NEWTON Q&A: ASK THE LLAMA

Q *How can I open an application so that it displays a particular data item?*

A If the target application supports Find, you can do that as long as three things are true:

- You know the application symbol.
- You know the application soup name and data format.
- The application supports the ShowFoundItem message.

If all of these are true, you can send the application the ShowFoundItem message with the appropriate arguments. Check the *Newton Programmer's Guide* for the arguments to ShowFoundItem. Be aware that not every application takes a soup entry as one of the arguments. That's why you need to know the application's data format. You can check whether the application supports the ShowFoundItem message with the following code:

```
local theApp := GetRoot().(kAppSymbol);
if theApp AND theApp.ShowFoundItem exists then
   // application installed and supports the message
```

Q *We have an application for a Newton device that communicates with the desktop. Because of the structure of our data, we would like to be able to request a particular NewtonScript object. We thought of sending the reference or address of the NewtonScript object to the desktop and using that as the identifier, but we could find no way to do this. Are we missing something?*

A Unfortunately (or fortunately, depending on your point of view), Newton 2.0 OS doesn't provide a way to get the memory address of an object. Actually, since NewtonScript can relocate objects at will, providing an address would not be a good idea. There's an alternate approach: you can maintain an array of the objects you want to export. The array index can be used in much the same way as the address. As an example, in the code below, the memory "address" for **object2** would be 1. In other words, **myObjectArray[1]** would give you **object2**.

```
object1 := "foo";
object2 := {can: 'aid, eee: "an", a: "...yep"};
object3 := [1,2,3];
myObjectArray := [object1, object2, object3];
```

If you need to indicate that an object has already been transferred to the desktop, you can simply replace the object at the relevant array index with NIL.

Q *I'm designing my data structures. I figure I could use either two cursors onto two different soups or two cursors onto the same soup. Which is the more efficient solution?*

A You can measure efficiency in two relevant ways: by time or by memory usage (or both). The time to create the two cursors will be the same regardless of the number of soups. More heap space will be required for two soups. With two soups, it may take less time to find items that exist in just one soup than when searching a larger, combined soup. However, with two soups you won't get as much benefit from the operating system's caching of the entries; there's more overhead information to swap in and out of the heap, which increases the time required to get data.

The real answer is to test it with your actual data and see. Overall, two cursors on one soup sounds like the more efficient way to go. Your question implies that you're going to have two completely different sets of data. You can do this in one soup by using indexes, because entries with either no indexed slot or NIL in an indexed slot won't participate in that index. That is, when you create a cursor that uses that index, entries with NIL values will be ignored by that cursor.

Something that might occur to you is using tags to implement the two different sets of data (that is, each set would have a unique tag value), but this doesn't work as well as using an index. With an index, you can navigate to an entry in O(log *n*) time, where *n* is the number of entries that are in that index. In other words, the time taken to navigate to a particular entry will be directly related to log of the total number of entries. If your query includes a subrange denoted by beginKey/endKey or startExclKey/endExclKey, the system finds that subrange very quickly. It can then quickly step through entries in between.

As far as tags are concerned, the operating system gets the set of tags for an entry efficiently, but it has to know which entries to get the tags for first. So with no other way to narrow the search, it will check all the entries, assuming you aren't using an index. Getting the tags is actually very efficient, but indexes work better for subranging.

Q *My communications program has a number of standard packets of information. I'm trying to set up constants for each of these standard packets. However, for the packet*

```
constant kHaltAndCatchFireMessage := "\u102cff1003";
```

Newton Toolkit complains that there's an "odd number of digits between \u's." I count ten, which looks even to me. Does Newton Toolkit need a remedial math course?

A Actually, Newton Toolkit is doing fine in math, but it should say "bytes" instead of "digits." There are ten hex digits in your string, but there are two hex digits per byte, so your string is five bytes long. Unicode is a double-byte representation, so there are four hex digits per Unicode character (two per byte). Since you have ten hex digits, you have two and a half Unicode characters, which is an invalid Unicode string.

You can either add two more hex digits to your string or use the MakeBinary and Stuff... functions. If you're dealing with data that's not strings, the latter method is the best one for compatibility. It's also likely to keep you saner.

Q *I'm trying to compile a program that works with both Newton 1.x and Newton 2.0 OS devices.; however, it won't compile. Newton Toolkit complains that I have a bad magic pointer, but I know that the value is defined in the MessagePad platform file. The offending code is as follows:*

```
local theCountries :=
   call kGetUserConfigFunc with ('commonCountries);
if ClassOf(ROM_Countries) = 'frame then
   // on a 1.x unit
   labelCommands := foreach item in theCountries collect
      ROM_Countries.(item).name;
else
   ...
```

This would give a really nice pop-up menu of countries on a 1.x unit. Why doesn't it work?

A This is a subtle problem. In Newton Toolkit 1.5 and later there are certain functions called *constant functions* that will evaluate at compile time when their arguments are constant. The most common ones are the GetLayout function, which will return a reference to another Newton Toolkit layout, and LocObj. The ClassOf function is another one of these.

At compile time, a magic pointer is considered a constant value. That means that the ClassOf call in your conditional is executing at compile time. Of course, there's no

Newton device around at compile time, so Newton Toolkit is unable to dereference the magic pointer. Hence the error.

One workaround is to set a local to the value of the magic pointer and use that local in your conditional. This works because the value of the argument to ClassOf is no longer a constant, so it will not be called at compile time.

```
local mpCountries := ROM_Countries;
if ClassOf(mpCountries) = 'frame then
   ...
```

Q *I'm trying to dial the following number using a Newton Fax Modem with my MessagePad 130:*

"18005551234,,,,,,,,1,,,,408-555-1234,,,,123-456-789-123,,,"

I get an error -16013 in my communications code whenever I do this. I need to use the long string because it contains a calling card number. My modem dials correctly and the modem at the other end picks up. I even hear the chirping whistle of exchanging bits. But suddenly things just stop and the error occurs. Any clues?

A Yes. First star on the right, then straight on till morning. But that's a different story. In answer to your question, it looks like you're timing out on the connection attempt. Modems have a set amount of time to establish a connection, and the commas are reducing the time they have.

Each of the commas will insert a delay into the dialing. For most modems, the time for each comma is controlled by register S08 and usually defaults to 2 seconds. You have 19 commas, so that's 38 seconds, which leaves very little time for the modems to sync up (the chirping whistle exchange you're hearing).

The solution is to increase the timeout of the modem to a more reasonable value. When you're thinking about the timeout, remember that each digit will take around 95 milliseconds to dial. There will also be a line connection time of about 2 seconds, a ring time of a few seconds, and the final sync-up or negotiation time of 2 to 15 seconds. You should increase your timeout values to at least 60 seconds. If that doesn't work, add 30-second increments. You can do a binary search to narrow it down to an optimal value.

To set the timeout for the modem, use 60 for the waitForCarrier (sixth) argument of the kCMOModemDialing option. The following bit of code will do this:

```
// make a modem option data structure based on user preferences
local option := MakeModemOption();
// modify the timeout value
option.data.arglist[6] := 60;
// set that option in your endpoint
ep:Option(option);
```

Q *How can I tell if a tap is the first of a double tap?*

A Unfortunately, the RUM (Read User Mind) ASIC didn't get completed in time for Newton 2.0 OS, so we were unable to implement the IsFirstTap global function. We also looked at a wireless link to one of the 900-number pay-by-the-minute psychic lines but couldn't figure out how to bill the user.

But seriously, you can't tell. The best you can do is to hold off processing the first tap for some amount of time. If you receive another tap in that time, it's a double tap. The drawback is that if it isn't a double tap, you've lost the unit parameter from the first tap, since you can't save this parameter.

The other option is to follow a user interface guideline: Make the second tap an extension of the functionality that happens with the first tap. As an example, in any text input view on a Newton device, the first tap selects a word and the second tap brings up the corrector slip. Because the double tap action is an extension of the single tap, there's no need to handle the first tap in a special way.

Q *We have a problem with union soups. We have an application that creates soups and transfers them to a PC. The soups can get sent down to a different MessagePad. If the user inserts a storage card and selects it as the default store, we can't successfully add an item to the soup. Our code does a GetUnionSoupAlways, then tries to add an entry using AddToDefaultStoreXmit. The Newton throws an exception that tells us there's no soupDef. We're sure that the soup doesn't exist on the store, but we thought that GetUnionSoupAlways creates the soup if you try to add something.*

One thing we thought of was to use RegUnionSoup, but our transfer application doesn't know what the soupDef is. Is there a way to copy a soupDef from one store to another or to get the soupDef from an existing soup?

A Well, first you need some good French onions, then some bread, mozzarella cheese ... oh, sorry, I thought you said "onion soup."

For a union soup to work properly in the Newton 2.0 OS, a soupDef must exist in at least one of two places. The soupDef can be registered with the OS via RegUnionSoup, or it can exist within a soup that's on a mounted store.

GetUnionSoupAlways should fail if there's no soupDef present. However, in the current release of the Newton 2.0 OS ROMs it doesn't. That means the problem is deferred until you first try to add an entry. This is when the OS tries to create the soup but can't find the soupDef. So that's why you get the error on the call to AddToDefaultStoreXmit. Of course, this doesn't help you, but there are a few options:

- Make sure that a soupDef is registered, via RegUnionSoup.
- Make sure that an actual soup exists on some store and that that soup contains an embedded soupDef. The soup doesn't actually have to have any entries. You can use the CreateSoupFromSoupDef function or the GetMember soup message to do this. For example:

  ```
  RegUnionSoup(kMySoupDef):GetMember(GetStores()[0]);
  ```

- Don't use union soups; instead, have your download application just send the store either the CreateSoupXmit or GetSoup message.
- Write some smart code that checks to see if a soup with the same name exists on any store and duplicate that soup on the new default store. If you use GetIndexes/CreateSoupXmit and GetAllInfo/SetAllInfoXmit, you should be able to make a reasonably similar soup.

Unfortunately, there's no supported way to directly access the soupDef of an existing soup.

Q *I have an application that performs some lengthy initializations in the installScript. I need a slip to come up and inform the user that this action is occurring. The problem is that the BuildContext slip I create at the beginning of the installScript doesn't show up when the installScript is running. How can I get a slip to come up in my installScript?*

A I assume that you do something like create the slip, send the slip an Open message and then do a tight loop with some initializations. If this is the case, the system has probably opened your view, but your installScript is still executing. That means the system cannot refresh the display.

One possible approach is to call RefreshViews to force the system to update the display. However, if your progress indication is dynamic, you'll have to call RefreshViews each time you change the progress slip.

A better approach is to use the DoProgress call, which provides a standard "Your Newton device is doing something" interface for the user. You may also want to do your initialization in a deferred action.

Q *Before I started using NewtonScript I used C. One thing that I really miss is a switch statement. Are you going to add one?*

A Good question. The answer is I don't know. However, there's a trick you can use to simulate a switch statement with the existing NewtonScript language. Actually, it gives you something much more powerful than a switch statement (which, for those who don't use C or C++, is basically a case statement).

The idea is to build a frame whose slot names are the possible cases in the switch statement. You can even add a default item, and you can add slots to frames at any time, so you effectively have a dynamic switch statement. Better yet, you can define your basic switch frame as a constant and then create a dynamic switch frame that protos to it. As an example, consider a switch statement that does message dispatching:

```
DefConst('kMyDispatchFrame, {
   a: func(item) item := item + 4,
   b: func(item) if item < 4 then :a(item),
   default: func(item) item := 0,
});

// If we don't want to use a default case, the switch is
// really simple:
MySwitch := func(switchFrame, whichCase, dataItem)
   if switchFrame.(whichCase) then
      PerformIfDefined(switchFrame, whichCase, [dataItem]);

// For a default case, things become slightly more complex.
DoSwitchDispatch := func(switchFrame, whichCase, dataItem)
begin
   // See if the case exists in the frame.
   // If not, use the default item.
```

```
   if not switchFrame.(whichCase) then
      whichCase := 'default;

   return PerformIfDefined(switchFrame, whichCase, [dataItem]);
end;
```

As you can see, you have a very powerful switch-like mechanism. You can use a similar technique to get data values. In that case, your switchFrame value would be data of some type and you would do a simple lookup instead of a Perform.

Q *I'd like to add another item to the address picker pop-up list in my "To:," "Cc:," and "Bcc:" pickers that would allow the user to create an e-mail address without adding it to the Names soup. The user interface reasoning behind wanting to do this is to avoid cluttering the Names soup with addresses that are used only once. I've successfully added an item to the picker and caught the pickActionScript for it. The pick item that I want to use to add this temporary name appears in the protoAddressPicker pop-up list. Now I want to bring up an editor and add my temporary item. I tried the call*

```
GetDataDefs('|nameRef.email|):New(tapInfo, self)
```

from my protoAddressPicker's pickActionScript, but I got an exception: Object {class: nameRef.email, name: "E-Mail addresses", preferredRouting: [string.email], ...} is read-only. Why can't I create a new data object with this?

A Usually answers to these questions are reasonably self contained; this is an exception. Before you can understand this answer, you really need to read up on list pickers and data, or you'll be tripped up by the subtle differences between nameRefs and dataDefs.

The transport system uses a structure called a nameRef for information on where to send things. It so happens that nameRefs use the data definition registry as a repository. However, a nameRef is a different beastie from a dataDef.

To create a new empty nameRef structure, you can use the call

```
GetDataDefs('|nameRef.email|):MakeNameRef(tapInfo, self);
```

Then you can use some sort of floating editor to enter the values. I suggest using a protoFloatNGo that contains a newtFalseEntryView and then appropriate slot views for the fields of the nameRef that you want to edit.

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, Henry Cate, Bob Ebert, David Fedor, Ryan Robertson, Jim Schram, Maurice Sharp, and Bruce Thompson for these answers.

If you need more answers,
check out
http://devworld.apple.com/dev/newtondev.shtml on the World Wide Web.