Behavior Research Methods, Instruments, & Computers
1997, 29 (3), 322–327

EXPE: An expandable programming language for on-line psychological experiments

CHRISTOPHE PALLIER
Rutgers University, New Brunswick, New Jersey

and

EMMANUEL DUPOUX and XAVIER JEANNIN
Laboratoire de Sciences Cognitives et Psycholinguistique, Paris, France

EXPE is a DOS program for the design and running of experiments that involve the presentation of audio or visual stimuli and the collection of on-line or off-line behavioral responses. Its flexibility also makes it a useful tool for the rapid design of protocols for testing neuropsychological patients. EXPE provides a powerful scripting language that allows the user to specify all the components of an experiment in a human readable file. Subjects' responses are saved in a user-specified format as well as in readable ASCII files. The user can easily add new commands to the language: All the instructions are calls to functions written in independent Borland Pascal units. Thus, users can link their own Pascal procedures to EXPE to meet virtually any special need. This makes it possible, for example, to adapt EXPE to new hardware, such as new sound or video boards.

When one designs a psychological experiment using computerized equipment, one can use a general-purpose programming language (C, Pascal, etc.), or one can use one of the software/freeware packages developed specifically for psychological experiments. The first solution is more common, but it has several drawbacks. Neither C nor Pascal is very well suited to describing experiments: Both force the experimenter to focus on many irrelevant details. More importantly, these languages require a level of programming skills that cannot be expected from the average user, especially if precise timing or synchronization is needed. The second solution, the use of specialized systems, has the advantage that such systems are simpler and more accessible, especially with systems that provide a graphic interface for the construction of experiments. However, such software is often tied to specific kinds of experiments, and it imposes relatively strict limits on experimental design, stimulus presentation, feedback, and so on. For example, it is generally impossible to modify the format of the feedback or to adapt the number or type of trials to the history of subject responses. Another serious problem with these programs is that they are not open and are often tied to a special-purpose hardware (e.g., response buttons/audio boards).

In this paper, we describe a program that we developed 5 years ago at the Laboratoire de Sciences Cognitives et Psycholinguistique (LSCP, Paris) to try to cope with these problems. Our aim was to design a system that had both the power and expandability of programming languages and the ease of use of the more specialized software. Power was achieved through a scripting system with general-purpose data and control structures. Expandability was achieved through a modular, open architecture that allows the easy addition of new functions through Borland Pascal modules. User ease was obtained by designing the language with a maximally simplified syntax and many primitive functions tailored to the needs of experimental testing (trial definition, stimulus presentation, response collection, timing, data saving, etc.).

Thus, EXPE is an interpreter of scripts that is similar in spirit to BASIC. It provides variables, expression evaluation, and all the necessary control structures (including subroutines); the BASIC functions Read and Data have been enhanced in many ways to allow a compact and legible description of experimental design (more on this subject in the section on EXPE's scripts). We want to stress, however, that EXPE is not a high-level experiment generator: Concepts of experimental blocks or groups are not "hard-wired" primitives of the language. Thus, for example, EXPE will not automatically generate a Latin square design; the user will have to specify the different groups of subjects and the block order. Nevertheless, most EXPE scripts are usually quite short, and our experience at the LSCP has shown that students can very quickly learn to use it to design and run experiments. Indeed, EXPE has been used extensively in the past 4 years to prepare and run auditory and visual experiments. It has also been used to test neuropsychological patients.

The hardware requirements have been kept to a minimum: EXPE is a DOS real-mode program that can run on

This article was written while C.P. was supported by a LAVOISIER grant from the French ministry of foreign affairs and then by a grant from the FYSSEN foundation. The audio functions rely on modules programmed by Xavier Jeannin at the LSCP in Paris, and by John Mertus at Brown University. Correspondence should be addressed to C. Pallier, LSCP, 54 bd Raspail, 75006 Paris, France (e-mail: pallier@lscp.ehess.fr).

old PCs with only 640K of memory (though some experiments may require a faster PC to handle the presentation of complex audio or video events). No complex hardware is required for timing and response-time measurements; the response buttons are simple mechanical switches that can be connected directly to the parallel port of the computer. The audio functions rely on Bliss audio drivers, a system designed by John Mertus at Brown University. This makes EXPE compatible with audio boards for which there exists a Bliss driver;[1] at the LSCP, we use MediaVision's ProAudio Spectrum 16.

The paper is divided into two parts: We describe (1) the script language and try to convey how easy it is to program new experimental paradigms with it, and (2) the expandability of the language through the linkage of Pascal units.

EXPE SCRIPTS

Two Examples With Commentary

Figure 1 shows the script of a simple phoneme categorization experiment: The subject listens to a list of stimuli, and after each one, has to press the key corresponding to the perceived phoneme (this is just a simple example; a real experiment would likely include the instructions for the subject, a training block, and maybe feedback).

The core of many experiments is a series of trials that differ only in terms of the actual stimuli used in each particular trial. It is convenient to separate the code (i.e., the commands to be executed in each trial) from the data (the description of the stimuli). In Figure 1, the first block of lines is the code section; the commands enclosed between the "WithData ... LoopData" construct will be executed once for each line of data (defined by the "Data ... EndData" construct). Within the "WithData ... LoopData" loop, #1 refers to the first column of the current line in the data block and hence varies on each pass. So, in the first pass, #1 corresponds to "stim1.adf"; in the second pass, it corresponds to "stim3.adf," and so on, and #2 corresponds to the second column. If the "database" "Materials" contains 60 lines, the instructions are executed 60 times.

In this example, the whole experiment consists of a single block of uniform trials; each trial starts by the playing of the auditory file, the name of which is given in the first column in the current data line ("Listen #1"). Then, the file's name, the expected answer, and the subject's response are saved together in the result file ("Save #1 #2 ReadKey"). Finally, the computer waits 2 sec before starting the next trial ("Wait 2000").

```
WithData "Materials"
  Listen #1
  Save   #1 #2 ReadKey
  Wait   2000
LoopData

Data "Materials"
  stim1.adf P
  stim3.adf K
  stim7.adf D
  stim2.adf P
  ...
EndData
```

Figure 1. Example of a phoneme categorization script.

Once this script is written, say, in the text file "phondec.pro," entering "expe phondec subjcode" on the DOS command line will launch the experiment. When the experiment is completed, the raw results will be immediately accessible in an ASCII file with four columns: the subject code, the file name, the expected response, and the actual response. Information about the subject and about the date and time of the run is also stored in the results file. It is easy to extract statistics from the results file with programs like Count, Mystat, and Anova, provided in the EXPE package; alternatively, the ASCII file can be imported into a spreadsheet or a statistical program.

In data blocks (databases) containing lists of stimuli, it is good practice to add some information about the category of the stimulus to each line. This information can then be saved with the response of the subject, facilitating easy extraction of the results. It is important to note that databases are not limited to storing stimulus lists. The format inside a database is completely free, and the action taken depends on the code section.

For example, we conducted an experiment in which subjects had to perform a click detection while listening to auditorily presented words. Every 5 to 12 trials, a recognition test was administrated to force subjects to pay attention to the words. In order to achieve this, lines with a special code were interspersed throughout the stimuli list to instruct EXPE to present the subject with an occasional recognition test. An instruction in the main loop branched between a click detection or a recognition test.

The database mechanism is quite powerful. The Result File itself is actually a database, so it is possible to access the history of subject responses within the experiment (e.g., to compute the mean reaction time [RT] in a block). Also, a given script can have several databases, which can be scanned sequentially or in parallel: WithData ... LoopData loops can be embedded. For example, one database can contain the name of others; this feature can be used to store the order of experimental blocks. The outer loop controls which list is used, and the inner loops scan through this list.

A second example of an application of EXPE is the script of a speeded phoneme detection experiment (Figure 2). First, the instructions are displayed on the screen. Then, the order of stimulus presentation is randomized (for each subject) with the instruction "Shuffle." Inside the loop, the instruction "RTtrial 2000 Listen" plays the stimuli and monitors the keyboard and the parallel port for a button- or keypress during 2 sec (2,000 msec). The RT can then be read with the function "RT," and the code of the button pressed is accessed with the function "Button" (if no button has been pressed, the "Button" returns "~"). The entire data line is saved, along with the response by the command "Save #0 button rt." Saving as much in-

```
Echo "                       INSTRUCTIONS FOR EXPERIMENT"
Echo "In this experiment, you will be presented first with a target phoneme"
Echo "on the screen, then by a spoken word."
Echo "Your task is to press the right button if the word contains"
Echo "the target phoneme, or the left button if the word does *not* contain it."
Echo
Echo "Respond as soon as you have heard the target phoneme."
readkey

Shuffle trials              ;randomizes the trial order

WithData trials
  Wait 1000                 ;wait one second
  Cls                       ;clear the screen
  WriteXY  38 12 #1         ;display the target phoneme
  Wait 1000                 ;wait another second
  Cls                       ;clear the screen
  Wait 500                  ;wait 500ms
  RtTrial 2000 Listen #2    ;make a timed response trial with
                            ;deadline 2000ms
  Save   #0 button rt       ;save the trial characteristics and responses
                            ;feedback for slow and wrong responses:
  If button=='~'  WriteXY 20 20 "Too Slow !"   ;no button press
  If button!='~' and (button!=#3)  WriteXY 20 20 "Wrong Response"
LoopData

Data trials   ;target-phoneme stimulus-file desired-response
  P    sheep.adf 1
  M    glop.adf  2
  D    disk.adf  1
  K    pump.adf  2
  ...
EndData
```

Figure 2. Example of a phoneme detection script.

formation as possible about each trial makes later analyses easier.

In a given script, several blocks of codes and data can be interspersed freely. Complex designs can be made by embedding WithData... LoopData structures at any depth. Conditionals and control structures allow for feedback or trials to depend on the subject's response. A WithData... LoopData loop can be interrupted before the end of the list (e.g., if the subject's performance has reached a certain criterion). The next section briefly describes the capacities of the language.

EXPE Syntax

EXPE syntax is deliberately minimalist. A program consists of a series of lines, each of which can be a command line, a data line (appearing between "Data" and "EndData"), or a comment (introduced by a percent sign). Typically, command lines are executed sequentially, one after another, in the order in which they are encountered. However, some commands allow looping over a block of instructions until a condition is met (we have already mentioned WithData... LoopData, but there is also While... EndWhile and For... EndFor). As in most programming languages, there also are conditional instructions (If... Then... EndIf) that allow execution of a given instruction only when a specific condition is met.

Each command line starts with a command, which may be followed by one or several arguments that can be numbers, strings of characters, variables, or other commands. Assignment (":="), standard arithmetic ("+," "−," "/," "*"), and logical ("and," "or," "xor") operations are special types of commands that take their arguments on their left and right sides. The standard operator precedence applies (i.e., $3 + 4 * 5 = 23$). Commas for arguments and parentheses for expressions are not mandatory (minimalist syntax), but they can be used to make complex expressions more legible.

Contrary to many programming languages, the user does not have to pay attention to the argument type: Conversions between numbers, strings, or Boolean expressions are performed implicitly as needed ("0" is equivalent to 0.0 and to False; however, improper conversions [e.g., "3 + hello"] will generate a run-time error).

EXPE also has variables that can be manipulated just as easily as constants. Again, variables have no particular type and can be used as arguments or functions requiring strings or numbers. Variables are especially useful for counting subjects' good and bad responses. Several of our experiments involve adaptative training where the training block ends when a certain performance criterion has been reached. As noted, inside a WithData ... LoopData loop, "#i" refers to the "*i*th" item on the current database line. In brief, EXPE takes care of type checking, memory allocation, input–output, and so on, and allows the user to concentrate on the most important aspect of the research—the execution of the experiment.

Functions for Stimulus Presentation

We now turn to the specifics of EXPE concerning audio/ video output and timing. The simple command "Listen <filename>" allows an audio file to be played. There is no limit on the size of audio files; EXPE reads them from the disk and feeds them to the audio board in real time. Currently, EXPE handles 16-bit Bliss.adf mono and stereo audio files, raw PCM linear 16-bit files, and Windows .wav pcm 8- and 16-bit files. The amount of time it takes to load the file will depend on the hard drive on the computer. If playout must start at a very precise time, it is possible to decompose this command in "Load <filename>" and "Play." "Load" will fill the audio buffers with the beginning of the file and "Play" will start the playout immediately (fetching more data on the disk if necessary).

EXPE is not limited to presenting only an entire, prerecorded audio file; it also provides a general audio mixing table that allows the user to specify millisecond-synchronized sequences of auditory events. The mixing occurs in real time and allows the user to overlap or gate stimuli or to play some files over the left and others over the right channel. Figure 3 shows a simple example of a stimulus made of two audio sounds played simultaneously, one in the left channel and the other one in the right channel and amplified by a factor of 2. Only the first 1,000 msec of each file is played.

By default, the mixing is done in real time. If this exceeds the capacities of the machine (because the hard drive or the CPU is too slow), it is possible to mix the sounds off-line (i.e., before starting playback) in a temporary file that can be played afterward.

The current video functionalities are relatively primitive; the basic graphic functions of Borland Pascal are accessible, as is a command to display pictures stored as bitmaps in TIFF files. The drawing is done in a hidden page that can be swapped rapidly with the page shown on the screen, allowing measurement of RT from the onset of the presentation. However, it is not yet possible to present a visual stimulus in the middle of the presentation of an audio stimulus; we are planning a revised version of EXPE in which visual events will be incorporated in the audio mixer, allowing for the possibility of arbitrarily complex sequences of audiovisual events.

```
Defitem
 audio "file1.adf"  0   Length 1000ms Gain 1.0 LEFT
 audio "file2.adf"  0   Length 1000ms Gain 2.0 RIGHT
Enditem
Playitem
```

Figure 3. Mixing two audio files.

With this basic set of instructions, EXPE can handle many current psycholinguistic or neuropsychological experiments that specify a relatively simple sequencing of events. However, experiments requiring the rapid succession of many visual frames (e.g., rapid serial visual presentation) or complex synchronization of audio and video events (cross-modal priming) may not be possible with the basic functions provided. When such a limit is reached, we recommend programming each particular complex stimulus presentation directly in Pascal and linking it to EXPE code (see the next section, about expanding the language).

EXPE can also communicate with other hardware or computers through the parallel port or the serial port. For example, for an evoked-potential study, we have been able to link through the parallel port a PC handling the presentation of audio stimuli to another PC recording the EEG. The first PC, programmed with EXPE, sent codes that were specific and synchronized with each stimulus to the second computer. In other experiments (head turning), we were able to use the serial port to drive a videotape recorder. The serial port has also been programmed to control a digital audio tape (DAT) recorder during a naming experiment (with the DAT recording the vocal responses of the subject).

Timing and Responses

Control on intertrial times, feedback duration, and so on, is achieved by the functions "Clock," "Wait," and "WaitTill." "Clock" gives back the time elapsed since the beginning of the experiment. "Wait <n>," Wait n msec," and "WaitTill <n>" will make the computer wait until the clock reaches the value *n*. For example, to program a series of trials of equal durations—say, 4 sec—"A: = clock" can be set at the beginning of the trial, and "WaitTill a+4000" can be put at the end.

Two types of functions are designed to obtain functions input from the subject—those that record off-time responses (ReadKey, ReadString, etc.) and those that measure on-line RTs. The latter can be achieved with external buttons linked to the parallel port (with millisecond accuracy) or through the keyboard (with less accuracy). For RTs the simplest function is "RTtrial <deadline> <command>." The RT is measured from the real onset of command (i.e., the first sample output in "Listen" or video retrace for video functions). The functions "RT" and "Response" can then be used to access, respectively, the value of the RT and the button (or key) pressed. If no button is pressed before deadline, RT is set at 0. It is sometimes useful to decompose the action of "RTtrial" into more elementary instructions ("e.g., EnableResponse," "RTonset := getonset <action>," "LatchResponse") to allow more control in the case of complex sequences of events. Proper connection of a voice key to the parallel port enables one to use vocal instead of manual responses.

Saving Results

The data recorded by EXPE (subject responses) are saved using the Save command. This command saves all its arguments, in plain ASCII format, in a memory area that is dumped onto the disk when the experiment ends. User interruptions or run-time errors are caught, and the data are saved on disk before the program exits. Results can be logged onto the same large result .res file or onto separate files for each subject. In addition, much other information regarding the experiment is saved in the .res file (date, time, experiment duration, type of machine, version of EXPE, etc.). Note that functions that perform the saving, like all other EXPE commands, can be replaced to use different formats. We now turn to an important feature of EXPE—its expandability.

EXPANDING EXPE

Hardware and software limitations of any experimental package are quickly reached by ingenious researchers. As an example of hardware limitation, the user already may have hooked up 10 computers with a particular kind of response button, but the hardware does not allow for reading out these buttons. EXPE has been programmed in a modular fashion, making adaptations to new hardware possible. The pieces of code dealing with the hardware are kept in separate Borland Pascal's units, which means that adapting EXPE to new hardware involves only local changes to these units. For video output, we rely on Borland's BGI drivers. For the audio output, we use the specifications of John Mertus's Bliss drivers, and, again, EXPE's code is device independent. Adapting EXPE to new audio boards would require writing up a new Bliss driver (5–6 very low level functions).[2] Finally, timing and response buttons are also modular parts of the code. In short, it is possible to adapt EXPE to new input boards or ports. For example, we have used it to command a robot arm (moving objects in a theater designed for baby experiments) and to synchronize a NeuroScan EEG recorder with auditory stimulus presentation.

Software limitations concern, for example, the types of graphic file format the package can display, or the type of audio file it can play. With most experimental software, the experimenter has no recourse if the package cannot accommodate a particular format. With EXPE, the experimenter can add commands to the language very easily. As long as the user has access to the required Pascal procedure, it is easy to link this routine with the other functions, resulting in an extension to EXPE's language.

Consider some examples. To save the results in a special format adapted to the analyzing tools being used, a special-purpose Save command can be added to the language. Similarly, it is easy to link a modular piece of code that achieves a special effect (e.g., self-paced reading) to EXPE.

All of EXPE's commands are in fact implemented as functions of the type "function myfunction:xvalue;far;" where "xvalue" is a hybrid type that allows for transmission of numbers or strings of characters. In order to add new functions to the EXPE language, one must do the following:

1. Write a Borland Pascal unit containing the code for the new function.
2. Add a line like the following to the initialization section of the unit:

   NewFunc("MYFUNC",myfunction,short-help).

   ("NewFunc" is part of the interface of a unit called "xcore," which must be included by the unit.)
3. Add the name of the unit to the list in file named "Module.pas."
4. Run the batch "Make" that invokes bpc to recompile EXPE.

Then, at run time, the keyword "MYFUNC" is a new command in the language that can be used within EXPE scripts to call for the desired procedure. It is possible, in a given unit, to add many other commands; the only thing needed is a "NewFunc" statement for each of them. In a nutshell, *EXPE is a scripting interface to independently written Pascal functions.*

The arguments in the script can be accessed by using the functions "GetStrArg," "GetIntArg," or "GetBoolArg." These read the next argument in the current line and convert it to the relevant Pascal type. The user need not consider syntax, expression evaluation, and so on, since the core of EXPE takes care of such details.

In brief, when the user has written code into an independent Borland Pascal unit,[3] he/she just has to add the name of this module to the list in the file modules.pas and to run "Make." This solution has the advantage of allowing the user to distribute his/her contributions as independent .tpu files. Useful contributions could be incorporated in the standard distribution of EXPE.

Availability

EXPE can be freely downloaded from the Internet at the URL http://www.ehess.fr/centres/lscp/expe/expe.html. Researchers using EXPE for their experiments are only requested to cite the present paper whenever they report on their work. We provide, along with EXPE, documentation and some tools for preparing experiments and performing data transformations and statistical analyses (including analyses of variance). The full package is self-contained and is sufficient to prepare, run, and analyze many psychology experiments.

CONCLUSION

EXPE provides an expandable and powerful scripting language for experiment construction. Although we do not wish to review all existing experimental packages, the most powerful of these packages—MEL (Schneider, 1988) and PsyScope (Cohen, MacWhinney, Flatt, & Provost, 1993) also have underlying scriptlike languages. In these packages, however, such scriptlike languages are machine oriented and are normally not programmed directly by the user. Rather, a menu-driven or graphics interface guides the user in the specification of the experiment. Such specification is used to generate the appropriate script code, which is then fed into the experiment engine. This approach has the advantage that users do not need to do any programming to implement an experiment. Typically, however, the complexity and range of designs or protocols that such packages can express is limited. For instance, the interface may not allow the user to modify the format of the feedback messages or to schedule trials that depend on the history of subject responses. Indeed, the user interfaces are often more restrictive and less powerful than the underlying scripting language. When the user tries to surmount these limitations, he/she has to abandon the user interface altogether and cope with an unfriendly machine-oriented scripting system.

EXPE lacks a "high-level" interface: Users cannot enter, in menus or through a graphic interface, parameters such as stimulus onset asynchrony or numbers of trials in a block. On the other hand, the scripting language has been designed to be human readable and to minimize the burden of programming. Still, the fact that using EXPE requires a basic understanding of instructive programming may be a drawback for a certain class of users. Also, unlike MEL and PsyScope, EXPE does not include a library of scripts for common experimental paradigms.

In summary, EXPE has the following advantages:

1. It provides specialized commands and structures for stimulus presentation and response recording. The audio functions are especially powerful, allowing for the playing of files of unlimited size and for mixing several files in real time (which is useful, e.g., for dichotic experiments).

2. Users do not have to learn to use a compiler, as they do with languages such as C or Pascal. EXPE is much more similar in spirit to BASIC, and is thus simpler to master than C or Pascal. Users can write scripts on their favorite word processor, and, at any time, they can get on-line help on EXPE's commands by calling "expe -?".

3. EXPE is open ended: If necessary, "power users" can add new functions by programming them in Borland Pascal and linking them to EXPE.

4. EXPE can be used without restrictions: The license allows users to make an unlimited number of copies.

REFERENCES

Cohen, J., MacWhinney, B., Flatt, M., & Provost, J. (1993). PsyScope: An interactive graphic system for designing and controlling experiments in the psychology laboratory using Macintosh computers. *Behavior Research Methods, Instruments, & Computers*, **25**, 257-271.

Schneider, W. (1988). Micro Experimental Laboratory: An integrated system for IBM PC compatibles. *Behavior Research Methods, Instruments, & Computers*, **20**, 206-217.

NOTES

1. Bliss drivers and tools are available on-line (ftp://jam.cog.brown.edu).

2. Writing up a new Bliss driver, however, is not necessarily an easy task. The structure of Bliss drivers requires the use of assembly language. And the programmer must not be afraid of dealing with the DMA and other capricious beasts.

3. It is also possible to link C code to Pascal, but with an important restriction: No functions of the C run-time library can be used. Future use of dll may solve this problem.

(Manuscript received November 13, 1995;
revision accepted for publication June 17, 1996.)