MEDIA REVIEW
A Comparison of Two Programs for the Control of Behavioral Experiments

Reviewed by W. Jeffrey Wilson
Department of Psychology and Neuroscience Program, Albion College, Albion, MI 49224

Two of the most widely used programs for the control of behavioral experiments are Med Associates' MedState Notation and Coulbourn Instruments' Graphic State 2. The two systems vary considerably in their approach to programming and data recording, with Graphic State 2 using a point-and-click interface that appeals to non-programmers while MedState Notation requires the typing of programming code. Graphic State 2 provides many data analysis routines, while MedState Notation allows the user to embed simple data analysis within the behavioral protocol. Graphic State 2 is simpler to use, but MedState Notation is more versatile.

Software designed for the control of behavioral experiments can facilitate research when it is well-designed and easy to use. I will contrast two systems designed for this purpose, in the hopes of providing a perspective that will allow the novice researcher to find a system that will work best for his or her purposes. Please view the following as the observations of an experienced behavioral scientist and user of both systems, but not a technical expert on either.

My experience in programming behavioral experiments began with my undergraduate experience in the laboratory of Earl Thomas at Bryn Mawr College in the 1970s, when relay racks and spaghetti wiring were the norm. I had had limited formal instruction in programming (anyone remember Focal?) but the logical nature of the task appealed to me, and I actually enjoyed rearranging the snap leads. Shortly thereafter Thomas' lab acquired a Logic Box (manufactured by Lehigh Valley Electronics) – a very clever device that incorporated the logic of several racks of relays into a small breadbox-sized console. Programming still involved rearranging leads, but the basic logical components of OR gates, AND gates, and flip-flops were prebuilt, so the overall task was considerably simplified.

In the 1980s my lab initially controlled its experiments using SKED, running on a DEC PDP-8. This was a fabulous language, and was the logical grandparent of the commercial systems that I will review. It was conceptually quite simple: the software could read inputs, could control outputs, and could keep track of time. Every fraction of a second the program would cycle through a list of "state sets," each of which could be thought of as a simple program. Each state within the state set would watch for an input (which might be a switch closure or the passage of a certain amount of time), and in response would increment counters, reset timers, and turn on or off various stimuli before going to a different state. Because conditional statements were allowed (e.g., "if Light=On and RightLever=Pressed then …" – not the actual language but it conveys its spirit) very powerful behavioral programs could be written. The major drawback to the use of SKED was the high cost of repairing the computer needed to run it; we were ultimately forced to stop using it because of this expense.

With the demise of our PDP-8 I decided to implement a SKED-like program on an IBM-PC clone (an 8088 processor that ran at 12 MHz). The reading of inputs and the control of outputs were accomplished via an interface similar to one described by Mangieri (1991) that was attached to the computer's parallel port. The program that I wrote in Borland's Turbo Pascal read the inputs, controlled outputs, and stepped through multiple state sets just as did SKED. The greatest difficulty in creating the program on the PC was timing: the internal clock was accurate to the nearest 0.055 s – and timing to the nearest 0.01 s was needed. I wrote a routine (based on Norton, 1985) that modified the computer's internal tone generator to serve as a timer to overcome this problem. My software worked very well, but was not documented in a manner that facilitated use by others, and was certainly not "dummy-proofed," in that an inexperienced programmer could easily write a program that would fail or worse yet damage the hardware. I blew up several power transistors in the several years that I used the system. See Wilson (1996) for a description of my system.

When it became clear that I wanted my students to be able to program the experiments, I opted for a commercial system. Med Associates' MedState Notation was based on Turbo Pascal (now Delphi) and so was very similar to the home-brewed system that I had used; this was the first of these two commercial systems that I tried, in 1999. A year later I purchased Coulbourn Instruments' Graphic State 2 (GS2) system. Herein I will compare the two from the standpoint of ease of use and versatility (versions evaluated: Med Associates' Med-PC for Windows [WMPC]; Coulbourn Instruments' Graphic State 2 ver. 2.002 – both have been upgraded since my purchase, but have not been changed in any fundamental ways that would cause me to alter my descriptions or conclusion). It will become clear to the reader that I favor one system over the other, but both have their advantages.

OVERVIEW OF THE SYSTEMS

Both the Med Associates and the Coulbourn Instruments systems can monitor multiple inputs and control multiple outputs. Both can run multiple experiments concurrently with ease. Both require a Windows-based system and really should be run on a dedicated computer

that is not burdened by being connected to a network. Both offer a runtime screen that provides information about the status of the current experiment. Both systems are designed to be used with proprietary interfacing equipment available from the manufacturer. The reader is urged to consult the manufacturers' web sites (www.coulinst.com and www.med-associates.com) for detailed descriptions of the products. I should note that I have received timely and helpful assistance from both companies when it was needed.

Med Associates' offering is a programmer's system. The user creates a text file consisting of statements written in MedState Notation (easily learned by a Pascal programmer, challenging for the user with no programming experience). This can be done using the editor provided with the system or with any editor or word processor capable of producing plain-text output. The program can contain as many as 32 distinct state sets, each acting as an independent "mini-program." Prior to use the program must be compiled, a process that will reveal coding errors that must be corrected before compilation can succeed. Once a program is successfully compiled it is available to be run from a pull-down menu within a separate Med Associates program.

MedState Notation allows data to be stored in 26 variables (labelled 'A' to 'Z'), each of which can take the form of either a single variable or a 1-dimensional array. These variables can store any data that the user specifies when the program is written. For example, one might contain the total number of times that a lever was pressed, another might contain a list of inter-response latencies, and a third might indicate the number of "correct" responses. Although limiting the total number of variables to 26 might seem constraining, I have yet to find this too restrictive, especially because arrays are allowed. The data are written to disk as a plain text file; several formats are available for the data file, and the one selected will probably be determined by whether or not the file will be imported into a spreadsheet.

Coulbourn Instruments offers a programming environment designed for the non-programmer. The programming window offers a picture of the front panel of the hardware that interfaces with the behavioral equipment. By pointing and clicking within this window the user can can specify the stimuli to be turned on during each state, the responses or time events that cause the state to terminate, and the target state(s) to be activated upon termination of the state. For example, to turn on the houselight for 10 s, the programmer would click on the houselight button to toggle it on, and specify that the state should end after 10 s. The program must be written via the Graphic State 2 programming interface, and the program is saved in a proprietary format that requires it to be viewed via Graphic State 2. Because of this it is impossible to print the program for archival purposes. Once the program is constructed, GS2 performs a check to ensure that it contains no states that are not called by at least one other state or that fail to call another state – a check that helps to ensure that the program is at least likely to terminate. Because the user writes no code, errors in programming syntax are far less likely than in Med Associates' system.

Graphic State 2 consists of a single state set; that is, only a single state can be active at any given time, as compared to MedState Notation's 32 concurrent states. Usually counters and timers are reset when a new state is entered, but the program allows them to retain their values across states or throughout an entire session.

Graphic State 2 logs all events while a session is being run, even if they have no effect on the program. If a state is ended by a response on Lever 1, for example, the program will also log responses on Lever 2 even if they do nothing. The data file that is created contains a record of every response or state change that occurred during the session, with a record of the time during the session when the event occurred and an indication of the cause of each state change. The programmer cannot specify any variables to be recorded or saved – instead a record of the entire session is available at its conclusion for analysis by Graphic State 2's extensive library of analysis routines or for exporting as a text file.

PROGRAMMING EASE AND VERSATILLITY

Without question, the easier of the two systems to learn to program is Graphic State 2. The point-and-click interface obviates the need to learn seemingly obscure programming language, and the fact that the program records everything that occurs during a session means that the user does not have to program variables to track relevant information. Indeed, a student with no programming experience watched me write a simple program one morning, and by the afternoon had mastered Graphic State 2 well enough to write a program to control a two-lever autoshaping procedure.

In comparison, MedState Notation requires text-based programming in a syntactically precise manner. It is true that the commands are reasonably intuitive (e.g., "on" turns on a stimulus), but they must be learned and they must be entered correctly. Furthermore, the user must specify and track all variables of interest; the program will automatically record the values of variables A through Z at the end of the session, but these values will be 0 unless the program contains code to update them (e.g., count responses in variable R, record number of trials in variable T, etc.). Users with no programming experience find the system somewhat intimidating.

In terms of versatility though, MedState Notation is superior to Graphic State 2. The availability of only one state set within a Graphic State 2 program makes some tasks either very difficult or perhaps impossible to program, especially those tasks that involve independent timing of various stimuli. For example, consider the truly random control condition in a Pavlovian conditioning study (Rescorla, 1967). The investigator wants a tone to turn on for 5 s and a feeder to operate for 0.1 s; the two should occur perhaps 20 times each during a 30-min session, but totally randomly with respect to each other. In MedState Notation each could be controlled by its own state set, without any reference to or input from the other (as if each stimulus was controlled by its own 1960s-era tape timer). In GS2, because a stimulus can be turned on only by entry into a state, and because only a single state set is available, the two stimuli must of necessity be controlled

together, and a complex program must be established consisting of states in which none is on, one is on, or both are on; the durations of these states will be varied and complicated. I think that the only way to implement this program in GS2 would be to create a fixed schedule of the two stimuli that simulates the random occurrence of both, then to program this fixed schedule. Different "random" schedules would require the generation of different simulations and then programming them. A program in which the two stimuli occur truly randomly with respect to each other might well be impossible in GS2.

DATA FILES AND ANALYSIS

The two systems differ considerably in their approach to data handling. GS2 creates a log of a session that contains every transition from one state to another (because stimuli are tied to states this corresponds to a record of stimulus presentations) and every response onset and offset (whether the responses cause state transitions or not). The result is a complete record of the session, available for export as a text file or for analysis within GS2 itself. Because the record of the session contains all responses that were available to be monitored the researcher can ask questions after the fact that were not anticipated at the time the session was run. A large number of routines are provided by GS2 that allow extensive analysis of the data, including frequency of entry into each state, number of responses within each state, inter-response intervals, etc. Unfortunately the explanations of these routines are far from intuitive, and the routines seem geared toward free-operant schedules. I run trial-based studies, often in runways or mazes, and the analyses have never been useful to me except as quick checks at the end of a session to ensure that data were recorded. I have relied on the exported raw data text files, which I then filter extensively and analyze with a spreadsheet program (e.g., Excel or Gnumeric).

MedState Notation's approach to data is far different: only the variables specified by the programmer are recorded; there will be no record of the number of responses if the program does not contain a counter that is incremented with each occurrence of the response. MedState Notation allows data to be saved in text files in a limited number of formats, some human friendly and some (supposedly) spreadsheet friendly. No analysis routines are included with the packages. In my experience the text-files must be manipulated extensively with a spreadsheet before useful analysis can occur. Med Associates offer an additional program called MPC2XL to ease translation from their data files to a spreadsheet; I have no experience with it.

Although both MedState Notation and GS2 data files could be made far more user-friendly in terms of how data are formatted for subsequent use by spreadsheet programs, I find MedState Notation's files easier to use. They are inherently smaller, because they contain only those data that I have requested. Because I can specify counters, timers, and other variables in the program, the data file contains the information that I want with very little further analysis, For example, some of my behavioral routines allow rats to control the onset of their own Pavlovian trials, and the number of trials initiated is a critical dependent variable. A variable within a MedState Notation program can record this number as the session runs, display it on the screen, and save it in the data file. In GS2 this variable would correspond to the number of times a particular state is entered; there is no way to display this, but instead an analysis must be run on the data at the conclusion of the session.

The ability to create user-defined variables that track meaningful information, and to display these variables as they are updated during the session, make a MedState Notation session far more informative than a session run in GS2. A well-designed MedState Notation screen can provide all of the data of interest at the end of a session. GS2 sessions are likely to require post-session analysis in order to get even the most rudimentary of data.

CONCLUSION

Med Associates' MedState Notation and Coulbourn Instruments' Graphic State 2 both offer the scientist useful tools for behavioral research. GS2's strengths lie in its user-friendly programming interface and its many analytical routines. For the non-programming scientist running operant studies GS2 might be a good choice. MedState Notation, with the availability of multiple concurrent state sets, offers a far more powerful programming environment. The time spent in mastering MedState Notation will be rewarded by the ability to program complicated procedures with ease. I believe that the serious behavioral scientist is better served by the Med Associates' product.

REFERENCES

Mangieri AM (1991) Build a parallel printer port I/O interface. ComputerCraft, pp 54-58, 76-79.

Norton P (1985) The Peter Norton programmer's guide to the IBM PC. Bellevue, WA: Microsoft Press.

Rescorla RA (1967) Pavlovian conditioning and its proper control procedures. Psych Rev 74:71-80.

Wilson WJ (1996) The □□maze: A versatile automated T-maze for learning and memory experiments in the rat. Behavior Research Methods, Instruments, & Computers 28:360-364.

Purchase of the systems described was supported with funding from Albion College's Neuroscience Program and Department of Psychology.

Address correspondence to: W.J. Wilson at wjwilson@albion.edu or 611 E. Porter St., Albion, MI 49224.