Software for Generating Psychological Experiments

Christoph Stahl

Albert-Ludwigs-Universität Freiburg

Abstract. This review compares four software packages for generating and running psychological computer experiments. It aims to inform researchers' decisions about which software to choose for their lab. Four widely used general purpose commercial packages available for the Windows platform are considered: DirectRT, E-Prime, Inquisit, and SuperLab. An overview of each package is given, and the implementation of two test experiments is described. Packages are evaluated with regard to the amount of complexity reduction they provide for the task of programming an experiment and the variety of experimental designs they can accommodate.

Keywords: experiment generation, software, computer experiments

For researchers in experimental psychology, the personal computer serves as an indispensable tool: it presents visual and sound stimuli and registers participants' responses from an array of different devices (e.g., keyboard, mouse, voice key, external devices) with high temporal accuracy.

To implement a study on a computer, researchers can either learn a programming language, or they can use one of the various available software packages for creating and running psychological experiments. These software packages attempt to reduce the programming skills necessary to create psychological computer experiments by either providing an experiment definition syntax with a small set of commands or a visual drag-and-drop interface. Thus, they reduce the complexity of the problem of programming a computer experiment. Yet, this sometimes comes at the cost of reduced flexibility: A reduced set of commands can implement only a reduced set of experimental paradigms. This review will focus on the ease of implementing standard paradigms as well as on the flexibility to implement a wide variety of paradigms.

Software packages for creating and running psychological computer experiments on the Windows operating system are considered if they are stand-alone packages (e.g., Matlab-based packages were not considered), are not focused on a subset of paradigms or bundled with measurement hardware (e.g., fMRI), and if user support is offered by the distributor. DOS-based packages (e.g., ERTS, NESU) are not considered because of limited hardware support.[1] The discussion section points to software packages that are not included in this review but are especially well-suited for psychophysics experiments, for Web-based research, or for teaching.

Table 1 presents the availability and requirements of the four packages included in this review: DirectRT (2004), E-Prime (2004), Inquisit (2004), and SuperLab (2004). All four packages rely on DirectX technology[2] to support stimulus presentation and response registration with high temporal resolution, and to interact with the output (e.g., video and sound) and input interfaces (e.g., keyboard, mouse, and joystick) available for the Windows operating system. The reviewed packages claim millisecond accuracy in stimulus presentation and response registration; supporting evidence has been reported for Inquisit (De Clercq, Crombez, Buysse, & Roeyers, 2003). It is beyond the scope of this review to provide a test of that claim for the other packages. In general, some caution is necessary regarding timing accuracy on a Windows operating system: Because it supports multithreading, (i.e., multiple processes running at the same time, sharing one central processing unit), perfect timing accuracy cannot be warranted (see also Myors, 1999). If an experiment program—be it one of the above mentioned packages or self-programmed—is to present a stimulus at time *t*, it can do so accurately if no other processes are running. If, however, another program is running in addition to the experiment program, it might occupy the central processing unit at time *t* and thus delay stimulus presentation. Yet, within the DirectX framework, several measures can be taken to minimize this and other sources of timing error (e.g., Forster & Forster, 2003), and it can thus be assumed that timing accuracy is potentially high for the reviewed packages (MacInnes & Taylor, 2001; Plant, Hammond, & Whitehouse, 2002).

The aim of this review is to inform the decision for an experiment generation software package by describing how each package attempts to reduce the complexity of the programming task. A description of each software package is given, considering the package's installation, stimulus and response features, the general approach to creating an experiment, running an experiment, as well as the data out-

[1] Because the DOS platform is outdated (i.e., is no longer supported by its manufacturer), new hardware devices are often not supplied with a device driver for that platform.

[2] The current DirectX version is freely available from http://www.microsoft.com/directx/homeuser/downloads/default.asp.

Experimental Psychology 2006; Vol. 53(3):218–232
DOI 10.1027/1618-3169.53.3.218

Table 1. Availability and requirements . . .

Feature	Software Package			
	DirectRT	E-Prime	Inquisit	SuperLab
Manufacturer	Empirisoft	Psychology Software Tools	Millisecond Software	Cedrus
Web site	www.empirisoft.com	www.pstnet.com	www.millisecond.com	www.superlab.com
Price for single license	$ 875,- (Academic License: $475,-)	$ 695,-	$ 745,- (Academic License: $ 395,-)	€ 620,- (Educational Price € 530,-)
System requirements	Microsoft Windows 95, 98, ME, 2000, or XP; DirectX 7 or higher	Microsoft Windows 95, 98, ME, 2000, or XP; DirectX; 100 MB of hard drive space, USB or parallel port	Microsoft Windows 98, ME, 2000, or XP, DirectX; 10 MB of hard disk space	Microsoft Windows 95 or later, 1.5 MB of hard disk space

put generated. Two test experiments are implemented within each package, allowing a closer look at their strengths and weaknesses. Table 2 presents a feature overview and summary of the results. The packages are discussed with regard to their flexibility and ease of use.

Test Experiments

To compare ease of use and flexibility of the packages, two experimental designs were implemented within each package: a Remember/Know test frequently used in the memory domain, and a Lexical Decision task that is common in priming research (a detailed description of the experiments' methods is given in the Appendix). During the implementation, an attempt was made to reduce the complexity of the task of generating an experiment and of the experiment definition itself. The implementations were then reviewed and optimized by the package manufacturers.[3] They are available from http://www.psychologie.uni-freiburg.de/Members/stahl/Expsoft.

Remember/Know Paradigm

In a Remember/Know paradigm, participants give judgments about their mnemonic states for a list of stimuli some of which they were presented with before. Typically, three response options are available: (a) Participants give a *remember* response to an item if they have a clear recollection of at least one aspect of that item's presentation episode, for example, its color; (b) a *know* response is given if participants are confident that the item has been presented based on its familiarity but cannot recollect any detail from its presentation episode; and (c) a *new* response indicates participants' judgment that the item has not been presented before.

Besides being presented simultaneously, these options are often presented sequentially. In these studies, participants first have to indicate whether or not an item has been presented before. Only after a positive response, a second screen presents the question regarding the mnemonic state, together with the response options *remember* and *know*. Thus, the second question conditionally appears depending on the answer to the first question. This is an important feature for an experimenting software package to support, as it can be required in a range of other paradigms (e.g., source memory, adaptive testing). The experiment will be implemented in the sequential version, if possible.

In the retention interval preceding the memory test phase, participants are to solve randomly created arithmetic problems for a fixed time of 5 minutes. Participants are to provide the solution by typing it on the keyboard. This solution is to be recorded and checked for accuracy, and feedback is to be given regarding the accuracy of solutions.

During the implementation of this paradigm, three critical features will be focused on: first, presenting stimuli conditional on the participants' responses; second, randomly generating arithmetic problem trials with open-ended text responses (i.e., participants are free to enter any sequence of characters, such as numbers or text); and third, accuracy feedback.

Lexical Decision Paradigm

In a Lexical Decision task, participants decide if a letter string (e.g., *butter* or *cbdghs*) is a word or not. This task is often used in semantic priming studies. In these studies, just before the target's presentation, a prime is presented briefly. When prime and target are related semantically, for example, when the word *bread* is presented as a prime shortly before the target word *butter*, responses are facilitated compared to when no prime is presented or when an unrelated prime (e.g., the word *shoe*) is presented.

This type of task is subject to the problem of speed-accuracy trade-off (a fast response is more likely to be incorrect, and a correct response is more likely to be slow), and variance of interest will likely be found in the latency data as well as in the accuracy data. A response window procedure can help to reduce this problem: Here, participants are required to respond within a short period of time (e.g., between 300 and 700 ms after stimulus onset). By

[3] The paradigms can be implemented in more than one way within each package. Therefore, it is possible that an experienced user can come up with less complex definitions than those provided here. However, because the present implementations were corrected and optimized by the manufacturers, these differences should be negligible.

Table 2. Feature overview and summary . . .

Feature	Software Package			
	DirectRT	E-Prime	Inquisit	SuperLab
Features required for Remember-Know paradigm				
Conditional presentation	yes	yes	yes	no
Random generation of arithmetic problems	yes*	cbi	no	no
Accuracy feedback	yes*	yes	yes	single-key responses
Features required for LDT with adaptive response window				
Fixed window	yes	yes	yes	no
Adaptive window	no	cbi	yes	no
Block feedback	no	cbi	yes	no
Randomization and stimulus selection				
Random trial order	yes	yes	yes	yes
Random stimulus selection with replacement	yes	yes	yes	no
Random stimulus selection without replacement	yes	yes	yes	no
Balanced orders	cbi	yes	cbi	cbi
Superimposing stimuli	yes	yes	yes	no
Randomized assignment of participants to conditions	yes	yes	no	no
Multiple sessions per participant	cbi	yes	cbi	cbi
Features for psychophysical research				
Stimulus adjustment	no	cbi	no	no
Stimulus calibration	no	no	no	no
Adaptive testing	no	cbi	cbi	no
Summary judgments				
Support quality	1	3	2	n/a
Implementation time	1	2	3	4
Handling ease	1	2	3	4
Flexibility	3	1	2	4

Note. yes = feature is supported; cbi = feature is not supported but can be implemented; no = feature cannot be implemented; * see text. Summary judgment values represent rank orders, with 1 = best ranking.

introducing a response window, variance within the latency data is reduced, and the variance of interest will more likely be found in the accuracy data. To avoid floor and ceiling effects in the accuracy data, the response window can be adapted to each participant's individual performance. After each block, mean accuracy is measured; the response window onset and offset times are increased by a specified amount if mean accuracy falls below a lower criterion, and decreased if mean accuracy rises above an upper criterion.

Flexibility of response options will be the focus during implementation of the Lexical Decision paradigm. If possible, it will be implemented with an adaptive response window; if not, a fixed response window or a fixed time-out is used.

Description of the Software Packages

DirectRT

Installation and Interface

In addition to the main program, the installation includes help files, sample experiments, and software tools to merge data files and test the parallel port and sound interfaces. Upon starting the program, a single window opens providing a set of menu commands. To create an experiment, an external editor (e.g., Notepad) is opened from the File menu. The Edit menu allows creating and modifying DirectRT presentation styles. Presentation styles define the visual properties of a text stimulus, for example its font, color, and size, as well as the response time-out setting for the stimulus. The Tools menu provides several diagnostic functions (e.g., a test of the monitor refresh rate, access to information about the display and sound interfaces and about input devices, setup and testing of DirectRT's voice input feature, and a tool to test TTL signal communication). The Help menu provides access to a detailed description of the software and its user interface as well as to about 20 sample experiments that can be used as templates in experiment generation. Figure 1 shows the DirectRT window, an experiment definition file, and the style editor.

Stimulus and Response Options

A word or line of characters can serve as a stimulus, as well as a longer passage of text, an image, sound, video, or an animation. DirectRT registers participants' responses from keyboard, mouse, joystick, and microphone. In addition to key press times, the time it takes to release a key can optionally be registered for keyboard responses. DirectRT also supports TTL signals from the parallel port and joystick motion (as a discrete or as a continuous variable). In addition to computing speech onset times as a dependent variable from the microphone input, the vocal response is optionally saved to disk for additional analy-

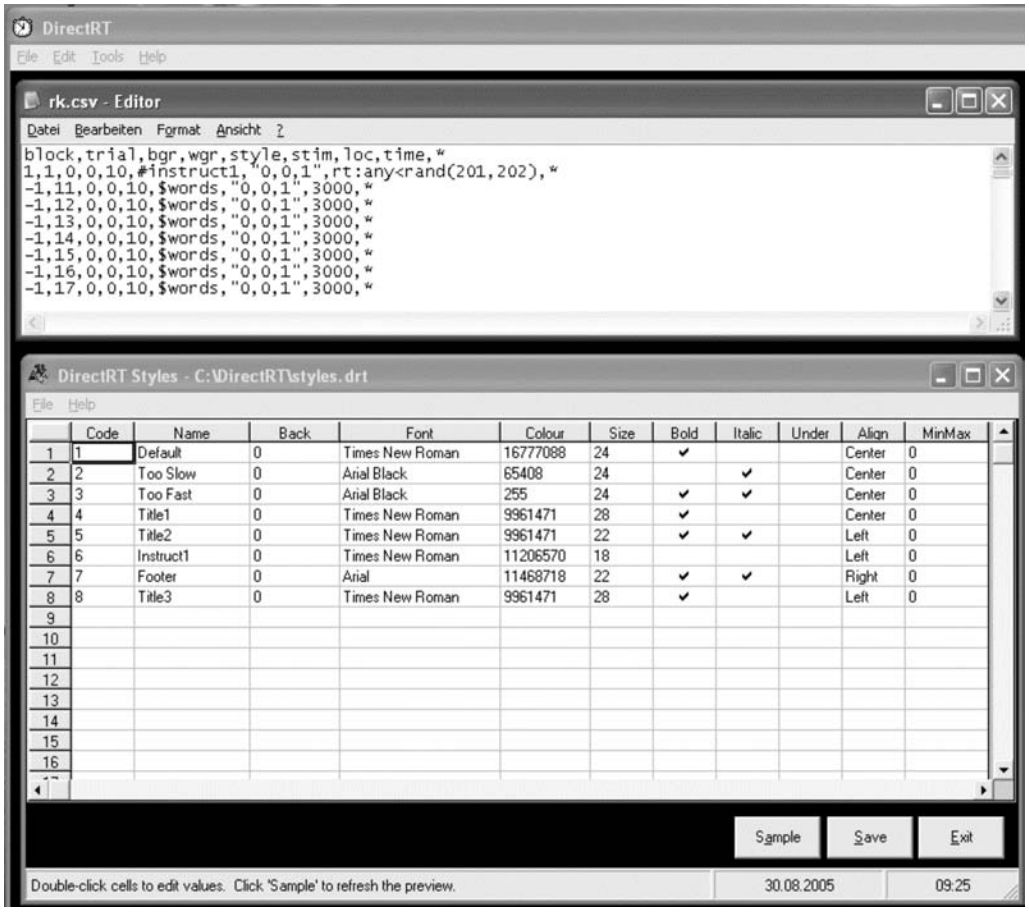

Figure 1. The DirectRT program window, experiment definition example, and style editor.

ses.[4] Together with the microphone input, an additional keyboard response can be collected (e.g., for coding of vocal responses by the experimenter).

Creating an Experiment

DirectRT simplifies the creation of an experiment by reducing the set of commands needed to define the trials and blocks that make up an experiment, and by using a simple text file format for its definition files. DirectRT experiments consist of a simple list of trials that are processed successively. Each trial is defined on a separate line by eight required values. These include the following: number of block, number of trial, number of presentation style (denoting a specific font size and color setting), stimulus content, stimulus presentation location, presentation duration or type of response, and two parameters controlling randomization of trial and block order. For stimulus content, the name of the file containing the stimulus is specified in case of images, sound, and video, or longer passages of text. Alternatively, if the stimulus consists of a single word, it can be included into the experiment file directly. As a third possibility, a text file can be specified that contains the names of the stimulus files. Location of stimulus presentation on-screen can be specified either in percentage or pixel values. Presentation duration is specified in milliseconds. For stimuli that require a response, the response device and the valid and correct responses keys are specified. The presentation sequence can be specified to continue with different trials conditional on participants' responses. By adding values for their content, presentation location, and presentation time, any number of stimuli can be added to each trial. A response can be registered for each of these stimuli.

The two remaining parameters control randomization between groups and randomization within groups of trials: The presentation order of two groups of trials can be randomized, and trial order can be randomized within a group of trials. Additionally, stimuli can be selected randomly with or without replacement from specified stimulus lists or from separate text files containing stimuli. Numerical values can be selected at random, either from a specified set of values or from a specified range.

[4] This feature should be used with caution, however, because as one reviewer remarked, accessing the computer's hard disk during experiment execution may interfere with time measurement and other procedures.

Running an Experiment

Once created, an experiment definition file can be executed either via the File menu, or from the command line. The participant's number, experimental condition, and the range of trials to be run have to be entered at start, or they can alternatively be specified in the command line call. Experiment execution can be terminated by pressing the Escape key. Pressing the Windows key or the Alt + Tab key combination results in switching to other programs' windows, or to the Desktop. This can be a problem when participants accidentally press the Windows key. It can be solved by switching back to the experiment window, which causes the execution of the experiment to continue.

Data Output

Data are stored in two comma-separated text files, one containing a complete log of the session, and the other containing only a subset of values. This subset includes the participant, condition, block, trial, trial order, code and name of response key, stimulus content, reaction time, and correctness of response variables. The complete log file contains additional data on date and time, the experiment file that was run, the actual onset of each trial computed from the start of the experiment, the actual duration of each trial, as well as content and actual presentation duration for each stimulus of that trial. DirectRT provides software tools to merge and reduce data files to a single line per participant.

Implementing Test Experiments

Remember/Know Paradigm

The presentation part of the experiment is quickly created by modifying a sample experiment included with the package. The instructions are defined on a single line each (with the respective texts contained in separate text files). Random presentation of rote versus imagery instructions is accomplished with a random jump from the welcome page to either one of the instruction pages. Subsequently, execution continues with the presentation of stimuli. For each stimulus, a separate line is included in the definition file, and stimuli are selected at random from a separate text file. Although these trial definitions do not differ except in trial number, further simplification (e.g., by defining the number of repetitions of a trial) was not possible.

Generating the retention interval task of randomly created arithmetic problems is more difficult. First, the duration of the arithmetic problems block could not be specified within the package. Second, while generating random multiplication problems was easily accomplished, it was not possible to provide accuracy feedback for these. This was due to the fact that open-ended responses could not be checked for accuracy.

These limitations were largely removed in a new version of the software that was supplied by Empirisoft upon request. The new release allowed a time-out to be specified such that the retention interval is aborted and the first trial of the memory test block is automatically executed after the specified time. Also, the new release allowed open-ended responses to be checked for accuracy. However, this applies only to each individual trial—feedback is not possible at the block level. Also, randomly generated arithmetic problems cannot be checked for accuracy because correct responses have to be specified in the definition file before the problems are randomly generated. Thus, a fixed set of arithmetic problems was created manually. The set size was chosen large enough to avoid running out of problems for fast participants. The retention interval was restricted to 5 minutes, and accuracy feedback can now be given for each of the arithmetic trials (but not for the entire block).

The Remember/Know memory test was implemented in its sequential variant in DirectRT. For the memory test block, the old/new question was added to the presentation block trials as an additional stimulus; and in the response options, the valid and correct response keys were specified instead of a time-out value. Similarly, a Remember/Know stimulus is added to each probe item; and a conditional jump is inserted such that in case of a *new* response, the Remember/Know question is not presented. The regular data file contains the data from the old/new trial in case of a *new* response, and the data from the Remember/Know trial in case of an *old* response (if the reaction time data for the old/new response is required, it can be extracted from the log file). Thus, with the exception of some minor points, the Remember/Know memory test was successfully implemented.

Lexical Decision Paradigm

Building on a sample provided with the installation package, the definition file for this experiment was easily created. Different text files were used to store primes and targets, with corresponding order for primes and targets. Stimulus selection was set to sequential sampling from each text file, and trial order was randomized. Thus, each prime-target pair was presented together on a trial.

An adaptive response window feature is not incorporated in DirectRT. As an alternative, a fixed response window was implemented. This was achieved as follows: An additional stimulus is added to each trial (a black "!") to indicate the window onset; it is displayed 300 ms after target presentation and replaced by a red "!" after an additional 400 ms, indicating window offset. A response within the window causes a jump to a feedback trial indicating a response within the window (a green "!" is presented for 300 ms) and the accuracy of the response. As mentioned above, the current release of DirectRT does not support block-level feedback on mean latencies and error rates.[5]

In summary, the Remember/Know paradigm was implemented largely as intended, and the Lexical Decision task was implemented with a fixed instead of an adaptive response window. Some limitations were encountered: First, it was not possible to simplify the definition of the struc-

[5] Empirisoft announced that these features will be included in the next release of DirectRT.

turally identical trials in the presentation phase, the old/new memory test, or the conditional Remember/Know memory test. Second, the randomization feature only supports the generation of arithmetic problems that do not include accuracy feedback. Third, when both old/new and Remember/Know responses are collected for one item, only the latter is stored in the regular data file, requiring additional effort if one also wants to analyze the former responses. Finally, performance feedback is not available beyond trial-based accuracy messages.

E-Prime

Installation and Interface

The E-Prime software package consists of five modules: Experiments are created with E-Studio and run with E-Run; three additional tools are available to merge and analyze data, and to recover lost data. E-Prime is copy-protected by a hardware key system: To use the software, a hardware key that is available for the USB or parallel ports has to be plugged into the system. Installation of the software as well as saving a newly created or modified experiment is possible only if the key's presence is detected by the software. Once saved, an experiment can be run on an unlimited number of computers. Figure 2 presents the E-Prime window with the LDT experiment definition file loaded.

Stimulus and Response Options

E-Prime registers responses via keyboard, mouse, and external response boxes connected to the serial port, as well as TTL signals from the parallel port. Multiple options can be realized for each trial. A new version of the E-Prime package is announced for the near future that is expected to include support for presenting video stimuli, presenting stimuli on multiple monitors, and recording vocal responses.

Creating an Experiment

E-Prime's approach to reducing the complexity of the task of programming a computer experiment strongly parallels the approach taken by modern computer programming environments like Borland's Delphi. In these environments, instead of writing each line of code by hand, programmers can add components and features to their program by selecting a corresponding icon from a toolbar and dropping it at the desired location. The program code required for the component or feature is added automatically in the background. The component's available options and attributes can be specified via the visual interface as well so that, typically, a much smaller portion of the program's source code has to be created manually.

The user interface of the main program, E-Studio, resembles such a visual programming environment (see Figure 2). Upon startup, three windows are presented onscreen: the Structure, Toolbox, and Properties windows. The first window shows a hierarchical tree view of the experiment's components. These are arranged onto a line representing the time axis. A new experiment consists of an empty time line. To create an experiment, components (e.g., stimuli or blocks) are added to the time line. The available components are listed in the Toolbox window. From there, a selected component (e.g., a text stimulus component) is dragged onto the time line and dropped at the desired position. The properties of the currently selected component (e.g., the text to be presented; its font, color, position, and alignment; as well as timing and response options) can be viewed and edited in the Properties window.

Experimental blocks are implemented with the List component. A list basically consists of a table, with columns representing the factors and rows representing the levels of the factor. Within one list, a single factor or multiple factors can be realized; alternatively, multiple lists can be nested to realize multiple factors. In addition to specifying experimental blocks, lists can also be used to specify trials. In that case, each row represents a trial, and the columns represent the trials' attributes. A time line is added to each list onto which the subordinate components are then placed. For example, a trial list's time line would contain a stimulus component presenting a fixation cross as well as the prime and target stimulus components. The attributes defined in a list component can be used as variables within the subordinate time line. To present a different word on each trial, the set of words is added to a trial list, and the column containing the words is passed as a variable to the text stimulus component located on the subordinate time line.

A Wizard feature is available to guide through the steps of the generation of simple choice reaction time paradigms. Ideally, those experiments (e.g., simple sequential priming or recognition memory tasks) can be created entirely from the drag-and-drop interface. For these, the E-Prime user does not need to learn any set of commands or write a line of code. At times, however, one will want to implement more complex designs, and those might not be supported by the drag-and-drop interface. A unique feature of the E-Prime package is that in addition to (or as an alternative to) creating an experiment via the visual interface, one can use the integrated programming language E-Basic (that is similar to Visual Basic, a widely used programming language) to add to or modify the experiment. This is supported by E-Prime in three ways. First, one of the objects that can be dropped onto the experiment time line, the Inline object, is a container for custom E-Basic program code (see Figure 3 for an example). Thus, a stimulus, response, or computation (e.g., a complex stimulus selection or matching procedure) that is not supported by E-Prime's drag-and-drop components can be added by including the necessary program code as an Inline object. Second, custom E-Basic program code can be packaged in special files from which the code can directly be executed. This is especially useful for reusing procedures or components in multiple experiments. Finally, E-Prime experiments are stored as E-Basic source code files that are compiled into an executable program by the E-Run module for each session anew. Thus, an experiment generated within E-Prime and saved to disk as source code can be modified and cus-

tomized before execution by a Basic programmer. To assist with the interface and with writing custom program code, a User's Guide and an extensive reference for the E-Basic programming language is provided in electronic and printed form.

Running an Experiment

On the experimenter's computer, an E-Prime experiment can be executed from within the E-Studio experiment generation program by clicking the Run button or by pressing the F7 key. On participants' computers, an experiment is executed analogously from within the E-Run program or from the command line. Beyond stimulus and session number, E-Prime can be configured to collect any other required participant information before the start of an experiment, for example age, gender, or handedness. Pressing the Windows key or the Alt + Tab keys results in switching to other programs' windows or to the Windows Desktop. The execution of an E-Prime experiment terminates with an error in that case.

Data Output

E-Prime creates two types of output files for a session. In a plain-text file, it keeps a human-readable log of all stimuli and responses. The same data is written to a data file that serves as an input to the E-Merge and E-DataAid modules. E-Merge provides a drag-and-drop interface to combine individual sessions' and/or participants' data into a single data file that can subsequently be opened with E-DataAid to discard irrelevant variables, filter cases, compute a variety of initial statistics, and convert data into formats readable by general-purpose statistics packages.

Figure 2. The E-Prime program window showing the LDT session time line and trial list.

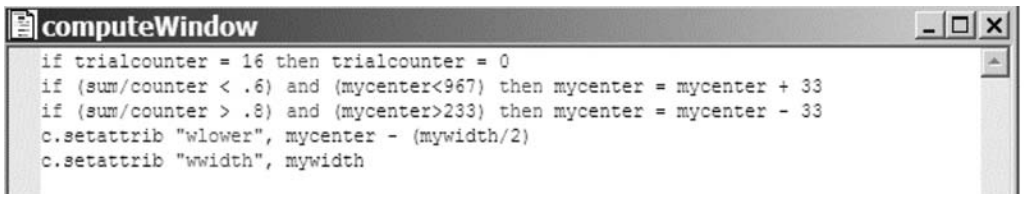

Figure 3. An example Inline object with E-Basic code.

Implementing Test Experiments

Remember/Know Paradigm

On the session time line, list components representing experimental blocks were added for the instruction, presentation, retention interval, and memory test blocks. The instruction list consists of a factor with two levels, one of which is randomly drawn, resulting in one of the two instructions being presented by a text stimulus component located on the associated time line. The presentation list consists of 30 levels containing the word stimuli, all of which are presented in a random order. The retention interval list contains one dummy trial with three placeholder values for the two numbers to be multiplied and the result. The dummy trial is repeated until the end of the retention interval; its duration is specified as time-out for the retention interval block. A small piece of program code is inserted to randomly generate the to-be-multiplied numbers for each trial anew. Feedback on accuracy on the response as well as on mean accuracy is given, and the numbers and accuracy data are written to the data file for each trial. The memory test list contains the 30 target stimuli along with the 30 distractor stimuli; the order of test trials is randomized. An inline object contains a line of custom program code that skips the execution of the Remember/Know trial after a *new* response.

Lexical Decision Paradigm

The Lexical Decision task is quickly created by adding two instruction pages and a top-level list object to the session time-line. The top-level list object represents the between-subject factor; two levels are created for the two conditions. A time-line is added to the top-level list object; then, a second list object is dropped onto that time line and populated with 80 levels representing the trials. Two attributes are added to the list object as containers for the prime and target stimuli, and an additional attribute is added for each level of the between-subjects factor to specify the correct response for each trial. The prime-target combinations are created manually and entered into the list object via copy and paste.

The trial time line contains the fixation, prime, masks, and target stimuli, as well as the response window stimuli. Although E-Prime does not provide any response window feature, a fixed or adaptive response window can be implemented nevertheless. To that end, a dummy stimulus is added to register responses after the end of the target stimulus presentation. The target stimulus component is set to register responses only until the start of the response window, but remains on-screen. If a response occurs within that period, execution continues with the presentation of a *too fast* message. If no fast response occurs, a transparent screen is presented as a dummy stimulus for the entire response window period to register participants' responses. If a response occurs within the window, execution continues with the presentation of a *hit* stimulus. If no response occurs within the window, a *too slow* message is presented. Responses that occur within the window are considered for computation of participants' accuracy. Accuracy feedback is given at the end of each trial.

To implement a fixed response window, the window onset time is specified as the target stimulus duration, and the window width is specified as duration of the response window stimulus. Adaptation of the response window can be accomplished via custom program code that keeps track of accuracy and modifies timing values of the target and response window stimuli such that the desired response window behavior results (see Figure 3 for the E-Basic code that computes the response window parameters).

In summary, all objectives can be accomplished within E-Prime. A few lines of custom program code are needed to implement the generation of random arithmetic problems and the conditional Remember/Know trial. Additional dummy and feedback stimuli are needed to simulate a fixed response window feature. By adding a few lines of custom program code, an adaptive response window was successfully implemented.

Inquisit

Installation and Interface

In addition to the program file, a help file and three sample experiments are installed. Upon startup, the Inquisit program presents itself as an empty editor window. Via the Tools menu, comfortable testing options for the components of the active experiment are available, as well as options to select stimulus font and color settings, and to modify and test settings for parallel and serial port communication as well as speech recognition. The Help menu provides access to the command syntax reference and links to the Millisecond homepage and sample experiments provided there. Figure 4 shows the Inquisit program window.

As a unique feature, Inquisit can also assist with data collection on the Web. A plug-in for the Internet Explorer Web browser is available that presents Inquisit experiments on participants' computers (note that although the Internet Explorer Web browser is widely used, the plug-in's restriction to that Web browser might result in biased data because potential participants that have no access to that Web browser are excluded). The collected data can then be stored on a remote server or sent to the experimenter by e-mail. A data encryption option is offered that should always be used to avoid compromising the confidentiality of data transmitted via e-mail.[6]

Stimulus and Response Options

Keyboard and mouse are supported as input devices, as well as joystick and touch screen. Two parallel and two serial ports of the computer can be used to interact with external hardware via TTL or serial port signals. The optional speech recognition unit selects the best match for a vocal response from a set of words specified as valid responses. Vocal responses are registered through a standard microphone or headset plugged into the sound adapter. In-

[6] The author wishes to thank an anonymous reviewer for pointing this out.

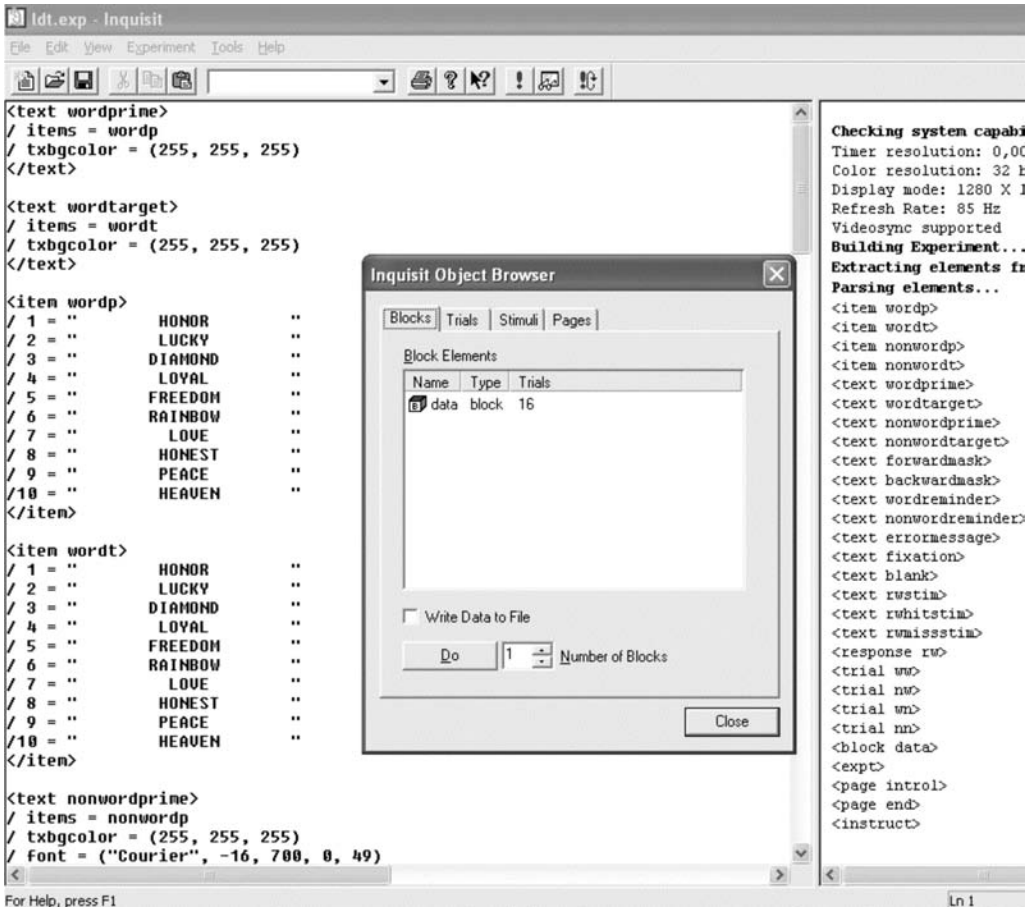

Figure 4. The Inquisit program window showing the LDT experiment definition and the Object Browser.

quisit does not provide an option to store vocal responses for later analysis.

Creating an Experiment

Inquisit addresses the complexity issue by specifying a reduced set of commands that is used to define experiments. The set is smaller and defined on a more abstract level than a programming language yet is not as simple as the list format used by DirectRT. Regarding complexity and structure, an Inquisit experiment definition file can be compared to an HTML[7] file. Experiments are generated as a hierarchy of objects, with experiment, block, trial, stimulus, and item levels. Objects on a higher level act as containers for lower level objects and can define attributes for the objects they contain. On each of the levels of the hierarchy, different elements can coexist. For example, multiple experiments can be defined within one file, with each one executed for a different subset of participants. Elements on the block, trial, and stimulus levels are not strictly associated to one higher level element but can be used by multiple higher level elements. For example, an item object (e.g., defining a set of stimulus words) can be used by multiple stimulus objects, with each stimulus object presenting the same set of words in a different color. Similarly, the same stimulus object can be used by multiple trials; a trial, once defined, can be included in multiple blocks, and one block can be part of multiple experiments.

Randomization with and without replacement, as well as sequential sampling, is supported for items of a stimulus, stimuli within a trial, trials within a block, and blocks within an experiment. Item selection can be linked between two stimulus objects, thereby equating the order of selection for those two stimulus objects. In addition, *counter* variables can be defined containing a pool of values drawn in random or fixed sequential order to control stimulus selection, position of stimuli on-screen, or time-out.

The behavior of each object is controlled by a series of required and optional attributes, controlling, for example, the visual features of a stimulus or its presentation duration. A number of possible options for each object (e.g., a stimulus's font size, color, and position) can be defined on different levels of the hierarchy, with lower level definitions overriding higher level definitions. For example, font size and color can be defined on an experiment level (e.g., as 12 pt. black). Unless otherwise specified, all stimuli will be presented in that font size and color. If one stimulus is to be presented in a different color (e.g., an error message

[7] HTML is a simple markup language designed to create World Wide Web pages.

that is to appear in red), this can be defined as an attribute of that specific stimulus; all other stimuli will continue to be presented in black.

On the trial level, different objects support different response formats. For example, the Trial object is used for a fixed response trial, whereas the Openended object is used for trials with open-ended text responses. Likert scales and verbal naming responses are additional response format options. Inquisit provides an adaptive response window feature that is capable of adjusting response window center and width to participants' average response latency or accuracy. Alternatively, fixed response windows can be specified at the experiment, block, and trial levels.

Because text stimuli are limited to a single line of text, separate objects are available for presenting participants with instructions. With these, entire pages of either nonformatted plain text or HTML-formatted text can be presented.

Inquisit assists in defining objects and attributes through a comprehensive syntax reference that also includes some examples. The reference is context-sensitive: Pressing the F1 button presents information about the current object and its possible attributes. Before running an experiment, the software checks if the definition file is valid. A list of valid objects is presented in an adjacent window, and syntax problems are highlighted. If an experiment definition is valid, its component objects (blocks, trials, stimuli and pages) can be accessed and tested separately via the Object Browser. In addition, Inquisit provides a second useful testing mode: Via the Tools menu, the experiment can be administered to a computerized "monkey" that works through the entire experiment by giving random responses to all trials.

Objects and their attributes are defined as plain text in the editor window. Alternatively, any other text editor can be used to create and modify Inquisit experiment definition files, and the plain text format also facilitates defining experiment files automatically. For cases in which complex stimulus materials or randomizations that are not supported by Inquisit have to be prepared anew for each participant, the software provides the feature of including externally generated experiment definition files. For example, it is possible to define the part of an experiment that stays constant across participants (e.g., block and trial structure) in one file, and to define those parts that vary across participants in another file (e.g., the stimulus material). After being generated manually or automatically, the second file is then linked into the first file, thus completing the experiment. This also helps in reusing objects that remain constant across multiple experiments, for example biographical questions or debriefing information.

Running an Experiment

An experiment is executed on the experimenter's computer from within Inquisit or via a command line call. A participant's number is required for the experiment to start; it can be entered manually or specified in the command line.

During an experiment session, pressing the Windows key or the Alt + Tab keys results in switching to other programs' windows. Upon switching back to the Inquisit window, execution continues with a repetition of the last trial. Pressing the Ctrl + B keys skips to the next block, and pressing the Ctrl + Q keys skips over the entire experiment. These keys cannot be deactivated, and because these events are not written to the data file, one cannot tell if participants have pressed any of these keys. This may potentially compromise the data. Inquisit can be configured to lock the screen at the end of an experiment session to prevent participants' access to the computer.

Data Output

Data are output either as a fixed-width or comma-, tab-, or space-separated text file at one trial per line. The values to be stored, including actual millisecond onset of each stimulus's presentation, can be specified in the experiment definition file. Data files can be encrypted and can be written either to the hard disk, to a network share, an ftp or http server, or sent to a specified e-mail address.

Implementing Test Experiments

Remember/Know Paradigm

Two Experiment objects were defined for each condition, differing only with regard to the instruction for the presentation phase. The presentation block randomly samples 30 trials from the concrete and abstract word presentation trials. Each trial presents a stimulus word at onset that is sampled from the respective pool of items without replacement. The retention block is defined with a time-out of 300 seconds. The automatic generation of random arithmetic problems is not supported. Therefore, arithmetic problems are drawn randomly without replacement from a manually generated, fixed pool of 100 problems (the number was chosen to be large enough so that the fastest participant will not be able to solve all problems before the time-out is reached). For the test block, 60 trials are randomly sampled without replacement from the four trial pools of concrete target words, concrete distractors, abstract target words, and abstract distractors. Each of the trials presents a stimulus word along with the old/new question and two response key reminder stimuli. When an *old* response is registered, a special trial is executed that presents the Remember/Know question along with two response key reminder stimuli for this question. After a response is registered to a Remember/Know trial, or when a *new* response is registered to an old/new trial, the next old/new trial is randomly drawn from the four pools.

Some limitations were observed during the implementation. First, Inquisit does not randomly assign participants to conditions. It does, however, assign them to different conditions according to the modulus of their participant number. By manually selecting a random participant number before starting the experiment, random assignment to conditions can be achieved. Second, Inquisit does not generate random arithmetic problems. Thus, the arithmetic problems have to be created manually and added to the experiment definition file, thereby presenting the same randomly created set of problems to all participants.

Lexical Decision Paradigm

To create the Lexical Decision task, a sample affective priming experiment included in the installation is modified: the positive and negative words are replaced with words and nonwords, and instructions and response key reminders are adapted, as well as stimulus presentation times and interstimulus intervals and block length. For the word-word trials, *counter* variables are defined such that selection of items is linked between the prime and target sets so that the prime-target pairs are always presented on the same trial. For the other trials, *counter* variables are specified such that word and nonword items are selected randomly without replacement. Additional stimuli are defined for the fixation cross and the response window stimuli. A response window object is defined, and the start values for center and width are specified, as well as the stimuli to appear during the response window and in case a response occurs within the window. An adaptive increment of 33 ms was implemented for the window center if accuracy is to fall below 60 percent; similarly, a decrement of 33 ms was implemented if accuracy is to rise above 90 percent. Minimum and maximum window center were set to 250 ms and 1000 ms, respectively.

Both paradigms could be implemented, but some limitations were encountered. Randomized assignment of participants to conditions was not automated, and arithmetic problems could not be generated automatically, nor could the responses be checked for accuracy. At the same time, some features were noted that are only provided by the Inquisit package. Firstly, flexible performance feedback can be given within instruction pages by including the desired performance variable. And secondly, Inquisit is the only package that includes an adaptive response window feature.

SuperLab

The current release of the SuperLab package does not include critical features necessary to implement the test paradigms. Therefore, no comparison with the other packages could be made. Specifically, SuperLab fails to collect open-ended responses and provides no support for the presentation of trials conditional upon participants' responses. An upcoming new release of the software is to include both of these features. Here, an overview of the current release of the SuperLab package is offered.

The main window presents a list for blocks, trials, and events (see Figure 5). Those are the components a SuperLab experiment is made of: a block serves as a container for trials, and a trial serves as an event container. When creating a new block, one has to specify a name and whether the order of this block's trials is to be randomized. After creating and naming a new trial, one or multiple events can be associated with it. Event types include presentation of text (of any font, size, or color), pictures, and sounds, as well as interstimulus intervals, output via the computer's serial port, and control via a serial response box. Event settings can be tested via the preview function that is available for each stimulus type. For each event, a number of input options are available. First, it can be specified whether an event accepts user input, and whether that input is to be recorded. Events can be set to terminate when the criterion response is registered, after a specified timeout, or when either one occurs. A correct response is defined experiment-wide as a set of response keys (e.g., "y" and "Y"). Responses can be registered via standard keyboard or mouse, as well as from a wide range of external response boxes that can be connected to the computer's serial port. For each experiment, however, only one input device can be used.

For each event, different feedback trials can be specified for one, some, or all of five cases: occurrence of a correct, an incorrect, or a delayed response; and occurrence of a reaction time less than a specified constant or greater than a (second) constant.

A new trial can be specified as a feedback-only trial. In this case, it is not presented with the list of other trials of that block but is only shown if it is specified as a feedback trial to some event (e.g., a trial presenting the *Error* string would be presented among the other trials of a block unless it is specified as a feedback-only trial to occur when an incorrect response is registered). Trials are associated with blocks, and events with trials by a mouse click on the marker next to the trial or event name. When a block is selected, the trials belonging to that block are marked. Similarly, when a trial is selected, the events associated with that trial are marked.

As mentioned above, the input device must be specified for the entire experiment and cannot vary between blocks or trials. Other settings that are fixed across the entire experiment are the screen's background color, the response key sets, and trial codes. Trial codes represent the dependent variables manipulated in the experiment, and its values represent factor levels. Each trial can be assigned to one level on each factor, and those assignments are written to the data file along with the responses, which helps with subsequent data analysis.

By providing a simple point-and-click user interface and a clear block-trial-event framework for experiments, SuperLab facilitates the creation of experiments that fit the given structure. This will be the case for many common experiments. The advantage of a simple experiment framework brings with it a lack of flexibility. Designs that do not fit the framework, for example, because they require more than one input device, different input devices for different blocks, different screen backgrounds for different blocks, or even conditional trial execution are not easily accommodated. While it is possible to simulate conditional trial execution to a certain extent on the trial level, and, for a skilled programmer, it is possible to extend the SuperLab package with custom input modules or events, SuperLab hardly reduces complexity in these cases.

Some additional points reduce the usability of the package. First, the program window does not represent the experiment structure visually. Assignment of events to trials and trials to blocks is not visible in the interface, with the exception of the currently selected block or trial. In addition, trial and event lists presented in the program window quickly fill up. As a result, even with a small experiment

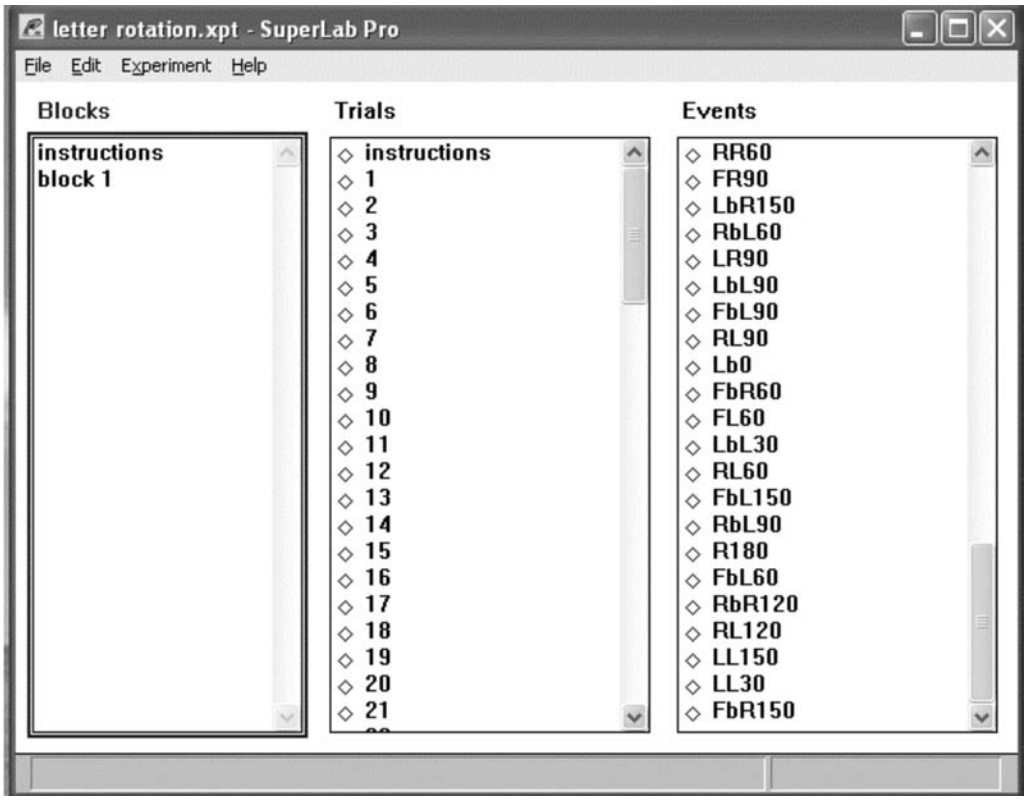

Figure 5. The SuperLab program window showing a sample experiment.

of less than one hundred trials, its structure is not made clear in the program window. Secondly, order of blocks, trials, and events cannot be changed once they are created. Order of creation of blocks (and trials, unless randomized) thus determines the order of their execution. Also, order of creation of events determines order of occurrence of stimuli in trials. This can only be circumvented by manually modifying the experiment definition file in a text or spreadsheet editor. Finally, open-ended responses cannot be registered. This limitation is critical, and it renders SuperLab incapable of administering one of the tasks included in the test paradigms of this review. Therefore, in the remainder of this review, SuperLab is not considered further. A new version of the SuperLab packages has been announced for the near future. The company promises to address the mentioned points and will improve the usability of the package.

Summary

All of the three remaining packages are capable of presenting a wide range of text, graphic, and sound stimuli in linear order. Similarly, all packages are capable of registering participants' responses via keyboard and mouse. Video stimuli and joystick responses are also supported (E-Prime support has been announced for the upcoming new release).

All packages are based on Microsoft's DirectX technology, therefore accuracy of stimulus presentation and response registration can be assumed to vary little between packages. When interference by third party software is minimized, it can be assumed that timing accuracy largely depends on hardware factors (e.g., monitor refresh rate, keyboard buffer time).

All packages support some form of error feedback, yet they differ in the flexibility they offer. While in Inquisit and E-Prime, any stimulus can be defined as an error message, DirectRT restricts the message to plain text. However, with the help of the conditional branching features incorporated in Inquisit, DirectRT, and E-Prime, this restriction can be circumvented. The conditional branching feature allows different trials to be executed depending on participants' responses to a preceding trial. Inquisit also supports control over the sequence and the number of repetitions of blocks and trials depending on participants' performance. Within E-Prime, the above restrictions can be overcome by inserting custom program code.

All packages feature the basic randomization procedure of selecting items from a pool with or without replacement. More complex, conditionally random sequences are generally not supported or not facilitated (i.e., the complexity of generating such a sequence is not reduced by the package). However, all packages support incorporating externally prepared stimulus material into the experiment.

Detailed performance feedback is supported by Inquisit and E-Prime. Inquisit features block- and experiment-wise feedback of a number of latency and accuracy statistics. E-Prime also supports basic latency and accuracy statistics; in addition, the desired feedback can be generated with the help of custom program code.

An adaptive response window feature is provided by Inquisit. E-Prime supports the definition of response deadlines, and an adaptive response window can be implemented with the help of custom program code. A fixed response window can be implemented in DirectRT.

Discussion and Conclusion

Among the reviewed packages, DirectRT achieves the most complexity reduction. A simple experiment is most easily implemented with DirectRT's simple list structure syntax, essentially reducing the task to specifying the stimulus item, its location, and presentation time in a text editor. The other packages require generating longer and more complex definitions (Inquisit), or first becoming acquainted with a multiwindow visual user interface (E-Prime). Therefore, DirectRT would be the package of choice where assistance with standard paradigms and a quick start with experimenting is the major objective. Also, DirectRT provides two unique response options: recording key release times, and a voice key option including saving vocal responses for later analysis. However, although chances are that a feature request will be fulfilled promptly, DirectRT is not easily extensible and thus not able to accommodate all experimental paradigms.

Among the reviewed packages, E-Prime retains the most flexibility. The possibility of including custom program code at any point in the course of events allows for a wide range of possible extensions beyond what is already provided by the package. E-Prime would be the choice for researchers who do not hesitate to write an occasional line of Basic program code and who wish to implement all or almost all experiments within one package. With E-Prime, one can most likely avoid encountering the necessity of learning an additional programming language; whereas users of other packages will not be able to avoid programming the more complex experimental paradigms using other programming environments and languages. In addition to providing a maximum of flexibility, E-Prime also strongly reduces the complexity of generating simple experiments with its drag-and-drop visual user interface. It allows the generation of standard paradigms without requiring any programming knowledge and has been successfully used in undergraduate and graduate courses (MacWhinney, James, Schunn, Li, & Schneider, 2001).

In between, Inquisit is positioned as a compromise option. Although it takes some time to get used to the more complex definition syntax, implementing the first experiment is still quite easily accomplished due to the available samples for a number of standard paradigms, the context-sensitive reference, and helpful debug options and messages. While it is limited in the paradigms it supports, Inquisit provides more flexibility than DirectRT (e.g., response window options) and supports features not included in the other packages (e.g., speech recognition, World Wide Web experimenting).

The reviewed packages provide the most assistance in implementing standard paradigms of serial visual and acoustic presentation and registration of responses via keyboard, mouse, joystick, and response boxes. For these applications, they provide robust frameworks that reduce complexity of the programming task as well as the time needed for creating and debugging experimental programs. A majority of scientists will be able to implement a majority of their studies within these packages. Yet, some paradigms and features (e.g., those that employ complex counterbalancing or interactive generation of stimulus materials, randomization that must satisfy nonstandard constraints, complex response window procedures, interaction with nonstandard external hardware, custom performance feedback, or a sequence of events that is conditional upon participants' responses) are often not well supported.

In particular, it will be difficult to implement dual-task paradigms. In those paradigms, stimulus presentation and response registration occurs in parallel for the two tasks. For example, participants can be asked to press the space key when a sound is played to the left ear while at the same time providing word/nonword responses with the A and 5 keys. In this example, two responses can occur at the same time. Registering responses to two different tasks in parallel is not supported in the reviewed packages. As a consequence, one should still be prepared to implement some studies with programming languages like C++.

Alternatives to the reviewed packages do exist. For example, the exclusion of MS-DOS based packages is based on the fact that support for new hardware is potentially not provided, but does not imply any judgment regarding ease of use and flexibility of these packages. They still provide valuable service to a lot of researchers using a variety of experimental paradigms, and the fact that these packages can run reliably on older computers can be considered an advantage.

Psychophysical research requires special support for a wide range of visual and acoustic stimuli, for adaptive testing, stimulus adjustment by participants, and calibration of output devices. These special requirements are not met particularly well by the reviewed packages. The Psychophysics Toolbox (Brainard, 1997), an extension to the Matlab package, provides these functions and is thus a better alternative for psychophysics researchers (for an overview of software for psychophysics researchers, see Strasburger, 2004).

Another set of alternatives is available for experiments for the Web platform. A class of Web-based software for data collection is currently being developed that does not require installation on participants' computers (e.g., Express, Yule & Cooper, 2003; SurveyWiz and FactorWiz, Birnbaum, 2000; WEXTOR, Reips & Neuhaus, 2002). These packages provide assistance in creating Web-based experiments that can also be run in the lab (for a review of Web experimenting tools see Reips, 2002.

Some of the mentioned software packages can also be used as a teaching tool (MacWhinney et al., 2001; Reips & Neuhaus, 2002). Other software packages that can be used for teaching are reviewed by Ransdell (2002).

In this article, four software packages for creating psychological experiments were reviewed. Although one cannot expect the reviewed packages to completely release researchers from any programming needs, by choosing a

package that works best with the paradigm used in most of his or her studies, an experimental psychologist can expect to reduce the effort necessary for programming and debugging and to lower the hurdle for conducting a first experiment.

Acknowledgments

The author wishes to thank Jörg Beringer of Berisoft, Stacie Jarman of Cedrus Software, Blair Jarvis of Empirisoft, Sean Draine of Millisecond Software, and Amy Eschman and Cindy Carper of Psychology Software Tools for providing evaluation copies of the software packages and for their assistance in implementing the test experiments, and Karl Christoph Klauer, Rainer Leonhart, Sarah Teige-Mocigemba, Andreas Voß, and two anonymous reviewers for helpful comments on an earlier version of this manuscript.

References

Birnbaum, M. H. (2000). SurveyWiz and FactorWiz: JavaScript Web pages that make HTML forms for research on the Internet. *Behavior Research Methods, Instruments, & Computers, 32,* 339–346.

Brainard, D. H. (1997). The psychophysics toolbox. *Spatial Vision, 10,* 433–436.

De Clercq, A., Crombez, G., Buysse, A., & Roeyers, H. (2003). A simple and sensitive method to measure timing accuracy. *Behavior Research Methods, Instruments, & Computers, 35,* 109–115.

Direct RT (Version 2004.1.0.55) [Computer program]. New York (http://www.empirisoft.com): Empirisoft.

E-Prime (Version 1.1) [Computer program]. (2004). Pittsburgh, PA (http://www.pstnet.com): Psychology Software Tools.

Forster, K. I., & Forster, J. C. (2003). DMDX: A Windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers, 35,* 116–124.

Inquisit (Version 2.0.41230.0) [Computer program]. (2004). Seattle, WA (http://www.millisecond.com): Millisecond Software.

MacInnes, W. J., & Taylor, T. L. (2001). Millisecond timing on PCs and Macs. *Behavior Research Methods, Instruments & Computers, 33,* 174–178.

MacWhinney, B., James, J. S., Schunn, C., Li, P., & Schneider, W. (2001). STEP—A system for teaching experimental psychology using E-Prime. *Behavior Research Methods Instruments, & Computers, 33,* 287–296.

Myors, B. (1999). Timing accuracy of PC programs running under DOS and Windows. *Behavior Research Methods, Instruments, & Computers, 31,* 322–328.

Plant, R. R., Hammond, N., & Whitehouse, T. (2002). Toward an experimental timing standards lab: *Benchmarking precision in the real world*. *Behavior Research Methods, Instruments & Computers, 34,* 218—226.

Ransdell, S. (2002). Teaching psychology as a laboratory science in the age of the Internet. *Behavior Research Methods, Instruments, & Computers, 34,* 145–150.

Reips, U.-D. (2002). Standards for Internet-based experimenting. *Experimental Psychology, 49,* 243–256.

Reips, U.-D., & Neuhaus, C. (2002). WEXTOR: A Web-based tool for generating and visualizing experimental designs and procedures. *Behavior Research Methods, Instruments, & Computers, 34,* 234–240.

Strasburger, H. (2004). *Software for psychophysics: An overview*. Retrieved August 31, 2005, from http://www.lrz-muenchen.de/~Hans_Strasburger/psy_soft.html.

SuperLab Pro (Version 2) [Computer program]. (2004). San Pedro, CA (http://www.cedrus.com): Cedrus Corporation.

Yule, P., & Cooper, R. P. (2003). Express: A Web-based technology to support human and computational experimentation. *Behavior Research Methods, Instruments, & Computers, 35,* 605–613.

Appendix
Experiment 1. Remember/Know Paradigm

Design

A 2 (imagery vs. rote instructions) × 2 (concrete vs. abstract nouns) design with a between-participants manipulation of the first and a within-participants manipulation of the last factor was implemented.

Materials and Procedure

Fifteen concrete and abstract target nouns (as judged by the experimenter) were selected for presentation (Sets A and B). Additionally, 15 abstract and concrete distractor nouns were used in the memory task (Sets C and D). Instructions were to inform participants about the following presentation phase and, in the rote rehearsal condition, to instruct them to keep repeating the words silently to be able to remember them better in the subsequent memory test. In the imagery condition, participants were to be instructed to vividly imagine each item visually. Items were randomly sampled without replacement; presentation duration was 3,000 ms.

In the retention interval between the presentation and the test phases, participants were to solve simple arithmetic problems for a fixed time of 5 minutes. The problems were multiplications of randomly created numbers between 1 and 10. The solution of each problem was to be typed into the computer as an open-ended text response. Participants' responses were to be checked for accuracy, and accuracy feedback was to be given. The number of solved problems as well as the number of correctly solved problems was to be recorded to assess participants' compliance.

The memory test instructions were to introduce participants to the memory test procedure as well as to the mnemonic states calling for a *remember* versus a *know* response. The 30 target items and the 30 distractors were to

be mixed and presented in random order without replacement. In the parallel test version, the item was to be presented along with the three response options: *remember*, *know*, and *new*. In the sequential test version, the item was to be first presented with the two response options *old* and *new*. Upon an *old* response, the next screen was to display the Remember/Know question along with the response options *remember* and *know*; whereas upon a *new* response, the next item was to be presented. At the end of the memory task, accuracy feedback was to be given.

Experiment 2. Lexical Decision Paradigm

Design

A 2 (Set A vs. Set B) $\times$ 2 (related vs. nonword primes) x 2 (word targets vs. nonword targets) design was implemented with repeated measures on the last two factors.

Materials and Procedure

Twenty pairs of semantically related words were constructed by the experimenter. One word from each pair was randomly assigned to Set A1, the other was assigned to Set B1. Similarly, 20 pairs of nonwords were constructed and divided into two sets, A2 and B2. In Condition A, each word from Set A1 was to appear as a target twice—once preceded by its Set B1 partner as prime, and once by a randomly selected nonword from Set B2. Similarly, each nonword from Set A2 was to be presented as a target twice—once preceded by a randomly selected Set B1 word, and once preceded by a randomly selected Set B2 nonword. Thus, in Condition A, each word and nonword from Set B was to appear twice as a prime—once for a word from Set A1, and once for a nonword from Set A2. Analogously, in Condition B, Set B words and nonwords were to appear as targets twice, once with a Set A1 word and once with a Set A2 nonword as prime. In total, 80 trials were to be presented in five blocks of 16 trials each.

Participants were to be instructed to decide whether a letter string is a word or not. A *word* response was to be given with the "A" key, and a *nonword* response was to be signaled by pressing the "5" key. In the fixed time-out condition, they were to be instructed that a response has to be given before a time-out of 700 ms. In the response window condition, instructions were to require that participants answer after a signal appears but before it turns to red. If a response was to fall into the response window, the signal was to turn green for 300 ms. The response window was to initiate at a center of 500 ms and a width of 400 ms, thus ranging from 300 ms to 700 ms. If possible, the center value was to be increased adaptively by 33 ms if accuracy for the last block was to fall below 60%, and decreased if it was to rise higher than 90%.

Each trial was to start with a fixation cross presented for 1,000 ms. A string of 10 *X*s was to appear as a forward mask for 70 ms. Subsequently, primes were to be presented at the center of the screen for 35 ms and were to be masked by a string of *X*s for 70 ms. At 500 ms after prime onset, the target was to be presented at the same location and to remain on-screen until a response was registered or until time-out, whichever occurred first. At window onset, a "!" sign was to be presented. In case of a response within the window, it was to turn green and remain on-screen for another 300 ms, or else the "!" sign was to turn red and remain on-screen for another 300 ms. Accuracy feedback was to be presented for 500 ms after each trial. At the end of each block, participants were to be informed about their mean reaction times and error rates.

Christoph Stahl

Institut für Psychologie
Albert-Ludwigs-Universität Freiburg
D-79085 Freiburg i. Br.
Germany
Tel. +49 (0)761 203 2418
E-mail stahl@psychologie.uni-freiburg.de

