NEWTON Q&A: ASK THE LLAMA

Q *When I try to print or fax something, the Newton usually runs out of memory. I'm sending a lot of data to my print format, but it seems that the **fields** frame passed to SetupRoutingSlip should have only a reference to the data. What am I doing wrong?*

A Unfortunately, you're missing one important step in the process of printing. The **fields** frame is eventually placed in the outbox soup. Thus all references are followed, which means that all the data you placed in the **fields** frame is duplicated. The duplication occurs in the soup:Add call.

In other words, you probably have a large data structure (or view, or proto) in your application. You put a reference to this structure in the **fields** frame in SetupRoutingSlip. When the user accepts the item to be printed (or faxed, beamed, or whatever), the **fields** frame is placed in the outbox soup, causing your structure to be duplicated.

Ideally, you should pass as little data as possible in the **fields** frame. As an example, assume that your data is all in a soup. You would pass information that allowed you to construct a query that returned the soup data of interest. This may be the index to search on plus the key to search for. You may even pass the query frame itself. Note that any changes made from the time the print/fax/... request is made to the time that the printing occurs will be reflected in the printed items. This may not be what you want. As in most cases, there are tradeoffs.

Q *I'd like to have something like a viewIdleScript in my communications endpoint. The endpoint proto doesn't contain a way to do this. What's a good way to do it?*

A One solution is to include your endpoint in a view object, such as a clView. You can then treat the whole clView as the communications object. You can use the view messages associated with opening a view to manage the control of your endpoint. Similar things can be done with viewQuitScript. And of course you can use a viewIdleScript.

Also note that using a view gives you a way to provide visual feedback (assuming it makes sense). Take a look at protoLlamaTalk in the LlamaTalk sample on this issue's CD for an example.

Q *How do I get a text view to redraw itself with a new font?*

A Simply use SetValue on the viewFont slot:

```
SetValue(theParaView,
   'viewFont,
   {family: 'espy, face: kFaceBold, size: 12});
```

Q *I'm trying to use a view that has vfFillGray as the background. I've found that it looks really bad. Why is this?*

A The pattern vfFillGray is an alternating on/off checkerboard pattern of pixels (i.e., 50% gray). With the current state of the technology, all passive LCDs have problems displaying large areas of 50% gray (or large areas of black). The problem (called *crosstalk*) is worst when you have alternating on and off pixels. Basically, the LCD freaks out. You should avoid using large areas of 50% gray where possible.

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

A That particular exception indicates that an entry in your soup has a value of NIL in the slot you're trying to create an index for (i.e., the entry contains the slot with a value NIL). You can easily recreate this error by trying to add an index on the **bday** slot in the names file. Here's some code that you can type in the inspector:

```
call func()
begin
   local nameSoup := GetStores()[0]:GetSoup(ROM_cardfilesoupname);
   nameSoup:AddIndex(
      {structure: 'slot, path: 'bday, type: 'Int});
end with ();
```

In this case the error occurs because the default cardfile entry has a value of NIL for the **bday** slot. The solution is to make sure that there are no soup entries with a value of NIL for the slot that you want to use for the new index. This is best done in the design of your soup data.

If this isn't possible, the only solution is to make sure that all entries in the soup either have a valid value for the new index slot or do not contain the new index slot. Unfortunately, you don't know in advance if the new index will fail. In this case you

can wrap the code that adds the index in a try/onexception clause. If an exception occurs that has the -48019 error number, you know that you have to iterate through the soup and fix entries.

Also note that you may want to keep a list of those “fixed” entries around since you may have to unfix them after the index has been added. In other words, it’s OK for an entry to have a NIL value in an indexed slot after the index has been added to the soup.

Q *I have a protoA2Z_TDS controlling a protoTextList. There are two things this combination doesn’t do: (1) As the protoTextList contents are scrolled, the protoA2Z_TDS doesn’t update the current letter, and (2) when the user clicks on a letter in the protoA2Z_TDS, I want to scroll the protoTextList to the appropriate place. How do I do these things?*

A For those who may not know, protoA2Z_TDS is sample code provided by Newton Developer Technical Support. In answer to the first question, all you need to do is set the curIndex to the correct value, where A is 0 and Z is 25. If you use SetValue, the display will update for you. So if your protoA2Z_TDS was declared as **indexer**, and you wanted to change it to the letter B, you would do this:

```
SetValue(indexer, 'curIndex, 1);
```

You could also write a method of the protoA2Z_TDS that would update the display based on a character:

```
SetIndex := func(newChar)
begin
   local newIndex := Ord(Upcase(newChar)) - 65;
   if newIndex < 0 then newIndex := 0;
   if newIndex >= numIndices then newIndex := numIndices - 1;
      SetValue(self, 'curIndex, newIndex);
end;
```

Note that this function will try to do the right thing with weird input. However, if you’re expecting the full range of Unicode values, you’ll have to change the function to accommodate multibyte characters.

Now that the easy one is done, it’s time to tackle question number 2. You need to know about three things in the protoTextList that aren’t yet documented in the *Newton Programmer’s Guide*:

1. There's a slot named lineHeight which contains the height of each line in pixels.
2. The protoTextList uses SetOrigin to scroll. Therefore, the slot viewOriginY contains the number of pixels that the view is scrolled (and viewOriginY DIV lineHeight is the line number of the top displayed line).
3. There's a method DoScrollScript(offset) that scrolls from the current position by the specified offset (in pixels).

Given these three pieces of information, here's a method for a protoTextList that will highlight a particular row and make it visible:

```
protoTextList.HiliteRow := func(index)
begin
   // highlight this item
   SetValue(self, 'selection, index);

   // scroll as necessary
   local topItem := viewOriginY DIV lineHeight;
   if index < topItem or (index >= topItem + viewLines) then begin
      // we need to scroll so that the index is the first item
   :DoScrollScript(- (topItem - index) * lineHeight);
   end;
end
```

Of course you still have to calculate what index to pass to the function. But that should be fairly straightforward. The protoA2Z_TDS will give you the first letter, which you can then find in your listItems array in the protoTextList. Note that if the listItems array is sorted, you can use a binary search to find the correct index.

Q *What is the origin of the Llama?*

A The first evidence of Llamas dates back to the Llama Raptor discovered by Dr. Leaky in the jungles of the Amazon. This find was dated back to the late "Jurassic Park" period. Early Llamas are thought to be both more violent and intelligent than today's breeds. Cave paintings from the hills of Venezuela clearly depict early humans in use as pack animals for tribes of Llamas.

It is not clear what happened to these early Llamas, and how they made the transition from a relatively intelligent animal, to today's humble animal. One can still find rare exceptions among Llamas (myself as a notable example), however the general level of education amongst Llamas has fallen below kindergarten.

Scientists are still trying to find the link between old Llama and new, but so far the Missing Llama Link has evaded their best efforts. At one time in the 1920's it was thought the link had been found, but it turned out to be the claws of a tiger and the rib cage of a modern Llama. Both had been artificially aged. The hoax was called Piltdown Llama.

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 Erik York and our Newton Partners for the questions used in this column, and to J. Christopher Bell, Bob Ebert, Mike Engber, Neil Rhodes, Kent Sandvik, Jim Schram, Maurice Sharp, and Bruce Thompson for the answers.

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