Q: As I understand it, the MIDI timer and I/O is handled with the **msg_receive** call, which is a kind of polling that happens whenever the application has time to poll. But if the application is busy with other things, might not the polling interval become too great or too variable, which could have a musically unacceptable effect?

A: It's true that your application must go into a **msg_receive** to receive MIDI. However, it's possible to structure your application so that there is a separate Mach thread that does nothing or little else than service this **msg_receive**. This should provide the response time you need. For example, if you use the MusicKit (see ../NEXTSTEP_Developer/Other_Kits/music_software), there is a feature that allows the MusicKit performance (and, hence, the MIDI **msg_receive**) to run in a separate thread from the graphics event loop. (This feature is also useful for non-MIDI performances, such as those using DSP synthesis.) Alternatively, if you don't use the MusicKit, you can fork your own thread to handle the MIDI asynchronously to the event loop.

Running the Music Kit in a separate thread makes it unlikely that graphics will prevent you from receiving your MIDI in a timely fashion. However, the MusicKit scheduler (just like the AppKit event loop) is non-preemptiveÐthere's no absolute limit to how much can be done before going back to the **msg_receive**. Therefore, if you write your own Performer, Instrument, or NoteFilter subclass, make it as efficient as possible, because you won't receive MIDI until you exit your code. In principle, the way to get the fastest MIDI response is to have your own separate thread loop processing the MIDI, without using the MusicKit. However, if that thread does much processing other than receiving MIDI, using the MusicKit would probably be just as efficient (and easier to program). The important thing is to get back to the **msg_receive** as quickly as possible, whether in the MusicKit, the AppKit, or your own thread.

QA621

Valid for 1.0, 2.0, 3.0