Letters

Dear Editor:

I would like to compliment John Barilla on his excellent article "Understanding Timecode Synchronization." However, it seems that I have become self-appointed Guardian of Timecode Facts, and as such, I feel a couple of problems should be pointed out for the benefit of readers new to this arena.

First, to clarify the most misunderstood aspect of timecode usage: drop-frame. SMPTE timecode comes in rates of 24,29.97 and 30 frames per second: 29.97 fps code is always used in video production today, and can be either drop or non-drop frame. However, because the timecode runs at 29.97 fps but counts to an even "30", at the end of one real-time hour the code has not yet reached the one-hour count. When the timecode finally reaches 1:00:00:00, the program is actually 108 frames (0.1%) too long. The drop-frame system was developed to correct this inequity between the timecode count and the true elapsed time when using 29.97 fps code. This is done by eliminating the first two frames in every minute, except the tens minutes (0, 10, 20, 30, 40, 50), thus reducing the final count by the necessary 108 frames. Contrary to the implication of the article, however, 29.97 fps non-drop frame code is commonly used by video houses, because it is simpler to deal with, except when exact timing is required (such as in broadcast).

Secondly, I would be less casual about the importance of frame rates. While it's true that things will generally work themselves out if all dubs are done properly, synchronizers do not simply "act a little strange" when dealing with mixed frame rates. Serious out-of-sync nastiness can result if attention is not given to this matter, particularly when interchanging with a video house. Remember, in a simple synchronizing system, all transports will be resolved to the rate of the master code. Even if they have different rate code, this can effectively change the actual "speed" of the audio. I recommend using 29.97 fps code if dealing with video.

Lastly, the terminology used by a given timecode reader/generator should be examined carefully when jam-syncing code. I interpret continuous jam to mean "follow the incoming code exactly at all times," which would not do for restriping a missing segment of code. In this case I would use momentary jam, which says "look at the incoming code initially, but then keep counting on your own" (locked to an internal crystal or external video sync, not tach pulses!).

Eric Wenocur
KLM Video, Inc.
Bethesda, Md.

John Barilla responds:

I am grateful to Eric Wenocur for both his kudos on my article and his criticism. There is no doubt that Mr. Wenocur is, in fact, a bona fide "Guardian of Time-Code Facts"—something that I do not claim to be. The purpose of my article was simply to point out the everyday practice of time-code usage on the level of *The Electronic Cottage* (the smaller, personal production studio).

On this level, commercials and industrial presentations are often done on a "quick and dirty" level; fact is, for short programs, this works sufficiently well. I didn't feel my article required going into the (admittedly important) distinction between the various time-code formats, however, Mr. Wenocur did an excellent job of doing so, and his letter serves as an informative postscript to my article.

Finally, I thank Mr. Wenocur for pointing out my inadvertent use of the term "continuous jam sync"—for the process known as "one-time" or "momentary" jam-sync. I am always happy to receive input from db's readership. db