EXPE: an Expandable Programming Language for On-line Psychological Experiments

Christophe Pallier
Rutgers University Center for Cognitive Science

Emmanuel Dupoux and Xavier Jeannin
LSCP, EHESS–CNRS, Paris

This is a preprint a paper published in *Behavior Research Methods, Instruments and Computer*, 1997, 29((3)), 322-327. Consult the latest paper for up-to-date information about Expe 5.1

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 makes it also a very useful tool for the rapid design of protocols for testing neuropsychological patients. EXPE provides a powerful scripting language which 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, also in readable ASCII files. A remarkable feature of EXPE is that 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 any special need. This makes it possible, for example, to adapt EXPE to new hardware, such as new sound or video boards.

When one has to make up a psychological experiment using computerized equipment, there are basically two choices: a) use a general-purpose programming language (C, Pascal, etc.), or b) use one of the special software/freeware packages made for psychological experiments. The first solution is the most general one, but it has several drawbacks. Neither C nor Pascal is very well suited to describe experiments: They 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, using specialized systems, is 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 imposes rather strict limits on experimental design, stimulus presentation, feedback, etc. 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 (response buttons/audio board).

In this paper, we describe a program that was born five years ago at the Laboratoire de Sciences Cognitives et Psycholinguistique (LSCP, Paris) from our efforts 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 was as easy to use as the more specialized software. Power was achieved through a scripting system with general-purpose data and control structures. Expansibility was achieved through a modular and open architecture allowing 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 which is similar in spirit to BASIC. It provides variables, expression evaluation, all the necessary control structures and even 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 it is not a very high level experiment generator in the sense that concepts of experimental blocks or groups are not "hard wired" primitives of the language. Thus, for example, it 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 use it to design and run experiments. Indeed, EXPE has been used extensively in the past four 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: it is a DOS real-mode program, which can run on old PCs with only 640K of memory (though some experiments may require a faster PC to handle the presentation of complex au-

Correspondance address: Dr. C. Pallier, Center for Cognitive Science, Rutgers University, New Brunswick, NJ 08903. Email: pallier@ruccs.rutgers.edu C. Pallier was supported by a post-doc fellowship from the Fyssen Fondation; part of this work was also realized while he was a post-doc at the University of Barcelona supported by a Lavoisier Grant from the French Ministry of Foreign Affairs. The audio functions rely on modules programmed by Xavier Jeannin at the LSCP in Paris, and by John Mertus at Brown University.

Figure 1. Example of a phoneme categorization script

```
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
```

dio 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 the Bliss audio drivers, a system designed by John Mertus at Brown University. This makes EXPE compatible with audio boards for which there exist a Bliss driver[1]; at the LSCP, we use MediaVision's ProAudio Spectrum 16.

The paper is divided into two parts: in the first, we describe the script language and try to convey how easy it is to program new experimental paradigms with it. In the second part, we describe the most unusual aspect of EXPE: the expandabilty of the language through the linkage of Pascal units.

Expe Scripts

Two examples with commentary

In Figure 1, we have provided 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 (it 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 consists of a series of trials where the only differences are the actual stimuli played 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. #2 corresponds to the second column. If the "database" "Materials" contains 60 lines, the instructions are executed 60 times.

In the 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 two seconds before starting the next trial ("wait 2000").

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 it 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 are also stored in the result file. It is easy to extract statistics from the result file with programs like "count", "mystat" and "anova" provided in the EXPE package; alternatively the ascii file can be imported into a spreadsheet or in a statistical program.

In data blocks ("databases") containing lists of stimuli, it is good practice to add on each line some information about the category of the stimulus. 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 five to twelve 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 in 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

[1] Bliss drivers and tools are available at ftp://jam.cog.brown.edu.

Figure 2. Example of a phoneme detection script

```
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
```

file" itself is actually a database so that it is possible to access the history of the responses of the subject inside the experiment (for example to compute the mean reaction time in a block). Also, in a given script, there can be 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 can be used to store the order of experimental blocks: the outer loop controls which list is used, and the inner loops scans through this list.

As a second example, we propose the script of a speeded phoneme detection experiment (see 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 "rt-trial 2000 listen..." plays the stimuli and monitors the keyboard and the parallel port for a button or key press during 2 seconds (2000 msec). The reaction time can then be read with the function "rt", and the code of 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 rt button". It is a good idea to save as much information as possible about each trial: it makes things easier for later analyses.

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 interupted before the end of the list, for example if the subject's performance has reached a certain criterion. The next section gives a quick description of capacities of the language.

EXPE's syntax

EXPE's syntax is deliberately minimalist. A program consists in a series of lines where each 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 seen “withdata...loopdata” but there is also while...endwhile, for...endfor). As in most programming languages, there also are conditional instructions (if..then.endif) that allow execution of some instruction only when a condition is met.

Each command line starts with a command, optionally 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. 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 are performed implicitly as needed (”0” is equivalent to 0.0 and to False; however, improper conversions (e.g. `3+"hello"`) generate a run-time error.).

EXPE also has variables which can be manipulated just as easily as constants. Again, variables have no particular type and can be used as arguments of 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 described above, 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, etc., 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 bits Bliss .adf mono and stereo audio files, raw PCM linear 16 bits files, and Windows .wav pcm 8 and 16 bits files. Depending on the hard drive on the computer, loading the file will take a variable amount of time. 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 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 play some files over the left and others over the right channel. Figure 3 gives 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 2. Only the first 1000 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), into a temporary file, which can be played afterwards.

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 which can be swapped rapidly with the page shown on the screen, allowing measurement of reaction time 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 plan a forthcoming version of EXPE in which visual events will be incorporated in the audio mixer, allowing for the possibility of arbitrarily complex sequences of audio-visual events.

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. rapide 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).

Another feature worth mentioning is the ability of EXPE to 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 one recording the EEG. The first PC, programmed with EXPE, sent to the second computer codes that were specific and synchronized with each stimulus. 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 recorder during a naming experiment (the DAT recording the vocal responses of the subject).

Timing and Responses

Control on inter-trial times, feedback duration, etc., is achieved by the functions “clock”, “wait” and “waittill”. “clock” gives back the time elapsed since the beginning of the experiment. “wait <n>”, waits “n” msec, and “waittill <n>” waits until the clock reaches the value “n”. For example, to program a series of trials of equal durations, say 4 sec,

Figure 3. Mixing two audio files

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

"a:=clock" can be set at the beginning of the trial, and put "waittill a+4000" at the end.

Turning now to the functions that get input from the subject, one can distinguish those recording off-time responses (readkey, readstring, etc.), and those measuring online reaction times. The latter can be achieved with external buttons linked to the parallel port (with millisecond accuracy), or through the keyboard (with less accuracy). For reaction times, the simplest function "RTtrial <deadline> <command>". The reaction time is measured from the real onset of <command> (i.e., 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 reaction time and the button (or key) pressed. If no button is pressed before deadline, RT is set to 0. It is sometimes useful to decompose the action of "RTtrial" into more elementary instructions ("EnableResponse", "rtonset := getonset < *action* >", "LatchResponse") to allow more control in the case of complex sequences of events. By properly connecting a voice key to the parallel port, vocal responses can be used instead of manual ones.

Saving Results

The data recorded by EXPE (subject responses) are saved using the `save` command. It saves all its arguments, in plain ASCII format, in a memory area which is dumped onto the disk when the experiment ends. User interruptions or runtime 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 capacity for extension.

Expanding EXPE

The limitations of any experimental package are quickly reached by ingenious researchers. Limitations are of two kinds: hardware and software.

As an example of hardware limitation, the user may already have hooked up 10 computers with a particular kind of response button, but the package 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 only involves 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 implies writing up a new BLISS driver (5-6 very low-level functions).[2] Finally, timing and response button 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), or to synchronize a NeuroScan EGG 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, there is nothing the experimenter can do if he wants to use a format not accommodated by the package. In contrast, EXPE allows new commands to be added to the language very easily. Assuming that the user has the source of a Pascal procedure that does all the necessary work, it is a trivial matter to link this routine with the other functions, resulting in an extension to EXPE's langage.

Consider some examples. If one wants to save the results in a special format adapted to his analyzing tools, a special purpose "save" command can be added to the language. If one has a modular piece of code achieving a very special effect (e.g., self-paced reading), one can link it to expe without too much trouble.

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

1. write a Borland Pascal unit containing the code for this function.

2. add to the initialization section of the unit a line like the following:
`NewFunc("MYFUNC",myfunction,short-help).`
("NewFunc" is part of the interface of a unit called "xcore", which must be included by the unit).

[2] However, this does not mean that this is 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. 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, keyword "MYFUNC" becomes 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 in order to add them to the language, is a "NewFunc" statement for each of them. In a nutshell, *EXPE is a scripting interface to independently written Pascal functions.*

The function written by the user can access the arguments in the script 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. Syntax, expression evaluation, etc., need not be taken into account since the core of EXPE takes care of such details for the user.

In brief, when the user has written his/her own 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 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://ruccs.rutgers.edu/p̃allier. Researchers using it 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 perform data transformations and statistical analyses (including Anovas...). The full package is self-contained and is sufficient to prepare, run, and analyze many psychology experiments.

Conclusion

EXPE provides an expandable, intuitive, but powerful scripting language for experiment construction. Although we do not wish to review all existing experimental packages, the most powerful of these packages (MEL, PSYSCOPE) also have underlying script-like languages. In these packages, however, such script-like 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.

However, it is typically the case that graphic or menu-driven interfaces have intrinsic limitations in the complexity and range of designs or protocols that they can express. 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. Yet, when the user wants to get around these limitations, he/she has to abandon altogether the user-interface 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 SOA, 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, in comparison with the above-mentioned packages, it should be mentioned that a library of scripts for common experimental paradigms is not available.

In summary, here are what we conceive as advantages of EXPE:

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, for example, for dichotic experiments).

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

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

4. It can be used freely: The license allows users to make an unlimited number of copies.

[3] There is also the possibility 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.