18 Advanced CAL Techniques

In Chapter 17, you learned about the Cakewalk Application Language—what it is, what it does, and how you can run prewritten CAL programs to tackle some of the editing tasks that the built-in SONAR functions can't handle. I briefly touched on the topic of creating your own CAL programs. This chapter continues the CAL discussion and will do the following:

- Teach you the Cakewalk Application Language programming basics.
- Explain the anatomy of a CAL program.

Introduction to CAL Programming

Unfortunately, there is no easy way to create your own CAL programs. To tap the full power of its functionality, you need to learn how to create programs from scratch using the Cakewalk Application Language. The problem is that teaching a course in CAL programming would take up an entire book. So instead, I'll just get you started by providing a brief introduction to the language. The best way to do that is to walk you through the code of one of the CAL programs that comes included with SONAR.

To get started, open the Scale Velocity.CAL program (see Figure 18.1). The first thing you'll see is a bunch of lines that start with semicolons and contain some text describing the CAL program. These lines are called *comments*. Whenever you insert a semicolon into the code of a CAL program, SONAR ignores that part of the code when you run the program. This way, you can mark the code with descriptive notes. When you come back to the program at a later date, you will understand what certain parts of the program are supposed to accomplish.

A little farther down, you'll notice the first line of the actual code used when the program is run. The line reads `(do`. All CAL programs start with this code. The parenthesis designates the start of a function, and the `do` code designates the start of a group of code. As a matter of fact, a CAL program is just one big function with a number of other functions in it. You'll notice that for every left parenthesis, you'll have a corresponding right parenthesis. CAL programs use parentheses to show where a function begins and ends.

gajendra D:/Thomson_Learning_Projects/Garrigus_163132/z_production/z_3B2_3D_files/Garrigus_163132_ch18.3d, 14/11/08/16:27:16, 16:27, page: 658

Scale Velocity.cal - Notepad

File Edit Format View Help

```
;; Scale Velocity.cal
;;
;; This is a sample CAL program that implements an editing command to
;; scale note velocities by a certain percentage.
;;
;; Demonstrates:
;;
;;   (forEachEvent)
;;   Getting input from the user
;;   Arithmetic operators
;;   Event kind and parameter variables

(do
        (include "need20.cal")  ; Require version 2.0 or higher of CAL

        (int percent 100)
        (getInt percent "Percentage?" 1 1000)

        (forEachEvent
                (if (== Event.Kind NOTE)
                        (do
                                (*= Note.Vel percent)
                                (/= Note.Vel 100)
                        )
                )
        )
)
```

Figure 18.1 The code for the Scale Velocity.CAL program provides a nice example for an explanation of the Cakewalk Application Language.

The `include` Function

The next line in Scale Velocity.CAL reads `(include "need20.cal")`. This is the `include` function, and it allows you to run a CAL program within a CAL program. You might want to do this for a number of reasons. For instance, if you're creating a very large program, you might want to break it down into different parts to make it easier to use. Then you could have one master program that runs all the different parts. You can also combine CAL programs. For example, you could combine the Thin Channel Aftertouch.CAL, Thin Controller Data.CAL, and Thin Pitch Wheel.CAL programs that come with SONAR by using the `include` function in a new CAL program. Then when you run the new program, it will run each of the included programs, one right after another, so you can thin all the types of MIDI controller data from your project in one fell swoop.

In Scale Velocity.CAL, the `include` function is used to run the need20.CAL program. This program simply checks the version of CAL and makes sure it is version 2.0 or higher. Some CAL programs check the version to avoid an error, in case a very old version of CAL is being used.

gajendra D:/Thomson_Learning_Projects/Garrigus_163132/z_production/z_3B2_3D_files/Garrigus_163132_ch18.3d, 14/11/08/16:27:16, 16:27, page: 659

Variables

After the `include` function, the code for Scale Velocity.CAL shows `(int percent 100)`. This is a *variable* function. In CAL programs, you can define variables to hold any values you might need while the program is running. In this instance, the variable `percent` is defined as an integer and given a value of `100`. You can use variables to store both number and text information. After you define a variable, you can refer to its value later in your code by simply using the variable name. That's what you see in the next line of code in Scale Velocity.CAL.

User Input

This next line reads `(getInt percent "Percentage?" 1 1000)`. Here, the program asks for input from the user. In plain English, this line of code translates to, "Get an integer between 1 and 1,000 from the user by having the user type a value into the displayed dialog box. Then store the value in the variable named `percent`." Basically, when SONAR reaches this line of code in the program, it pauses and displays a dialog box (see Figure 18.2) and waits for the user to input a value and click the OK button. It then assigns the input value to the variable `percent` and continues running the rest of the program.

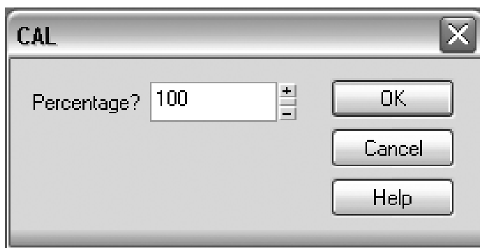

Figure 18.2 A CAL program gets input from the user by displaying dialog boxes.

The `forEachEvent` Function

The main part of the Scale Velocity.CAL program begins with the line of code that reads `(forEachEvent`. This is known as an *iterating* function. In this type of function, a certain portion of code is run (or cycled through) a specific number of times. In this case, for every event in the selected track(s), the code enclosed within the `forEachEvent` function is cycled through one time. So in Scale Velocity.CAL, the following block of code is run through once for every event in the selected track(s):

```
   (if (== Event.Kind NOTE)
  (do
   (*= Note.Vel percent)
   (/= Note.Vel 100)
  )
)
```

What does this code do? I'll talk about it in the following sections.

gajendra D:/Thomson_Learning_Projects/Garrigus_163132/z_production/z_3B2_3D_files/Garrigus_163132_ch18.3d, 14/11/08/16:27:16, 16:27, page: 660

Conditions

Within the `forEachEvent` function in Scale Velocity.CAL, every event in the selected track is tested using the `if` function. This function is known as a *conditional* function. Depending on whether or not certain conditions are met, the code enclosed within the `if` function may or may not run. In Scale Velocity.CAL, every event is tested to see whether it is a MIDI note event. This test is performed with the line of code that reads `(== Event.Kind NOTE)`. In plain English, this line translates to "Check to see whether the current event being tested is a MIDI note event." If the current event is a MIDI note event, then the next block of code is run. If the current event is not a MIDI note event, then the next block of code is skipped and the `forEachEvent` function moves on to the next event in the selected track until it reaches the last selected event; then the CAL program stops running.

Arithmetic

The final part of Scale Velocity.CAL is just some simple arithmetic code. If the current event is a MIDI note event, the velocity value of the note is multiplied by the value of the `percent` variable, and the resulting value is assigned as the note velocity. Then the new velocity value of the note is divided by 100, and the resulting value is assigned to be the final value of the note velocity. This way, the program scales the velocities of the notes in the selected track(s).

Master Presets

One of the most effective uses I've found for CAL is in creating what I like to call *Master Presets*. SONAR lets you save the settings for some of its editing functions as presets. This way, you can use the same editing parameters you created simply by calling them up by name, instead of having to figure out the settings every time you use a function.

Presets are a real timesaver, but unfortunately, you can save presets only for each of the individual functions. What if you want to combine a few of the functions to create a certain editing process? For example, suppose that you like to shorten your MIDI tracks before you quantize them. To do so, you need to select the tracks, use the Length function, and then use the Quantize function to process your tracks. For each of the editing functions, you have to make the appropriate setting adjustments. If you create a CAL program to automatically run through the process for you, though, all you need to do is select your tracks and run the CAL program.

To show you what I mean, I've cooked up a sample Master Preset you can run as a CAL program and use in your projects. You need to complete the following steps:

1. Open Windows Notepad.
2. Type in the first few lines of code, as shown in Figure 18.3.
3. Examine the code. The first line is just a blank comment. The second line designates the beginning of the program. The third line tells SONAR to activate the Length function using the parameters shown in Figure 18.4.

gajendra D:/Thomson_Learning_Projects/Garrigus_163132/z_production/z_3B2_3D_files/Garrigus_163132_ch18.3d, 14/11/08/16:27:16, 16:27, page: 661

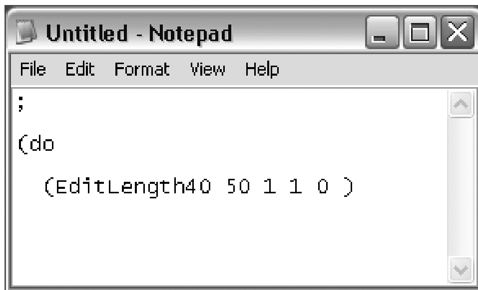

Figure 18.3 These are the first few lines of code in your new Master Preset.

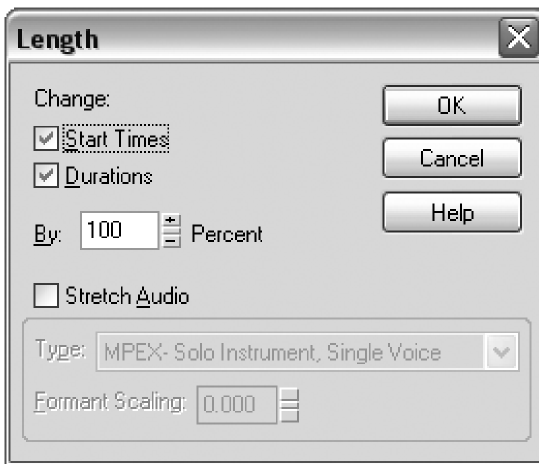

Figure 18.4 The first part of the CAL program shortens the selected MIDI tracks by 50 percent with the Length editing function.

In the source code, the command `EditLength40` tells SONAR to activate the Length function. The number 50 corresponds to the Percent parameter in the Length dialog box. The numbers 1, 1, and 0 correspond to the Start Times, Durations, and Stretch Audio options, respectively. A 1 indicates that the option is activated; a 0 indicates that it is not.

4. Now type in the last two lines of code, as shown in Figure 18.5.
5. Examine the code. The command `EditQuantize40` tells SONAR to activate the Quantize function using the parameters shown in Figure 18.6. The numbers following that command designate the following parameter settings: Resolution, Strength Percent, Start Times (on/off), Note Durations (on/off), Swing Percent, Window Percent, Offset, and Notes/Lyrics/Audio (on/off).
6. Save the new program with a file extension of .CAL.

Now when you run this CAL program, it performs all the editing functions for you automatically, with the same settings you used. It's too bad that CAL doesn't support SONAR's MIDI or

gajendra D:/Thomson_Learning_Projects/Garrigus_163132/z_production/z_3B2_3D_files/Garrigus_163132_ch18.3d, 14/11/08/16:27:16, 16:27, page: 662

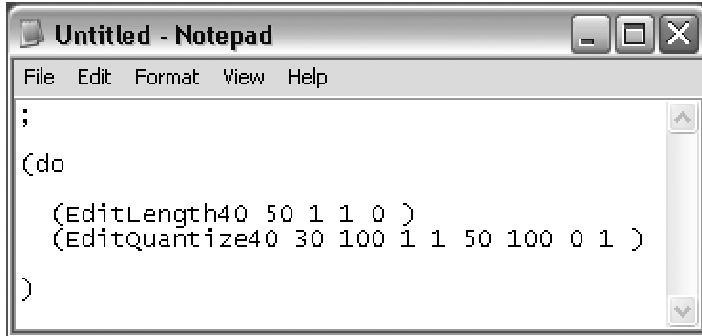

Figure 18.5 The final source code should look like this after you edit it.

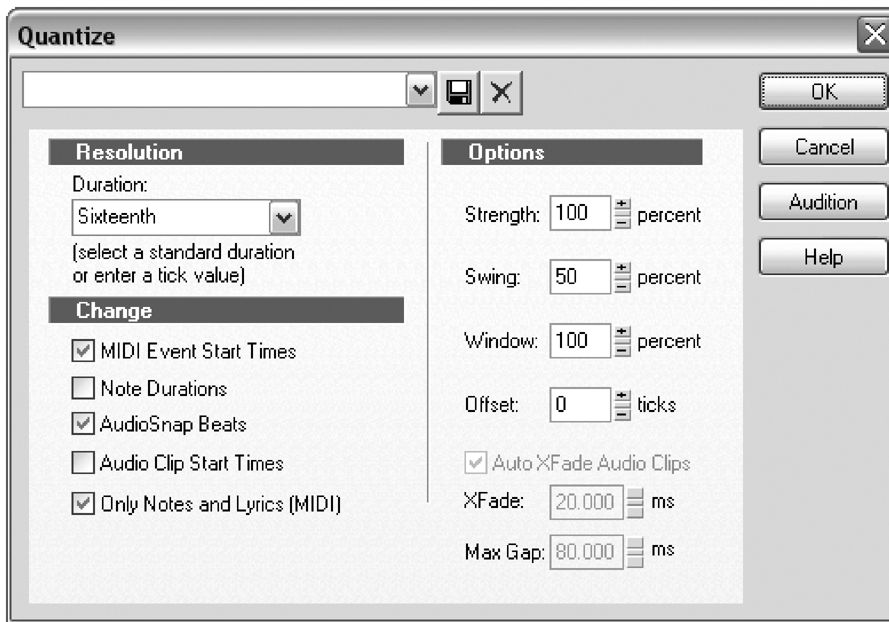

Figure 18.6 The second part of the CAL program quantizes the notes in the selected MIDI tracks with the Quantize editing function.

audio effects functions. I really wish it did, because then you could create Master Presets to process your audio tracks, too. That capability would make CAL a hundred times more powerful than it already is. I hope Cakewalk will add this functionality in a future version. In the meantime, you can still find plenty of uses for CAL.

CAL Conclusion

So are you totally confused yet? If you've had some previous programming experience, you should have no trouble picking up the Cakewalk Application Language. If you're familiar

gajendra D:/Thomson_Learning_Projects/Garrigus_163132/z_production/z_3B2_3D_files/Garrigus_163132_ch18.3d, 14/11/08/16:27:16, 16:27, page: 663

with the C or LISP computer programming languages, CAL is just a stone's throw away in terms of functionality.

Really, the best way to learn about CAL is to study the code of existing CAL programs. If you still find yourself lost in all this technical jargon even after this discussion, you can utilize CAL by using prewritten programs. As I mentioned before, this part of SONAR has a lot of power, and it would be a shame if you let it go to waste. CAL can save you time and even let you manipulate your music data in ways you might never have considered. Don't be afraid to experiment. Just be sure to back up your data in case things get a bit messed up in the process.