sYbil

by Norman Weinberg

Once in a while a new product arrives on the scene that changes the face of the scene itself. *sYbil* is such a product. *sYbil* isn't a new type of drum, electronic controller, drum machine, or any other type of hardware; *sYbil* is software for a variety of personal computers.

Software is an amazingly versatile animal that can transform a personal computer into a new and powerful tool. A few of these tools have been around a while, and many have gone through several "generations." In the ever-changing world of electronic music, sequencers, notation programs, librarians, editors, and even logarithmic composition programs have dominated the scene to such an extent that recent progress has been measured in the area of user interface and "fine-tuning" the copious features. It's been quite a while since a program ventured into areas that were previously unexplored. *sYbil* was released at the Summer NAMM show of 1989, and has since caused quite a stir.

The program is billed as "Music Software for Real-Time Performance." In a nutshell, *sYbil* forces a computer to act as an interpreter between a MIDI controller (a device that will be sending MIDI messages) and a MIDI sound generator (a device that will be receiving the MIDI messages and producing the sound). You simply plug the MIDI-Out of the controller into the computer's interface, then connect another cable between the interface and the sound generator. All MIDI messages then have to pass through the computer before going to the sound generator—allowing *sYbil* to do its thing.

But this particular interpreter would find it difficult to land a gig at the United Nations. Instead of merely passing the MIDI information on down the cable, *sYbil* is capable of creating messages of its own—based on the desires of the "sYbilist" who programmed the "identity." (More on this in just a second.)

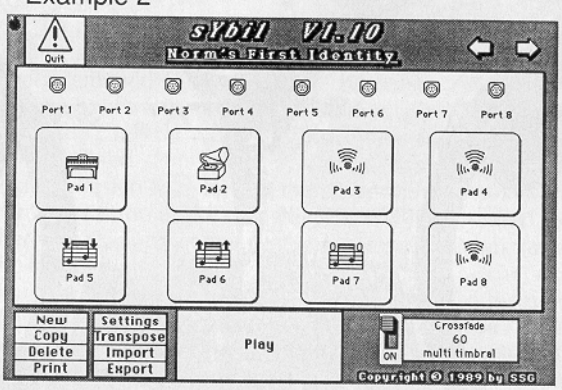
In order to run *sYbil*, you're going to need an appropriate computer, two floppy drives or a hard disk (recommended), a MIDI interface for the computer, a velocity-sensitive sound generator, and some type of controller. (I used a *Mac Plus*, which also requires a copy of *HyperCard* 1.2 [or higher].) The program's namesake was a woman who suffered from extreme multiple personalities. (Perhaps you saw the movie.) *sYbil* can give your MIDI percussion controller more personalities or "identities" than the real Sybil ever dreamed of.

Imagine a computer program that would perform actions based around the following rules: Whenever MIDI note number 55 comes into the computer, send note 60 out on MIDI channel one, note 63 out on MIDI channel two, note 67 on MIDI channel three, and note 70 out on MIDI channel four. In addition, give each of those MIDI notes its own relative volume and duration! All of a sudden, your old, tired, four-program *Octapad* becomes (picture a spandex-clad *Octapad* with legs, running out of a phone booth on a crowded street): ***sYbilpad***.

Let's take a look at what *sYbil* can do. After first launching the program from the desktop, you're presented with a screen that lets you further configure *sYbil* for your system (Example 1). If you have a MIDI interface that runs at an uncommon speed, or want to run *sYbil* from the printer port of your computer, now is the time to let *sYbil* know your intentions. Clicking on the little button labeled "settings" brings up the dialogue box for selecting files. These "settings" are actually *sYbil's* identities.

Example 1

Example 2



After loading an identity map into *sYbil*, you get a graphic similar to the one in Example 2. The major portion of this window is taken up by the white box showing eight squares and eight more MIDI jacks. The eight squares loosely resemble the eight pads of a standard *Octapad* or *Octapad II*. The icons of the MIDI jacks represent eight additional triggers. In other words, *sYbil* can respond to 16 different trigger surfaces.

In order to have *sYbil* change the identity of your controller, you'll have to program MIDI note numbers from your controller to run consecutively in any two eight-note groups. Any note that falls outside of these groups will be ignored by *sYbil*, but passed along by the computer to your sound generator (much like a MIDI-Thru function).

Before you can hear anything from your sound generator, you need to click on the large button labeled "play." This command actually loads the on-screen identity map into the computer, and after about 25 seconds, the computer will beep at you—indicating that your controller now has a new identity.

To program a new identity for *sYbil*, you must first exit from the play mode, and go back to the identity screen. Clicking on one of the pads calls up a note screen similar to the one in Example 3. While this screen lets you see what information the pad will be sending when it is struck, the only real editing available is activating the notes by clicking in the little check box under the note numbers. If there is an "x" in the box, the data for that pad will be sent; if the box is empty, then that particular event has been turned off. In order to actually change the data for a particular pad, you have to click somewhere in the data rectangles and move to a different screen.

Once in the new screen (Example 4), you can select the MIDI channel by clicking on the "push-buttons" at the top of the window, the MIDI note by clicking on the keyboard, the volume (as a percentage of full volume) by clicking on the arrows at

the bottom right corner, as well as the gate time or length of the event. In addition, the radio buttons at the bottom of the screen will let you determine whether these values are only for a particular note, all notes assigned to the pad, or all pads in the setting (identity).

In addition to programming a pad to send up to four different MIDI events, pads can also be programmed to perform something called "toggles." Toggles serve to change the entire controller's programming in some predetermined way. If you hold down the option key when you click on a pad, you're taken to the toggle selection window. As shown in Example 5, any pad can have one of nine different functions. Keep in mind that a pad with a toggle assigned to it will perform the toggle function and fire up to four different events. What follows is a brief explanation of the nine functions, moving down in columns:

The first selection—called "pad"—means that the pad will only send note information (no toggle). The second—called "sustain"—will force MIDI data on all pads to sustain until a variety of conditions are met. For example, all pads can sustain until the toggle pad is hit again, sustain until another pad is struck (sending a different set of note numbers), or sustain until some pad is struck twice in a row (sending the same set of note numbers). As you can see, the sustain options are exceptionally flexible. The last toggle in the first column lets you change some of the rules for the sustain parameters.

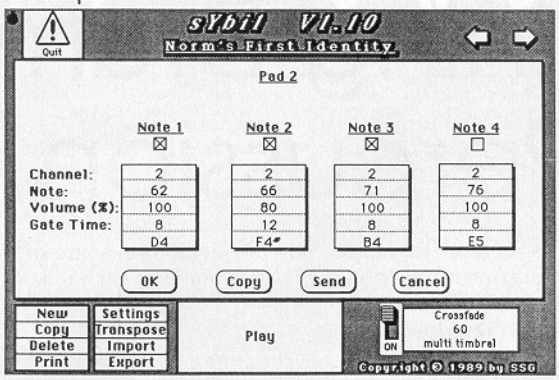
The first toggle in the second column forces all the pads in the setting to transpose up by some number of half-steps. For example, if you have the toggle set to seven half-steps (a perfect fifth), then if pad number one was programmed to trigger a C7 chord, it will now fire a G7. The next toggle will transpose all the pads in the setting down by an equal number of half-steps. The last toggle is used to reset any transposition toggle back to its original position.

The first toggle in the last column is used to return all pads back to "square one." In other words, any changes that were in force due to activating any of the toggles would be cancelled whenever a "square one" pad was struck. The next toggle allows you to link up to four other identities into a chain. That's correct: Hit a pad that has a chain assigned to it, and bingo—an entirely new identity is now active on the controller. The last toggle is used to move between four sets of program change messages. You can ask *sYbil* to send any program change message over any of the 16 MIDI channels. (And when you consider that there are four sets of these program changes, you can see a slew of possibilities for multi-timbral operation.)

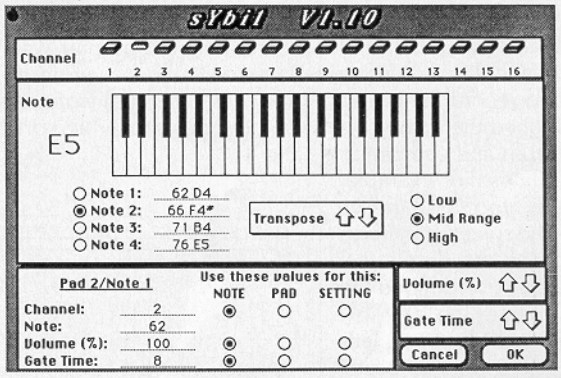
As if all these options weren't enough, you may have noticed a crossfade switch on the main identity screen (Example 2). If crossfade is activated, each pad can serve double duty by switching to a new identity whenever a certain volume threshold is passed.

If you're a "right-thinking" electronic drummer, your mind should be racing with possibilities. An electronic drumkit with minimal MIDI implementation has just been given a new lease on life. Any drum machine now has an "ultimate-stacked-note" feature. Using the program with an external sequencer can produce intricate structures with a minimum of programming.

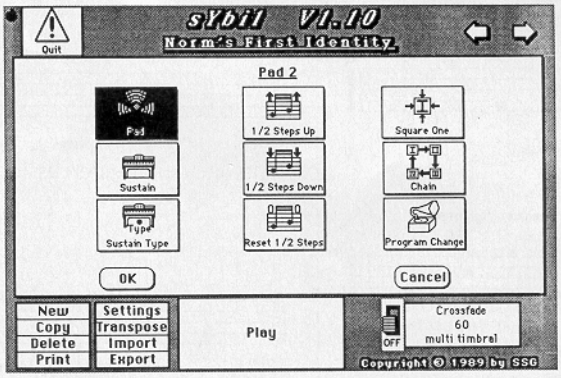
Example 3



Example 4



Example 5



How about using *sYbil* with a mallet controller like the *KAT* or *Silicon Mallet*? (Remember that any note number outside of the specified range will be passed through the program.) You can even use *sYbil* with another player.

Another player? Your keyboardist or guitar player can "direct" your tonality by playing certain "key" notes. Perhaps you're using an *Octapad* with only eight pads, and you program pad nine to call up another identity with the chain toggle. Then, whenever your musical partner plays that particular MIDI note, your pads will all send different data.

The program itself is copy-protected, but can be easily installed on a hard disk. The *HyperCard* stacks that represent identity maps can be freely shared between friends, downloaded from bulletin boards, or traded in *sYbil* user groups that are bound to spring up. There is even a feature for exporting a map (thus pulling the current map out of the main stack) and importing maps (adding an exported map back into the main stack). This feature makes sharing maps a simple proposition, and I wouldn't be surprised if hundreds of maps show up on bulletin boards fairly soon.

With all this going for it, *sYbil* has a serious flaw. Just as Sybil (the person) had a few problems when trying to function in the real world, *sYbil* (the program) has a few afflictions of its own. Let's not pull any punches: *sYbil* is slow. Very slow. Too slow. While there is no noticeable delay when using *sYbil* as a performing tool in real-time, moving from screen to screen and programming the values you desire while defining *sYbil's* persona can be exceedingly time consuming.

The program is written in computer languages called C, C ++, Pascal, and Assembly Language—which are very fast. But the human interface (how a person deals with the program) is written in *HyperCard*, which is very slow. *HyperCard* is a great tool for programmers, as it lets the developer design a prototype interface with a minimum amount of time and trouble. The developer can give the program to people, let them bang around on it for a while, and if something doesn't look right on the screen, it's a simple matter to draw a new picture and give that picture a particular function.

Using *HyperCard* in this way makes programming *sYbil* identities very easy. One doesn't need to be a computer whiz to get up and running. But *HyperCard* is currently too slow for a program like *sYbil*! It can take close to a minute to move from the identity screen to the programming screen to the note selection screen, select the note you want, and exit back to the identity screen. A minute may not seem like a long time, but when you're trying to select up to four notes for 16 pads, it adds up.

The biggest problem with the interface is

continued on page 104

ELECTRONIC REVIEW continued from page 45

that it isn't active in real-time. The only way you can play your controller is to leave the programming mode entirely, by clicking on the play button and waiting for the identity to load into the computer. Perhaps you selected a wrong note, chose the wrong gate time, or just want a particular voice to be stronger in the mix: Back you go to the identity screen, the programming screen, the note screen, and...well, you get the idea.

With today's technology, there is no reason to force the user through a bunch of cumbersome controls. Today's computer musicians are used to seeing a particular value, selecting it, typing in the new value, and hearing the fruits of their labor on the instrument. *sYbil* will let a single percussionist sound like a four-piece band. One note number can be a drum sound, another a bass note, another a melody note, and the fourth an inner part. But keeping all those rhythmic, melodic, and harmonic aspects in your head (or even written down, for that matter) is close to impossible without hearing how the pads react to each other in real-time.

The developers of *sYbil* are aware of the problems with the interface, and are doing their best to remedy the situation. For one thing, there has been some talk of a new version of *HyperCard*. Since each new *HyperCard* release has shown a vast improvement in speed, there is a strong reason to believe that the next version will speed *sYbil* up considerably. The developers are also working with the interface to make note and toggle selection faster. Instead of programming a single pad at a time, it will be possible to change the parameters of all pads from a single screen. This in itself will be a vast improvement. And all owners of sYbil 1.0 will be offered a free upgrade to the next version (which will be 2.0).

Warts and all, it's amazing that software like *sYbil* even exists. It's possible to draw an analogy between *sYbil* and one of the early Moog synthesizers. At first, musicians were forced to spend a lot of time plugging patch cords from one oscillator to another, adjusting scores of knobs and sliders, and carting around hundreds of pounds of gear. Creating sounds took a very long time, but if you wanted to hear those sounds at all, you had to put up with the "price of technology" headaches. At times, *sYbil* 1.0 is frustrating, but one thing is for sure: Using *sYbil* is a hell of a lot more creative than not using it!

sYbil carries a retail price of $299—a small price to pay for adding a tremendous amount of performance power to your ex-

isting electronic system. I believe that *sYbil* and the next generation of "Real-Time Performance" programs could well define the future of music software. After all, once you're happy with a sequencer or notation package, how many more do you need? My recommendation? If you own a controller, a compatible computer, and a sound generator, consider *sYbil*. You'll be getting in on the ground floor of a new potential in your performance abilities.

Editor's note: As we went to press, we were informed by Scorpion Systems—manufacturers of sYbil*—that the improvements recommended by Norm had been implemented. Version 2.0 of the* sYbil *program is now being offered. Version 1.0 had been designed for use only with the Apple Macintosh, and Norm's comments reflect its performance with that system—including the use of* HyperCard *for its editing and the delays that Norm mentions. New 2.0 versions may also be used with Atari* ST*, IBM and compatibles, and Yamaha* C1 *computers. These versions don't rely on* HyperCard*, and all offer instantaneous editing. The Macintosh version of 2.0 still uses* HyperCard*—which Scorpion Systems acknowledges is still slow—but the interface has been completely reworked and the delays described by Norm have been reduced tremendously.*