NEWTON Q&A: ASK THE LLAMA

This column first appeared in volume 28 of DEVELOP (the Apple technical journal for developers).

Q *I recently saw the announcement of two new Newton devices — the MessagePad 2000 and the eMate 300 — and I have a few questions. First, will my application run fine?*

A If your application was written correctly for Newton OS 2.0, for the most part it will work fine on Newton OS 2.1, which is what the new devices use. “Correctly” here means that you call only documented functions, including platform file functions where appropriate. It also means that your application works with different screen sizes by using GetAppParams in combination with minimum and maximum sizes.

Q *What are the most important things to check in my application to ensure that it’s completely compatible with the new devices?*

A There are four broad areas to check: speed, screen size, views, and PC cards.

Speed. The new units are faster than the current ones. In the case of the MessagePad 2000, the difference is quite large. Unfortunately, it’s possible for some things to be too fast. The new OS takes care of several speed issues for you — scrolling, for example — but there are still some areas you should check.

Check those places where you’re doing repeated actions from a viewClickScript. A typical such usage would be a button that will continually perform an action as long as the user presses it. If you use a loop in a viewClickScript to do the repetition, you may find that there are too many repetitions or that the repetitive action occurs too quickly. The same problem can occur if you don’t use the scrolling API provided by the system, as scrolling is one area that has deliberately been slowed down on the MessagePad 2000.

Loops can also cause problems when they’re used for timing. In general, you shouldn’t use loops for this purpose; use Ticks instead. If you must use a loop, set the counter for that loop based on a known reference like Ticks or TimeInSeconds.

Operations that used to be long enough to require user feedback may now happen fast enough that no feedback is required. This can happen in two places:

- You may have been deliberately turning on the busy box. The result can be a busy box that flashes very briefly, which can be distracting.

- You may have implemented progress bars using DoProgress, protoStatus, or even a protoGauge. Try removing the progress indicator and checking whether the operation is now fast enough. Note that most of the progress indicators take time to draw and update — in some cases significantly longer than the time to do the operation for which the progress is being displayed.

Screen size. Be sure your application works properly in both portrait and landscape orientation, with the button bar on both the left and the right. In addition to the size of your overall application on the screen, check areas where you use complex justification or dynamic allocation of view children. Check that the children are correctly aligned—and that there are the correct number of children. Also check that your borders do not go outside the application area. Note that borders are drawn outside the view. You can find your global bounds with borders by calling GlobalOuterBox.

Views. The most likely problem related to views is assuming that the top left of the application area is always at (0, 0) in global coordinates. This is no longer the case since the button bar, which is 46 pixels wide, can be on the left side of the screen. A typical place for this problem to occur is in a viewClickScript where you do something with a point that's actually 46 pixels out from where you expect it to be. Rotate to landscape orientation and put the button bar on the left; then try tapping on the right side of your application. One sure way to cause problems is to forget to send SyncView to your base application view after a ReorientToScreen message has been sent.

Another possible view problem is that any view can be the keyView. Don't assume that the keyView can accept text input; in particular, don't use calls like SetValue to jam the text slot (which may not be there). You may want to check the class of the keyView.

PC cards. You should check that your application works when two PC card slots are being used for storage. Search for the following pieces of bad code:

```
GetStores()[1]; // BAD -- 0 is the only number documented to work.

Length(GetStores()); // does not tell you the number of PC Cards.
```

If you have code like this, you'll have to change it. The first case — assuming there's a store at the second position in the array — is not a good idea. Even if there's a store at that position, it may not be the same store that was there the last time you checked. Also note that the positions in the array do not correspond to physical PC card slot

positions. The second case can fail for similar reasons. That is, checking the length of the array returned by GetStores doesn't tell you how many PC cards are currently installed.

Along the same lines, if you're still using the action (routing) picker to move items to the card, you should change to using the filing interface. Also, make sure that you use the FileThis method to move items to different stores, and that you look at the arguments provided by FileThis; some application code seems to assume that there are only two stores. For adding soup entries, remember to call AddToDefaulStoreXmit instead of using the store directly.

You might also encounter a problem with endpoints that could use PC card modems. To set up your modem endpoint call MakeModemOption, which will construct the correct options based on the user modem preferences and available PC card modems.

Q *Are there any features that are important to support in the new devices?*

A The most important thing is to make sure your application works. After that, there are some important updates you can make to support features in the new devices.

First, make sure your application can be dragged, assuming of course that it's not full-screen in all orientations. You can do this by changing your base view to a protoDragger with either some sort of status bar (preferably newtStatusBar) or a protoLargeCloseBox. Remember to make sure the borders of the protoDragger will be contained in the application area. You can use GlobalOuterBox to check this. And while you're changing that, you can check that your application will handle any screen size.

You may also want to see whether you can improve your use of screen space. Note that your major layouts (for example, detail and overview) don't have to be the same size. The Names application is a good example of this.

Another important feature to support is the use of the keyboard. Add the required keyboard commands to your application. As of this writing you can find this information in the "User Interface Guidelines for Newton OS 2.1 Keyboard Enhancements" document. If your application is based on NewtApp, most of this work is done for you, otherwise you'll have to add almost all the keyboard commands yourself. Once you've supported the required set, add other commands that make sense for your application. Don't forget keyboard navigation in your overview.

A related feature that's good to support on both Newton OS 2.0 and 2.1 devices is conditional display of embedded keyboards. You can use the KeyboardConnected global function to check whether a keyboard is connected; if it is, don't display embedded keyboards unless they're highly specialized.

If you're using the infrared (IR) communications tool, you should use IrDA if possible. This will give you a faster transfer rate and a much more robust protocol. If your application may be using IR to communicate with older units, be sure to give your users a choice of which type of IR connection to use since older units can only use the ASK protocol. Newton OS 2.1 still supports all the IR options from 2.0. Note that using the action menu to beam information will do the right thing.

You should also take advantage of the grayscale feature, by using the new RGB-based gray shades (that is, kRGB_Gray1 through 15) instead of the dithered gray patterns. Dithered patterns are usually specified as vfGray, vfLightGray, and so on. You can also change your own patterns to use grayscale. Although the dithered patterns still work, the true gray RGB shades look a lot better. You'll want to wrap the specification in a check to make sure that grayscale is available. Naturally you'll want to update important parts of your application to use grayscale — for example, your splash screen and Extras icon.

Finally, if you're targeting the eMate and the education market, you should update your application for multiuser mode. This could be an extensive change, since you'll have to modify your interface and the names of all soups that you save.

The llama is
the unofficial mascot of the Developer Technical Support group in Apple's Newton Systems Group. Send your Newton-related questions to dr.llama@newton.apple.com. The first time we use a question from you, we'll send you a T-shirt.

Thanks
to jXopher Bell, Bob Ebert, David Fedor, Ryan Robertson, Jim Schram, Maurice Sharp, and Bruce Thompson for these answers.

If you need more answers,
take a look at the Newton developer Web page, at http://www.devworld.apple.com/dev/newtondev.shtml.