DEVELOPMENT ENVIRONMENTS

Computers and computer programs are becoming ever more sophisticated. This advancing technology has tended to make the need for programming obsolete. For instance, to write a manuscript, one simply pulls up one's favorite word processor like WordPerfect, Microsoft Word or Nota Bene and starts typing. Similarly, financial problems or models with graphical output are entered directly into spreadsheet programs like Lotus, Microsoft Excel or Quattro Pro. The very success of these programs seems to prove that programming is no longer necessary. Indeed, there are many applications today that need no programming to be used successfully. Nonetheless, the companies that developed the programs listed above realized that it is impossible for their programs to perform every imaginable task required by the user. If nothing else, the program developers themselves could not think of every possible problem. Therefore, each program has included a powerful macro language that allows the user to customize or to add almost any function one wishes.

Interestingly, the latest trend is to supply the user with more powerful rather than less powerful programming environments. For instance, anyone who has used Lotus version 2.1 realizes there are a wide variety of add-in programs, like *Always*, designed to extend Lotus because many needed features were not built in (Lotus has a special facility built in to add utility programs). One of the new features of Lotus version 3 is its powerful macro facility. Similarly, a powerful spreadsheet program is an Apple Macintosh product called WINGZ, which includes a comprehensive programming language that allows the user to change the program completely. In fact, the spreadsheet itself was programmed in that language. This programming facility is one of the reasons for the rave reviews earned by WINGZ. The new word processing program Word for Windows from Microsoft includes a programming language that uses BASIC-like commands quite similar to AxoBASIC. Almost 70% of the microcomputer database market uses dBASE, because of its built-in programming language. Companies have realized that no two databases are exactly the same; therefore, considerable programming is necessary when customizing an application.

AxoBASIC

You may be smiling to yourself right now, saying, "What has any of this to do with me? I type manuscripts in WordStar and have never changed any part of the program. I use spreadsheets like EXCEL to do graphics but, again, have never used macros. In fact, these programs have so many more features than I could ever use, I can't imagine doing any programming." Scientists indeed should not have to do programming of word processing or spreadsheet programs. If scientists need extendibility in any program, it is most likely in software for data acquisition and analysis used in conducting and analyzing experiments. A good word processor is nice but not critical; a good graphics program is quite important, but satisfactory programs are available (although users can generate graphs in AxoBASIC that even some outstanding graphics programs cannot duplicate). But the highest priority is to be able to perform the required experiments and to gather and analyze data. Scientists should not be limited by what their software programs can handle; consequently, some programming may be required to accomplish certain tasks. Hopefully, programming will be the exception rather than the rule. AxoBASIC allows users to create a set of programs that will be widely distributed and available to all. For instance, at present, there are whole-cell and single-channel programs available for the acquisition and analysis of voltage-dependent channels. These include programs for controlling membrane potentials and acquiring data from two independent patch clamps, as is done when studying gap-junction channels or recording synaptic currents from two neurons? A fura-2 program for use with a photomultiplier tube system is also available. There is an excellent set of programs for recording from tissue slices. Currently, programs for ligand-gated channels are not available, but they may become available soon. The programs are moderately complete, but they are also customizable for any particular application that the user requires. For instance, an ugly-looking screen can be changed. If one prefers different keys to change program parameters, they too can be changed. If the colors of the data on the screen are not drawing your eye to important events, change them. Everything can be changed or programmed. This is not to say that turn-key systems can never be as useful as programmable systems. If the turn-key system does all or most of what is required by a laboratory, then this is the system of choice. Turn-key programs are usually simple to learn; in most cases new users can be performing experiments on the same day. Nonetheless, *programs written for programmable systems can be taught just as easily*. Programming these systems takes a little longer, but almost all users find the extra capabilities worth the effort. In addition, users usually obtain a better understanding of the programs and, therefore, are able to add desired features and to discover program bugs.

Is programming in AxoBASIC too difficult or time consuming to be worth the effort? The answer to this question depends on how much of the user's requirements are satisfied by an existing program. If a turn-key system provides all or most of the needed applications, the benefits achieved by small improvements in the acquisition or analysis programs may not be worth the effort of reprogramming. But if you have an application that simply does not exist anywhere else, then a programmable, expandable environment is preferable. Or if you have seen one of the application programs available for AxoBASIC and have decided it's exactly right for your laboratory, then AxoBASIC is for you. For instance, fura-2 acquisition and analysis programs for use with a photomultiplier tube are available only in a few standard turn-key systems, each of which has certain undesirable limitations. For instance, one particular program can acquire only 30 minutes' worth of data at a time, whereas a typical continuous acquisition time should last 3 or 4 hr. In addition, these systems are very expensive. A set-up based on software written in AxoBASIC, or other programmable language environments like ASYST, can be built for about one third of the cost of commercially available software. Moreover, the user-programmable software can be customized within a short period of time to have precisely the features needed by the user. For example, the fura-2 setup took less than three days to get running (it's that simple!). The original programming took probably two to three weeks to get working. Since then the programs have slowly been improved; nevertheless, 90% of the programming was done in the initial two to three weeks. Now these fura-2 programs are available for general use. Another example is from a laboratory that couldn't find an ideal program for hippocampal slice work. The researchers wanted to record field potentials simultaneously with intracellular potentials. They wanted to divide their display into four separate windows, two showing raw data as it came in and two showing a running analysis of the experiments. Originally, the programs took quite a while to write. Now the slice programs have been used by several people for a couple of years. The effort of writing the program was more than offset by the increased efficiency of the laboratory using a program specifically designed for their needs. Was it difficult? The answer depends on the user; but many users are comfortable programming in AxoBASIC on their first day. Users will be writing sophisticated programs usually after two or three days.

If you have had any programming experience at all, AxoBASIC will be very easy to learn. Non-programmers should understand that the software was designed to be easy to use, much like the macro programming languages mentioned above. All of the difficult machine interfacing has been done for you. BASIC was chosen because it is the easiest language for novices to learn. New versions of BASIC, like AxoBASIC, which are based on Microsoft QuickBASIC, have all of the modern programming constructs such as the following structures: **IF...THEN...ELSE, WHILE...WEND, DO...UNTIL,** and **CASE SELECT...CASE ELSE...END SELECT**.

AxoBASIC has real subroutines, making it very modular. Line numbers can be used but are not required, nor are they recommended. In older versions of BASIC, if you needed to make a histogram from data residing in the computer memory, you would go to the part of the program that made histograms with a GOTO statement. A typical example would be GOTO 5000 (this jumps the program to line 5000). After finishing a typical program there would be many GOTO statements. This lead to a common BASIC condition known as "spaghetti code." The programs would be jumping all over the place, without any simple way of keeping track of where they were jumping. Two months after finishing a program, GOTO 5000 did not tell the user anything. This made programs difficult to maintain and hard to give to other laboratories who wanted to change the programs. AxoBASIC, being a modern language, does not have these flaws. The actual code for the histogram-making program would be in a small subroutine of no more than 100-200 lines (named, for instance, HISTOGRAM). If that subroutine did not provide enough space, one would write additional subroutines that would be called by the HISTOGRAM subroutine. Experience with the C programming language has shown that small, easy-to-understand subroutines are the simplest to maintain. Therefore, that programming model would be followed (some C programmers will limit subroutines to 50 lines of code, but that may be too restrictive). After completing these subroutines, one can generate a histogram anywhere in the program simply by typing in the word HISTOGRAM. Any time in the future, when one sees the word HISTOGRAM one will still understand what it means. And since the subroutine HISTOGRAM itself is small, stand-alone and fully functional, one will also understand how it works. In fact, BASIC now has all of the structured programming constructs of PASCAL or C, except for variable typing. (Variable typing refers to rules requiring variables to be declared before they are used. If the variable has been defined previously, the user will not be able to use an integer variable where a floating-point variable is required, which is a very common error in programming). BASIC programmers claim that the decision to exclude variable typing (as well as some other BASIC features) make programming simpler and more powerful; but there is not a complete consensus of opinion on this issue. Nevertheless, in recent years Microsoft QuickBASIC has won many awards from a variety of computer magazines as the most outstanding development environment available. For these reasons, AxoBASIC was originally based on QuickBASIC version 4. Currently, AxoBASIC is based on MicrosoftBASIC PDS (Professional Development System) version 7.1.

Is AxoBASIC an interpreted or compiled language? It is usually very painless and fast to develop programs in an interpreter, but such programs typically run up to 50 times slower than those compiled with a compiler. Interpreters usually translate lines of code into machine language as they are run. Compilers translate an entire program into machine language and then run the result. For example, the program listed below runs the program contained within the **FOR...NEXT** loop 100 times:

```
J  =  0
FOR I  =  1 to 100
J  =  J+I
NEXT I
```

This program adds every number from 1 to 100. A compiler would convert the entire program to machine language, whereas an interpreter would convert and execute each line as it is used. Consequently, for each of the 100 times through the loop, the interpreter will translate each statement into machine code, a very slow process, whereas the compiler will do it only once. Nevertheless, many programmers use interpreters because program development time is shortened significantly. The compile-link step in most compiler systems can be excruciatingly slow during program development, even in sophisticated environments that include a MAKE function. New in-memory compilers (like Turbo-Pascal or Quick C) have started to change the way we use compilers because they almost function like interpreters. Nevertheless, BASIC's interpretive environment will usually cut software development time significantly. Furthermore, because of the threaded interpretive software techniques used in QuickBASIC and the pre-compiled assembly language libraries used by AxoBASIC, programs typically run much faster than those written in Microsoft C. AxoBASIC has all of the advantages of an interpreter with none of the speed penalties, which makes it an almost ideal development environment.

Does that mean that programming in AxoBASIC is for everyone? Probably not. It is a matter of personal preference. Nevertheless, Axon Instruments' goal is to have a network of users linked together that trade application programs. Even users who enjoy programming realize that their priority is doing scientific research, not writing programs. Therefore, almost anyone would rather get a pre-written program than write it themselves. Of course, this does not preclude users from customizing their applications to perform their individual tasks — a process that fortunately is quite simple.

Programming in AxoBASIC is *simple*, right? How simple is simple? Below we have listed a few examples of actual code that would be useful in the laboratory. For programmers, it will look very simple. For the uninitiated, it may look complex, but it's not. For instance, anyone who has looked at the C> prompt in DOS or the blank WordPerfect word processing screen for the very first time feels somewhat lost. After a short period of time, users happily move around DOS or WordPerfect. We hope that AxoBASIC will provide a similar experience. And for anyone who has wanted to learn to program, this may be the most painless way.

A Programming Example

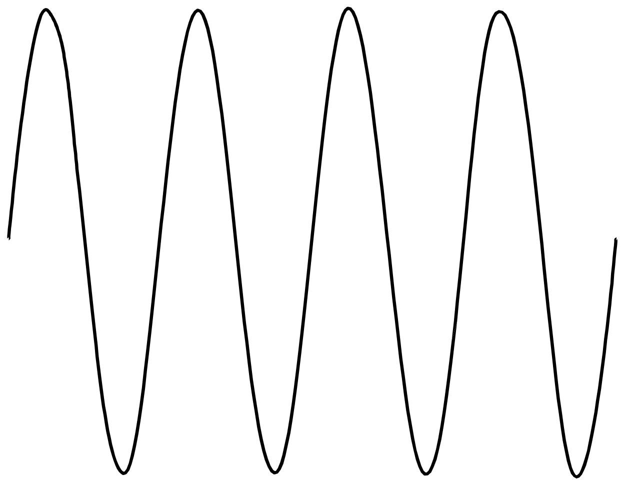
Turning the Computer into a Digital Oscilloscope

Most readers are aware of the usefulness of a digital oscilloscope. In general, when the scope is free-running, any signal brought into the scope is displayed on the screen. There is a scale on the screen allowing the user to measure the amplitude of the signal. At any point in time, the user can hit a button to hold the sweep on the screen. In another mode, the user can trigger the scope from some external event. At all times the user can change the gain and timebase.

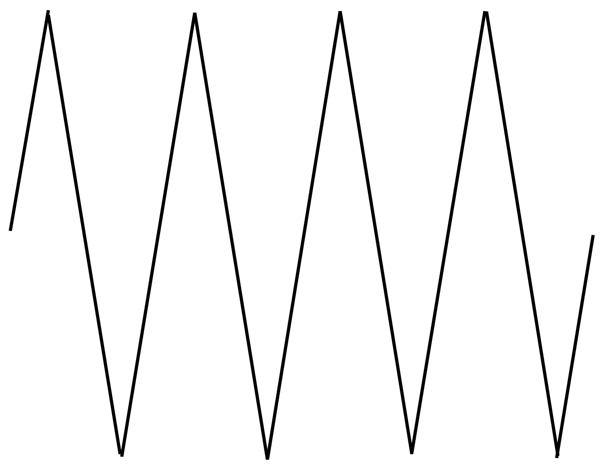
The entire program required for a free running scope (but with no display, no scale and no options) is the following:

```
TEST:
wipe                               'Clears the display screen of previously stored
                                   'data sweeps
samplel  50, 1000, 1, 0, 0, 1      'Digitize one channel of data, '1000 points at 50
                                   µs per point
disl  0, 1024                      'Display the data just acquired
GOTO  TEST
```

If a function generator producing a sine wave with a frequency of 80 Hz is connected to channel 0 of the ADC, we would see the following on the AxoBASIC VGA graphics monitor.



If we now changed the output of the function generator to a triangle wave, we would see the following on the screen.



So far this is pretty straightforward but not too useful. We have no way of changing anything or stopping the program. Next we will put in the options to hold the present sweep, put in a defined trigger pulse, acquire the next sweep and stop, free run, change the sampling rate, change the voltage gain, or exit the program.

```
 choices$ = "F"                        'Start the program free-running
 oldchoice$ = "F"
 scalefactor% = 1
 clock% = 50                           'Sampling clock 50 µs per point
 axisx -2048, 0, 4096, 0, clock%       'Make an x-axis (Time) calibration bar that
                                       starts 'at the left of screen and goes all the way
                                       to the 'right. Time 0 is at left. Time 50 (ms) is
                                       at 'right. (We are taking 1000 data points at 50
                                       µs 'per point, therefore we will have 50 ms of
                                       data.)
axisy 0, -2048, 4096, -10/scalefactor%, +10/scalefactor%
                                       'Make a y-axis calibration bar that shows the
                                       'data coming in in the range from -10 volts to
                                       '+10 volts.
WHILE (choices$ <? "S")                'Run all of the program until the wend statement
                                       'unless the variable choices$ is "S" (stop).
choices$ = inchar$(1)                  'Read the keyboard to see if a key has been
                                       'pressed. If yes return key to variable. Variable
                                       'choices$ is a string.
IF choices$ = "" THEN                  'No key was pressed by user, will repeat
                                       'previous command.
   choices$ = oldchoice$
END IF
```

```
IF choices$ <> "C") AND (choices$ <> "F") AND (choices$ <> "H") AND
(choices$ <> "N") AND (choices$ <> "S") AND (choices$ <> "T") THEN
    choices$ = oldchoice$               'Check whether user hit a wrong key
END IF

CASE SELECT choices$                    'We determine which key has been pressed.
CASE "C"                                '"C" was pressed, meaning new clock rate for
                                        'sampling data is required.
    INPUT "Enter new sampling clock rate (µs) - "; clock%
    axisx  -2048, 0, 4096, 0, clock%
    axisy  0, -2048, 4096, -10/scalefactor%,+10/scalefactor
    oldchoice$ = "F"                    'Return scope to free running condition
CASE "F"                                '"F" key pressed so user wants to scope to free
                                        'run.
    wipe                                'Clear display screen
    samplel  clock%,1000,1,0,0,1        'Digitize data into the computer.
    scmul  =  scalefactor%              'Factor to multiply data before display
    disl  0, 1000                       'Display the data just acquired
    oldchoice$ = "F"                    'Return scope to free running
CASE "G"                                '"G" key was hit so user wants to change voltage
                                        'gain.
    INPUT "Enter new multiplication factor (1-100) -"; scalefactor%
    axisx  -2048, 0, 4096, 0, clock%
    axisy  0, -2048, 4096, -10/scalefactor%, +10/scalefactor
CASE "H"                                '"H" key was pressed so user wishes to hold on
                                        'this sweep.
    hold                                'Hold here until any key is pressed.
    oldchoice$ = "F"                    'Return scope to free running
CASE "N"                                '"N" key means acquire next sweep and then
                                        'hold.
    wipe                                'Clear display screen
    samplel  clock%, 1000, 1, 0, 0, 1   'Take data into the computer.
    scmul  =  scalefactor%              'Factor to multiply data before display
    disl  0, 1000                       'Display the data just acquired
    hold                                'Hold here until any key is pressed.
    oldchoice$  =  "F"                  'Return scope to free running
CASE "T"                                '"T" key means trigger the next sweep externally
                                        after a delay set by the user.  Trigger will come
                                        in on A/D channel 15.
    ASK_AGAIN:
    INPUT  "Please enter delay after trigger - "; delay
    IF  (delay  <  0)  THEN  GOTO  ASK_AGAIN
    INPUT  "Please enter trigger potential (-10 to +10 volts) -"; trigpotential
    IF  (trigpotential  >  10)  OR  (trigpotential  <  -10)  THEN  GOTO  ASK_AGAIN
    TRIGPOTENTIAL  =  TRIGPOTENTIAL*204.8
                                        'Convert to computer units by multiplying by
                                        '204.8
```

```
        INPUT  "(1) Trigger greater or  (2) trigger less than  trigger potential - ";
        trigdirection
        IF (trigdirection < 1) OR (trigdirection ? 2) THEN GOTO ASK_AGAIN

        IF trigdirection = 1 THEN
            WHILE (ad(15) < trigpotential)
            WEND
        ELSE
            WHILE (ad(15) > trigpotential)
            WEND
        END IF

        waitclock delay                      'Wait number of ms specified by delay
        wipe                                 'Clear display screen
        samplel clock%, 1000, 1, 0, 0, 1     'Digitize data into the computer
        scmul = scalefactor%                 'Factor to multiply data before display
        disl 0, 1024                         'Display the data just acquired
        hold                                 'Hold here until any key is pressed
        oldchoice$ = "F"                     'Return scope to free running
    CASE ELSE
    END SELECT
    WEND
    STOP
```

That's it. About 70 lines of code make this program run quite well. This program illustrates the power of the AxoBASIC program. With a relatively small number of commands, a user can write some quite powerful programs. It would not take experienced users very long to write a program like the one above (perhaps an hour or two), and they would probably write a much improved version over the one above. This program also illustrates the structured constructs available like WHILE**...**WEND, CASE SELECT and IF**...**THEN**...**ELSE.

Comparison Between AxoBASIC and BASIC-23

BASIC-23 was originally written for the Digital Equipment Corporation line of PDP computers. It was a dialect of BASIC specifically designed for laboratory data acquisition and analysis. The first version was written by Dr. Dan Brown while he was in the laboratory of Dr. Charles Stevens. The language was updated later by Dr. Gary Yellen while he was in Dr. Stevens's laboratory. The concept of the language was simple — to have an interpreted, easy to program language that took care of all of the hardware support like A/D converters, data display, etc. To provide speed, all data manipulation was to be done using precompiled (in assembly language) subroutines that worked on blocks of data in a USER BUFFER. This concept of operating on arrays of data was first popularized in array processors. Drs. Brown and Yellen did a magnificent job. The language was small, fast, easy to program, and allowed for large amounts of data storage. Both integer and floating point arithmetic were supported. Unfortunately, the platform on which the system was based, the PDP-11, has become obsolete. Today's IBM PC-based microcomputers are much faster and much cheaper than the PDP line of machines. Unfortunately there was no BASIC-23 available for IBM PCs, so AxoBASIC was written to be a clone of BASIC-23. The use of the Microsoft QuickBASIC environment reduced the programming task enormously. The parser, the editor and some of the hardware support are performed by QuickBASIC. AxoBASIC adds all of the buffer manipulation routines and laboratory hardware support.

How similar is AxoBASIC to BASIC-23? It is very similar: 90% of the commands and the syntax are identical. For the remaining 10%, AxoBASIC has equivalent commands that in some cases are simpler to use and are otherwise required because of unavoidable constraints. For instance, RESET clears the display screen in BASIC-23. RESET is a QuickBASIC reserved word, so now CLRSCRN clears the display screen. With few exceptions (*e.g.*, the DISPLAY command), there are equivalent AxoBASIC and BASIC-23 commands. The BASIC-23 parser has a more flexible parameter stack than that used by AxoBASIC. For instance, to display 256 points in BASIC-23, you would use either:

```
    DISB 0, 1
```
or
```
    DISB 0
```

BASIC-23 was smart enough to supply elements on the parameter stack that were missing. In AxoBASIC you must specify the entire parameter list or the program will tell you that you are missing an element. In AxoBASIC the following is the only accepted syntax: DISB 0,1.

It usually does not take very long to convert BASIC-23 to AxoBASIC, since the languages are almost identical. The hard part is actually getting the text from a PDP to an IBM machine in ASCII (not compressed or tokenized) form. Once transferred, it takes about 2 -3 hours to get modestly long programs, of 1,000-1,500 lines of code, to work. Usually, all it takes is to find the BASIC-23 commands in which the entire parameter stack was not specified, *i.e.*, in which the programmed defaults were used. Then change the few commands that now have new names (*e.g.*, RESET to CLRSCRN). At this point the program should work as is.

Although AxoBASIC supports line numbers and lots of GOTO statements, this form of programming is not recommended. Programs should be re-written to employ the new more modern programming constructs available in AxoBASIC. In the long run, that will make the programs much easier to maintain. The nice part is that re-writing is optional; it's not required.

Future Directions

What is lacking in AxoBASIC? New features are being added all the time. Many constraints are imposed by DOS itself and the 640 KB (640 kilobyte) limit on IBM-compatible computers. There are many statistical features that would be beneficial to AxoBASIC users. For instance, wouldn't it be nice to have data sitting in the computer in an array and be able to do a linear regression with a command like LINEREG. Or if you need to fit a double or triple exponential, you could just pull up a SIMPLEX fitting routine built into AxoBASIC. While there are programs that do regressions or fit with SIMPLEX, that is not quite as convenient as having them built into the language itself. There are many statistical functions that could be in the language, but have not been incorporated due to space limitations. The more functions we put in, the fewer of the 640 kB remain available for programming by the user. This is the common trade-off in the MS-DOS environment. Fortunately, version 7.1 of the QuickBASIC extended environment (QBX) allows very large programs if the user has expanded memory available in the computer. In fact, there will be no practical limit on AxoBASIC's program size in the QBX environment. Nevertheless, the AxoBASIC library, QBX and the data buffers still have to fit into 640 K; therefore we cannot now add all of the above-mentioned statistical functions. Version 1.1 of AxoBASIC also increases the number of graphic objects that can be sent to hardcopy, since the display is quite difficult to overload. You can load in 100 sweeps of 1,024 vectors *and* 16,500 vectors *and* 16,500 plot points. Additionally, unlimited sized arrays can be set up in expanded memory. The USER BUFFER is still limited to 64 KB, and future plans will focus on moving the USER BUFFER to extended memory where it will be able to be as long as required.

What about hardware improvements? In fact, 80386 computers today make wonderful laboratory computers because they are fast and cheap. With memory prices coming down, computers with 4-8 MB (megabytes) of memory will soon be in everyone's laboratories. The only drawback at present for computers is in the display. AxoBASIC itself can be used in two configurations; one of which has a very high resolution graphics subsystem. This combination is monochrome and a little expensive, but it is superb for single-channel experiments because all channel transitions can be seen easily by the user. AxoBASIC is also available for VGA monitors where it works well, but the screen could have more resolution for single-channel experiments. Nevertheless, everyone has VGA and it is relatively inexpensive? Higher resolution modes like 1024 x 768 graphics are commonly (and cheaply) available for PCs but as yet there is no standard definition for this mode. At present there are several incompatible systems battling it out to see which one wins the 1024 x 768 standard. As soon as the dust settles, there will be a version of AxoBASIC available for the winning graphics standard.