NEWTON Q&A: ASK THE LLAMA

Q *Now that Newton 2.0 is shipping, what has changed?*

A A fair question, and one that's been much on my mind. Newton 2.0 solves some of the problems previously presented in this column in much better ways. So I've gone back over old questions to see what has changed. I'll start out this time by revisiting those questions that have new answers.

Questions that dealt with subsystems whose APIs on the Newton 2.0 OS are drastically different will not be covered. Most of these have to do with routing, which has undergone a significant change for the better; some have to do with communications.

Q *How do I create my own class of binary object? (Issue 18)*

A In the Newton 1.x OS you had to use SetClass on a string object to make some other binary object. In 2.0 you can just call the new function MakeBinary. So the line of code to define the canonical CharID (see the original answer) object changes to

```
DefConst('kDefaultCharIDOBj, MakeBinary(4, 'CharID));
```

Q *I would like to add a [button|view|Llama] to [Notes|Dates|Names|etc.]. How can I do that safely? (Issue 19)*

A In the Newton 1.x OS there was no supported way to add items to the built-in applications. In 2.0 there are a few ways you can do this.

For general changes, you can add new stationery to Notes, Dates and Names. For example, you could add graph paper to Notes. You could also define new card styles or views of a person in Names.

Names and Dates also let you add whole new classes of things. For instance, you could add a Pet type of names entry that would appear in the New pop-up menu along with Person, Company, and Group.

The Dates application has an API to add new types of meetings. It also lets you add items to its Info button.

In addition, there's a general API to register buttons that can show up in the blessed application's status bar. It's up to each application to decide whether and how it will display registered buttons. You should no longer use the unsupported keyboardChicken hack.

Note that the Newton 2.0 platform still does not support adding buttons to built-in slips. For example, if you wanted to add something to the alarm picker for a meeting, you would need to add a new type of stationery that's a superset of the alarm picker.

Q *I've written my own IsASCIIAlpha, IsASCIINumeric, etc. functions. They seem to be really slow. Why is that? Here's my IsASCIIAlpha: [code not repeated here; all the functions work on strings] (Issue 20)*

A Most of the comments from the original answer still hold. However, in the Newton 2.0 OS the string could be a Rich String; that is, there could be an ink character inside the string. That means the compare functions have to check whether a particular character was kInkChar.

Q *When I try to add an index to my soup I sometimes get an exception -48019, but not always. What's going on? (Issue 22)*

A In early versions of Newton, if you added an index on a slot and an entry in that soup had the value NIL for that slot, you would get an error. As of the Newton 2.0 OS this is no longer a problem. You can add an index even if there are entries with NIL values for the slot in the soup.

Q *I have an application that uses ADSP to connect to a server on the desktop. I want the server to handle multiple Newton devices connected simultaneously. Unfortunately, if a connection fails after it's opened, the server doesn't seem to be able to identify it as a new connection when the Newton device reconnects. This causes problems in the server's ability to handle multiple connections. Can you help? (Issue 23)*

A In the Newton 2.0 OS this no longer occurs. The Newton device will generate a new ID for the connection.

Q *Since there are changes between Newton 1.x and 2.0, what features in 2.0 can I rely on? What is the core set that defines Newton 2.0?*

A At this time there is no published core set of NewtonScript-level features that you can rely on. We are confident you can rely on the features of the NewtonScript language and major components like the view system and communication endpoint interface. However, you can't count on individual protos or even the internal applications being there. Since we license Newton technology to other companies, they could produce a Newton device that does not include Names, Dates or other built-in features. They may also produce Newton devices that have features not present in Apple products.

The key is to test the features you rely on. If you find that some of the features you need are missing, you can either run in a less featured mode or just not open your application. As a simple example, suppose your application runs only in a limited set of screen sizes and aspect ratios. You can give your base application view a viewSetupFormScript that looks something like this:

```
myBaseView.viewSetupFormScript := func()
begin
   local screenSize := GetAppParams();
   local aspectRatio := screenSize.appAreaWidth /
         screenSize.appAreaHeight;

   // very simplistic test, no MINIMUM even!
   if aspectRatio > 1.0 then  // landscape
   begin
      local maxHeight := kMaxAppWidth;
      local maxWidth := kMaxAppHeight;
   end;
   else begin                 // portrait or square
      local maxHeight := kMaxAppHeight;
      local maxWidth := kMaxAppWidth;
   end;

   if screenSize.appAreaWidth <= maxWidth AND
      screenSize.appAreaHeight <= maxHeight then
   begin
      self.viewBounds := RelBounds(....
      // other setup stuff
   end
   else begin
      // cannot operate at screen size
      :Notify(kNotifyAlert, EnsureInternal(kAppName),
         EnsureInternal(kErrorWrongScreenSize));
      AddDeferredSend(self, 'Close, nil);
   end;
end;
```

For global functions and variables, you can use the GlobalFnExists and GlobalVarExists utility functions. To find out whether a built-in application exists, you can check the root view with the appropriate symbol:

```
// check for Dates
if GetRoot().calendar then

// check for Names
if GetRoot().cardfile then

// check for Extras
if GetRoot().extrasDrawer then
```

For protos, you can try to access the proto and catch a frame reference exception. If the exception occurs, the proto is not present.

In general, it's a wise idea to do all your existence testing as your application is launching. Set flags in your base application so that you test for existing features only once.

Q *Is there a hardware-unique ID that I can access on a Newton device?*

A At this time there's no built-in hardware-unique ID, nor is there an API for accessing one if it existed. However, this doesn't rule out having such an API in future Newton devices.

Q *I'm using a Newton 2.0 protoSoupOverview and I want to change the font style. How do I do that?*

A This is one of those things that are obvious once you make the connection. You use the Abstract method of protoSoupOverview (and protoOverview, for that matter) to build the shape that's displayed for a particular soup entry. Notice that you're returning a shape, with all that entails. The chapter on drawing in the *Newton Programmer's Guide* says you can include a styles entry in a shape array, allowing you to specify things like font style. See the DTS Sample Code Checkbook for an example.

Q *I noticed that some of the built-in applications have keyboards in their slips — for example, the new name editor in the Names file. Is this stationery based? Is there a magic slot I can set? Is there a proto?*

A Those keyboards are just views based on protoKeypad that are laid out as a child view of the slip. All you need to do is lay out your own protoKeypad and set up the definitions appropriately. There is no supported magic slot.

Q *I'm trying to use a protoListPicker to display a soup structure that has nested frame entries. I can't get the listPicker to work. Am I doing something wrong?*

A No. The default listPicker proto doesn't work with items that are accessed via path expressions. However, if you make the following three changes, your listPicker should work fine.

First, you have to specialize the GetObjSlot method of your pickerDef:

```
GetObjSlot: func(item, fieldPath)
begin
   if ClassOf(fieldPath) <> 'pathExpr then
      // if not a path expression, return the inherited value
      return inherited:GetObjSlot(item, fieldPath);

   // otherwise, if there is no item, return NIL.
   if not item then
      return nil;

   // there is an item, so get the real value, since the item
   // could be a NameRef or an Entry
   if IsNameRef(item) then
      local val := EntryFromObj(item);
   else
      val := item;

   // assuming we have a real thing, access the real data via the
   // path expression in fieldPath
   if val then
```

```
    val.(fieldPath);
end
```

Second, if you specify a validation frame in for your listPicker, the nesting of that frame must match the nesting of your soup entry.

Finally, modify your pickerDef so that the column that displays the data based on the index path uses the appropriate index path.

Q *Do you do contract programming?*

A I tried, but I couldn't find an insurance company that would sell me liability insurance.

The llama is
the unofficial mascot of the Developer Technical Support group in Apple's Newton Systems Group. Send your Newton-related questions to NewtonMail DRLLAMA or AppleLink DR.LLAMA. The first time we use a question from you, we'll send you a T-shirt.

Thanks
to our Newton Partners for the questions used in this column, and to jXopher Bell, Henry Cate, Bob Ebert, David Fedor, Jim Schram, Maurice Sharp and Bruce Thompson for the answers.

Have more questions?
Need more answers? Take a look at Newton Developer Info on AppleLink.