# NEWTON Q&A: ASK THE LLAMA

This column first appeared in volume 23 of DEVELOP (the Apple technical journal for develo


**Q** *I have a program that communicates with the desktop. Part of the information sent is real numbers. I've found functions to stuff almost every other type of data into a binary object except real numbers. How do I do that?*

**A** You have two choices. First, you could just print the real number as a string (using SPrintObject), send the string, and convert it back on the other side. Clearly this isn't a good idea if you want to maintain a high degree of precision.

The other choice is to construct the correct type of binary object for the target desktop machine. In other words, take the Newton real representation and convert it into, say, IEEE floating point. Once you've constructed your target binary representation, you can use BinaryMunger to stuff that binary object into whatever packet of data you're constructing.

Note that Newton uses SANE representation for real numbers that are in the representable range. However, the representation of exceptions (such as NAN and infinity) are different and undocumented. At this time you should avoid converting these types of real numbers.

**Q** *Can you give me a short and clear description of the different types of Newton memory?*

**A** There are three important "pools" of so-called internal memory, each with different tradeoffs.

The NewtonScript heap (about 90 to 96K on current devices) is where all the runtime data from NewtonScript lives. Any result from the Clone family of calls will take up NewtonScript heap space. The view frame made at run time from your application templates will take up this heap space. NewtonScript heap space is very precious, and you should try to use as little of it as possible, especially when your application's base view isn't open.

The user store (192K in the MessagePad 100, larger on other devices) is where application packages stored internally live, and where soups are located. The entries in the soups are located in this space. While not quite as precious as the NewtonScript heap, this space can certainly run out. This is the space that's "extended" when a RAM PCMCIA card is inserted.

There is also some system heap space, which is used for, well, everything else. The viewCObjects and drawing objects live here. Recognition uses memory from here. You can run out of this space (in which case you get the Cancel/ Restart dialog) but it's less of a programming issue.

**Q** *I have an application that uses a protoRollBrowser. When I expand the items, they have lines separating them. I didn't put them there and I can't seem to get rid of them. Is this a bug?*

**A** What you're seeing is part of the default definition of a protoRollItem. It includes a 1-pixel border. You can remove that border by modifying the viewFormat of your rollItems. In addition, you may want to set the fill to white.

**Q** *I'm using a protoRoll (not protoRollBrowser) in my application. But it never shows up. What's the problem?*

**A** You need to give it a viewFlags slot and make sure the Visible bit is checked. The default is Application and Clipping, but this won't make the protoRoll visible if it's included inside another view.

**Q** *I have a text view that the user can use to enter text. I wanted to extend a selection. I knew the insertion caret was at the end of the selection, so I called SetHilite(newPoint, newPoint, nil), where newPoint is the new position for the selection extension, but I got no highlight. What's going wrong?*

**A** The behavior is actually perfectly correct. There's a not quite obvious interaction between the caret and SetHilite. As shown in the table below, how SetHilite behaves depends on four things: the **start** and **end** character positions (the first two arguments) being equal, the value of **unique** (the third argument), the presence of a previously highlighted selection, and the presence of the caret. Note that the following explanation refers to the case of a single paragraph view. If there are multiple paragraph views then with **unique** nil, it's possible to have multiple discontigous selections. Note that there can only be one selection in a single paragraph view.

| Highlight and unique | When start = end | When start <> end |
|---|---|---|
| No previous highlight, **unique** true or nil | If there's a caret, move caret; otherwise, no effect | Create new highlight from **start** to **end** |
| Previous highlight, **unique** true | Clear highlight and, if there's a caret, move caret | Create new highlight from **start** to **end**, remove old highlight |
| Previous highlight, **unique** nil | Extend highlight to include **start**/**end** | Extend highlight to include **start** and **end** |

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

**A** We'll assume that the Newton tries to reconnect shortly after losing the connection. In that case, the Newton doesn't generate a new connection ID, so your server probably acts as if the connection didn't close, while the Newton is acting as if it's establishing a new connection. Currently the only solution is to force the Newton to wait three minutes after an improper disconnect before trying to reconnect.

**Q** *I have a communications program that always sends a string of the same size to the desktop. The string is quite large, and I would like to preallocate it and fill it with a particular value. What's the best way to do this?*

**A** As with all things in programming, the answer is a tradeoff between space and time. Let's assume that you want a string of 2K characters filled with the character 'A', and that you control the contents of the string (that is, if you get user input, you make sure the input is a string).

The first option is to allocate the string at compile time. Note that you shouldn't allocate your string constant with a double-quoted string ("a string"), since typing 2K (less the terminator) characters is monotonous and error prone. The way to allocate the string is with the following SetLength trick:

```
constant kNumberOfUnicodeCharsForString := 2048; // 2K chars
DefConst('kMyBigString, call func()
   begin
      // SetLength uses Bytes, Unicode chars are 2 bytes each
```

```
      local aStr := SetLength("",
            2 * kNumberOfUnicodeCharsForString + 2);

      // initialize the string
      for i := 0 to k1KUnicodeChars - 1 do
         aStr[i] := $A;
      return aStr;
   end with ());
```

At run time you can clone kMyBigString and do what you need to fill it with characters. Note that the object is not a string; you would need to use StuffByte to put in individual characters.

The disadvantage of this method is that it puts a 4K object in your package (Unicode strings are two bytes per character). The advantage is that it's very fast: it averages less than one tick (60th of a second) for the clone.

If you can't afford the 4K in your package, you need to generate the string at run time. Using the above code at run time averages 52 ticks.

Another possible runtime method is to use smart strings, which allow you to preallocate strings and concatenate them in a more efficient way. The first attempt at doing this seems to be inefficient, at an average of 175 ticks:

```
// defined constant somewhere in your project
constant  kNumberOfUnicodeCharsForString := 2048;

local s := SmartStart(2 * kNumberOfUnicodeCharsForString + 2);
local l := 0;
for i := 1 to kNumberOfUnicodeCharsForString do
   l := SmartConcat(s, l, "A");
SmartStop(s, l);
```

However, simply concatenating two characters at a time reduces the average to 88 ticks; four characters reduces it to 44; and so on. A lesson here is that testing and measurement are your friends.

**Q** *I'd like to train my dog to code in NewtonScript. How can I do that?*

**A** I'm afraid the prospect isn't promising. Dr. J. L. Fredericks at SITAP (Stanford Institute for Training Animal Programmers) has been trying for ten years to train

different animal species to program computers. Although he's had some success training dogs to do simple programs, he says "Anything more than a simple statement is beyond them. No loops, no conditionals." Besides which, paws don't work well for moving mice. For Newton programming the best he has been able to achieve is training a rat to reset the Newton on command. "Never underestimate the usefulness of a ratset."

**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, Bob Ebert, David Fedor, Neil Rhodes, Jim Schram, Maurice Sharp, and Bruce Thompson for the answers. Thanks especially to Bob Ebert for the Newton memory description.

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