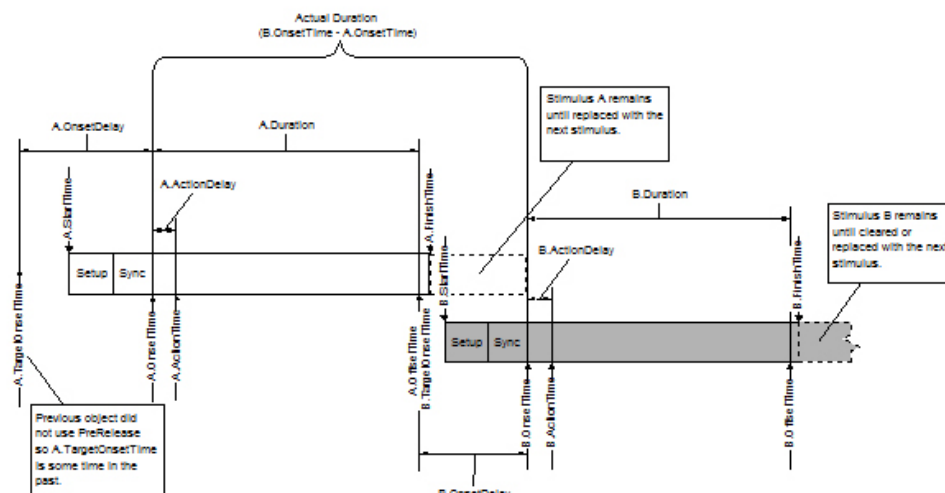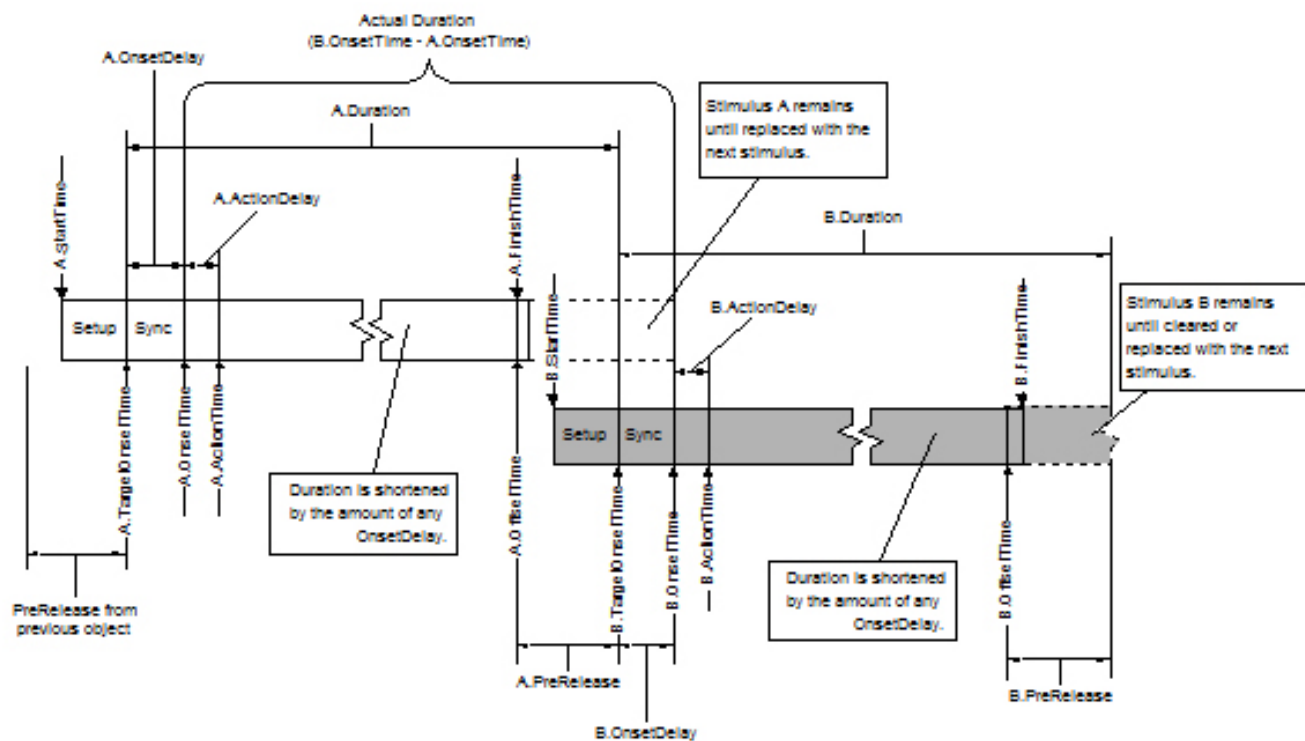

237

Figure 3. Cumulative mode timing with sufficient PreRelease
Duration is calculated from the target onset time so that the onset of each event occurs at fixed intervals. With Cumulative-Cumulative mode (and PreRelease), the Duration is shortened to maintain a fixed interval between onsets of objects.

Glossary

Term	Definition
Administrator	E-DataAid permits the opening of files as an Administrator via the Admin Open command in the File menu. To open a file as an Administrator, the user must know the password for the machine. Opening a file as an Administrator permits access to the security restrictions for the opened file. An Administrator may restrict the amount and type of edits permitted within the file, and may restrict access to annotations.
altered data files	Any EDAT or EMRG file that has been modified in any way.
Analysis files (.ANL)	Pre-created or saved analysis configurations to be loaded into E-DataAid.
annotations	Annotations describe modifications made to, or processes performed on, an E-Prime 2.0 data file. E-DataAid's annotation feature is like an electronic version of the experimenter's notebook, in which descriptions of any edits or processes performed on a data file are automatically entered as separate annotations. Annotations may be viewed using the Display Annotations command in the View menu.
attribute	Structure holding the individual values (i.e., levels) for the independent variables and control variables used within the experiment. Attributes are defined in the List object, or using the SetAttrib method in script.
Attributes window	The Attributes window is a dockable window located on the right side of the application window by default. It features a list of every attribute declared within the experiment's List objects, regardless of the List in which it resides.
Batch Analyses	Allows the user to run multiple pre-saved analyses in rapid succession without individually loading and running each analysis file.
Batch Analysis	Set of pre-defined analyses to be run as a group.
binning	The grouping of the values within a specific variable according to a bin value during an analysis. For example, reaction time may be binned into 100 ms groups. Only variables containing all integer data may be binned.
bracket notation	Method used to pass information by referencing an attribute. The bracket notation (i.e., [attribute]) alerts E-Run to resolve the value by referencing the current sample in the running List object.
Browser	View listing all defined E-Objects within the experiment specification.
caching	The process of pre-loading stimuli into memory in order to reduce time required by subsequent access to the same data.
Canvas object	The Canvas object abstracts the programming interface to a particular canvas/page/surface, and supports all of the drawing/graphics calls that are available in the system.
checklist	Type of filter allowing the user to check specific values for a variable to be included in a filter.
colon syntax	Method used to sample multiple items from a single list on a single trial.

Term	Definition
compile	In E-Run, compile takes the generated EBS file from E-Studio and feeds it into the E-Basic compiler. The process of compiling checks the syntax of the EBS file contents to make sure that it is in an acceptable format to run.
conflicts	Incompatibilities or inconsistencies between two data files, which can lead to problems during a merge operation.
context menu	Collection of related commands pertaining to a specific object. Access a context menu by right clicking on an object.
Context object	Encapsulates the data and routines involved in the overall experiment.
copy protection	See hardware key
Cumulative mode	Timing is maintained relative to a set of consecutive events. This mode attempts to ensure that the inter-stimulus-interval remains accurate but allows the duration of the event to be modified. Actual durations of individual events are adjusted to maintain the cumulative duration of the set (i.e., the system will attempt to keep the stimulus onset time of consecutive events accurate by shortening each event's duration to consume any setup or error time encountered).
Custom mode timing	Timing is maintained by the user. This mode allows the user precise control over the onset and offset of an event. Duration and PreRelease settings are ignored in this mode and the lifetime of the event is totally controlled by the CustomOnsetTime and CustomOffsetTime properties. The required use of Custom Timing Mode is rare, and users are encouraged to first try to solve their timing problems with either Cumulative or Event mode timing in combination with adequate PreRelease values.
cycle	A single, complete execution of the designated sampling set, and determines when the exemplars should be returned to the List for continued sampling.
Design-time	All activities related to experiment specification prior to generation.
E-Basic	The scripting language underlying E-Prime 2.0.
EBS file	E-Basic Script file resulting from the Generate command, and used to run the experiment.
EDAT file	E-Prime 2.0 data file containing data collected using E-Run.
E-DataAid	The E-Prime 2.0 application allowing viewing and editing of data files.
E-Merge	The E-Prime 2.0 application allowing individual files to be merged into a file containing more than a single session of data.
EMRG file	E-Prime 2.0 data file containing data for one or more sessions.

Term	Definition
E-Objects	Design-time building blocks of an experiment located in the Toolbox within E-Studio. The registered E-Objects in the Toolbox are used to build an experiment by dragging and dropping the icon on a valid target, and to define the events of individual procedures.
E-Prime 2.0 text file	File containing data collected using E-Run, which has been exported to a tab delimited text file using the Export command in E-DataAid. This file is converted to an EDAT file upon successful completion of a run, or may be manually converted using E-Recovery.
ES2 file	The E-Prime 2.0 experiment specification file containing the objects and properties defining the structure of the experiment.
E-Studio	The experiment development application within E-Prime 2.0.
Event mode	Timing that is maintained relative to an individual event. This mode attempts to ensure that the duration of each event remains accurate but allows the inter-stimulus-interval to vary (i.e., the actual event duration is the user-specified Duration for that event and no adjustments to the event's duration will occur).
exemplar	Single instance of stimulus/independent variable database (i.e., a row in a List object).
Experiment Object	The Experiment Object is created automatically for the user when a new experiment specification file is opened. It appears only as the first object in the Structure window, with the experiment name listed in parentheses, and is used to set global properties of the experiment.
Export	Writes the E-DataAid spreadsheet or table of means to a text file for import into another application (e.g., Excel).
FeedbackDisplay object	Object used to present feedback based on input collected by another object.
File List view	View, in E-Merge, allowing selection of files to include during a merge operation.
filter (E-DataAid)	Restriction placed on the spreadsheet to limit the amount of data displayed or exported.
filter (E-Merge)	Restriction placed on the File List to limit the types of files displayed.
flush input buffer	Clears pre-existing responses from the input buffer. May result in negative RTs.
folder tree	View, in E-Merge, displaying the available drives, folders, and sub-folders on the machine.
generate	Process by which experiment specifications are converted to script for execution.
hardware devices	Devices (i.e., keyboard, mouse, SRBox, etc.) that are enabled and initialized by the Devices tab for use in experiments.

Term	Definition
hardware key	The E-Prime 2.0 development application (E-Studio) has a copy protection in the form of a hardware key that connects to the computer's parallel or USB port.
ImageDisplay object	The ImageDisplay object is used to display pictures in BMP format.
Import	Read a MEL Professional data file saved as text, a PsyScope data file, or an E-Prime 2.0 data file saved as text into E-DataAid.
inheritance (E-DataAid)	Inheritance allows any multi-level variable cell that has missing data to inherit its data value from the cell in the same row for that variable's next highest level. Inherited values are italicized in the Spreadsheet
InLine object	Object used to enter user-written script in the experiment specification.
Label object	The Label object is used to mark a position in a procedural timeline.
level (Context)	Hierarchical level in the structure of the experiment. For example, a typical experiment involves a Session level, Block level and Trial level.
level (List Object)	A single line entry in a List object. See also exemplar.
List object	Object organizing the data to be manipulated within the program (i.e., stimulus values, independent variables, etc.).
log level	Hierarchical level in the structure of the experiment at which the data is stored. For example, a typical experiment has a Session level, Block level, Trial level, etc.
MEL Pro text file	File containing data collected using MEL Professional which has been exported to a text file using the ANALYZE or DEDIT programs from MEL Professional.
merge operation	The process of merging a source file into a target file.
merged data files	Merged Data files (EMRG) are E-Prime 2.0 data files which contain data from more than one session. They may be merged into other files (i.e., act as a source file) or receive data from other files (i.e., act as a target file).
missing data	A cell, in E-DataAid, containing no value. Missing data is displayed as NULL to be distinguished from a null string (""), or the string value "NULL".
modified date	The last date on which any edits were performed on a data file.
nesting/nested lists	Technique used during sampling to allow for multiple levels of randomization and selection.
non-E-Prime 2.0 data file	Any file which is not a single session data file collected by E-Run (EDAT file), or a data file containing merged single session data files (EMRG file).
null string	An empty string ("").
observation	A single data point collected by a summation object.
Output	View, in E-Studio, supplying feedback concerning script generation, running of script, or debugging commands called in user-written script.

Term	Definition
PackageCall object	The PackageCall object permits a block of E-Basic script to be loaded into the Experiment Specification file from a file in a specific format.
parent object	An object which holds/houses other objects (e.g., a Slide object holds SlideState objects).
password	Keyword required to open a data file in E-DataAid as an Administrator.
PreRelease	Amount of time released during the processing of the current object to allow for setup of the next object.
Procedure object/ Procedural timeline	E-Object used to define the events of an experiment. Objects are dragged from the Toolbox to Procedures, which define the timelines for the various levels of the experiment.
properties	Assignable characteristics of an object.
Properties window	The Properties window is a dockable window that is used to display all of the standard properties and property values associated with objects used in the experiment. By default, the Properties window appears to the immediate right of the Toolbox and just below the Structure window.
PsyScope data file	File containing data collected using the PsyScope application.
range	Type of filter allowing the user to specify a range of values for a variable to be included in a filter.
Recursive Merge	The process of merging all unmerged single session E-Prime 2.0 data files that reside in the active folder and its sub-folders into the designated target file.
Refresh cycle	Process of painting the display from top to bottom.
Refresh rate	How often a display is redrawn per second.
reserved words	Words recognized by E-Basic as part of the E-Basic language. These words are considered "keywords" and their use is restricted.
Run	Occurring during the execution of the E-Basic script file.
script	Programming language code generated automatically by E-Studio based on the Experiment Specification (ES2) file, or user-written code which is entered as part of the es file.
Script window	The Script window features two tabs, User and Full. The Full tab displays the entire script as generated based on the object placement and specification within E-Studio. The User tab enables the user to add his/her own subroutines or global variable declarations.
scripting object	Scripting objects differ from E-Objects in that there is no graphical user interface with which to set or determine properties. All properties and methods for scripting objects are only accessible through user-written script in an InLine object.

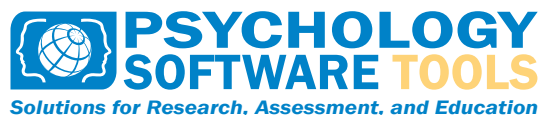| Term	Definition
security levels	The E-DataAid application offers a security feature allowing a user to set security options when supplying a valid password. For example, in E-DataAid, a user may set the security options to disallow edits to participant and session number, or to disallow all edits. When a source file containing a variable with a certain security level is merged into a target file containing the same variable with a different security level, the variable in the target file will have the security level that is the stricter of the two files upon completion of the merge operation.
selection	The randomization method used to determine the sampling order from a List (e.g., sequential, random, etc.).
Session	A single execution of a program during which a single participant data file is collected.
Session level variables	All parameters enabled via the Startup Info tab are logged as session level variables in data files. Their values do not vary during the session.
single session data files	Single session files are E-Prime 2.0 data files generated by the E-Run application containing data for a single participant. They have the EDAT file extension.
Slide object	Object used to present text, pictures, audio, or combinations thereof simultaneously.
SlideImage	Sub-object on a Slide object or Feedback Display object designed to insert a picture.
SlideState object	Objects used to enter and organize sub-objects for presentation or feedback.
SlideText	Sub-object on a Slide object designed to insert text.
SoundOut object	The SoundOut object is used to present pre-recorded digital audio sound in WAV file format to the participant. It maintains and manages a buffer on a specific sound device.
Source file	A file, containing E-Prime 2.0 data, to be merged into a target file.
spider down	Command used to expand the branch selected in the Structure window, including all sub-branches.
spreadsheet	The spreadsheet is the main view in the E-DataAid application, and appears in the center of the application display. The spreadsheet displays the data included within the currently opened file in a grid format similar to Excel. The columns in the spreadsheet represent the experiment variables, and the rows represent the lowest log level in the experiment (e.g., Trials).
standard merge operation	The Standard merge operation (default selection) is the process of selecting source files from the File List view and merging them into a target file.
Structure window	A hierarchal representation of the experiment events and structure.
sub-objects	An object which is part of another object (e.g., a SlideText object exists only on a Slide object).

Term	Definition
Summation object	Summation objects are used to collect a series of observations, and perform calculations on the collection.
target file	File into which data files will be merged.
termination	Condition upon which the processing of an object ends (e.g., duration expires, input is received, etc.).
text data file	Any data file (e.g., E-Prime 2.0, Mel Professional, PsyScope, etc.) saved in tab delimited text format.
TextDisplay object	Object used for displaying text stimuli.
Toolbox	Framed window in E-Studio that displays the icons of all currently registered E-Objects (e.g., Procedure, TextDisplay, etc.).
unaltered data file	A data file which has not been modified.
Undo (E-DataAid)	The Undo command reverses the last operation performed on an E-Prime 2.0 data file.
Undo (E-Merge)	The Undo command reverses the last merge operation performed. E-Merge supports only one level of Undo.
Unmerged	A data file which has not been merged into another data file.
Unreferenced	Objects not explicitly called by a Procedure object (i.e., unused objects, or objects launched only via script).
User	E-DataAid permits the opening of files as a User via the Open command in the File menu. A User is participant to the security restrictions set on a specific file by the Administrator.
variables	Variables are distinguished from attributes in that they are not related to the Context object, they are defined within a particular scope. That is, variables temporarily hold or pass information, and are not seen by the program outside of the procedure in which they are defined. Variables are not logged in the data file.
vertical blank	The start of a refresh cycle is referred to as the vertical blank event, the time that the electronic gun on the monitor moves from the bottom of the display to the top to restart the refresh.
Wait object	Object used to delay processing of the experiment script for a designated duration.
Weight	Number of repetitions for an exemplar within a list.

Psychology Software Tools, Inc. was founded with the vision of creating innovative and affordable technologies and solutions which improve the efficacy of human behavioral research, assessment, and education. For more than two decades, Psychology Software Tools has worked to meet the needs of researchers and educators, and has acquired the experience necessary to provide the best possible tools and highest quality product support to our loyal users. The ongoing success of E-Prime® has been essential in helping us to realize our corporate mission. E-Prime® is a leader in the field of behavioral research and education, with systems in use in over 50 countries. We are certain that E-Prime® will benefit you in your research, and we welcome you to the E-Prime® user community!

Psychology Software Tools, Inc.
311 23rd Street Extension, Suite 200
Sharpsburg, PA 15215-2821 USA
Voice: 412.449.0078
Fax: 412.449.0079
Email: info@pstnet.com
www.pstnet.com

PSYCHOLOGY SOFTWARE TOOLS
Solutions for Research, Assessment, and Education