DEBUGGING

Introduction

This document addresses Newton Debugging issues that are not available in the currently printed documentation . Please note that this information is subject to change as the Newton technology and development environment evolve.

TABLE OF CONTENTS:

Debugging Q&As

Path Failed 16 Error (9/15/93)

Q: What does a Path Failed 16 run-time error mean?

A: This means you have made an invalid reference. Because local variables are assigned NIL when they are defined, this error is often caused by simply forgetting to

initialize the variable with a known value when it is created. For example, the code fragment below would produce a Path Failed 16 run-time error:

```
local vb, result;
result := vb.top;
```

References to Child Frames Causes Memory Problems (9/15/93) (Obsoleted by NTK User's Guide, Chapter 5, Debugging)

Q: When I keep references to the child frames stored in my application, I get "Frames Out of Memory" errors. What is the problem?

A: The problem is that the child frame references you keep in your application make it impossible for the garbage collector to free the child view frames when they are no longer needed. This is a problem especially if you keep track of many child frames.

Nil = Nil returns True (9/15/93) (Obsoleted by NTK User's Guide, Chapter 5, Debugging)

There are cases where errors in slot names may cause the application to run well and yet cause unexpected results. Here's an hypothetical example of a function that returns true when doing comparison of two different frames, even though two of the slots being compared are irrelevant:

```
func compareFrame(frame1, frame2)
begin
   IF (frame1.date = frame2.date) AND (frame1.month = frame2.month) THEN
      RETURN true;
   ELSE
      RETURN nil;

Frame1 := {
   day: 5,        // note day instead of date
   month: 12,
};

Frame2 := {
   day: 3,        // note day instead of date
   month: 12,
};

compareFrame(Frame1, Frame2)
#1A       TRUE               // huh, the frames are different!
```

Always check to see that a slot is properly named, or if it has the value nil by mistake.

Stack UnderFlow Errors (9/15/93)

Q: What does the message Stack underflow errors mean?

A: This is an internal error. If you encounter this message, please post a source code snippet to PIE Developer Talk (Bug Report board), AppleLink, so we can further investigate the cause of the error.

Memory Statistics and Stats (11/11/93) (Obsoleted by Newton Programmer's Guide Documentation, Utility Functions chapter)

Q: The documented memory function MemoryStatistics does not exist; how can I check the memory?

A: This was a documentation error, the new function is called Stats, and it displays both free heap space and used space, and returns free space in bytes as an integer. Stats will only report the NewtonScript heap statistics; the system has other memory domains for recognition, communications, OS, CObjects and so on.

Print Tricks (12/19/93)

You can't issue Print statements inside state frames or, in general, in Communications. This is because:
a) The serial port is already in use for some comms purpose; Beaming is an exception, but even in this case you can't print concurrently, because only one comms channel can be open at a time.

b) The Print statements generate a lot of interrupts, causing bad behavior on the serial line, and this might lead to sudden drops and hangs in the comms connections.

Here are some tricks for printing in these various cases. You could signal what is happening, using the Notification system. If you want to signal a quick Help message from within comms, you could do something like the following example:

```
GetRoot():Notify(kNotifyAlert, EnsureInternal("My Comms App"),
      EnsureInternal("Help, I'm dying!!!"));
```

Another technique is to create a special error array in the application base view, and add strings to this array from within comms, as in this example:

```
   AddArraySlot(GetRoot().kAppSymbol.DebugArray, "My Comms App:"
&& myDebugData);
```

Making Trace take effect immediately. (1/10/94) (Obsoleted by NTK User's Guide, Chapter 5, "Debugging")

If you change the value of "trace" in the middle of a script, nothing happens, because the interpreter only checks its value occasionally. Here's one way to make the interpreter look at the trace variable immediately:

```
trace := 'functions;
Apply(func () nil, []);    // make the interpreter notice
```

This is, of course, completely dependent upon implementation, and is subject to change without notice.

Don't Forget to Clean Up the Prints (2/21/94)

Any Print functions shipping with the application will slow down the performance. The slow-down depends on the case; inside an iteration a Print function will slow down the performance quite a lot, while in plain code blocks the performance loss is more neglible.

Here's a simple test done with a MessagePad detached from the NTK Inspector environment:

```
// non-print test with a simple for loop
func()
   begin
   local x := 0;
   tempTicks := Ticks();

   for i := 0 to 10000 by 1 do
   begin
      x := i;
   end;
   tempTicks := Ticks() - tempTicks;
   SetValue(value1, 'text, NumberStr(tempTicks));
end

// non-print test with a simple for loop
func()
begin
   local x := 0;
   tempTicks := Ticks();

   for i := 0 to 10000 by 1 do
   begin
      x := i; Print(i);
   end;
   tempTicks := Ticks() - tempTicks;
   SetValue(value2, 'text, NumberStr(tempTicks));
end
```

The non-print function took on average 100 ticks, while the print test took on average 976 ticks. The problem with the slow-down is that more ROM code needs to be called as part of the Print statement.

You could easily find all the Print functions using the NTK List function. Note also that Print statements will expose your code constructs to outsiders, so it makes sense to hide things from the Inspector window.

Check for Application Base View Slots (3/6/94)

Here's a simple function that will print out all the slots and the slot values in an application base view. This function is handy if you want to check for unnecessary slots stored in the application base view; these eat up the NewtonScript heap and eventually cause problems with external PCMCIA cards.

```
call func()
begin

   local s,v;
   local base := GetRoot().|ChickenClock:PIEDTS|; // name of the
app
   local prot := base._proto;

   foreach s,v in GetRoot().|ChickenClock:PIEDTS| do
   begin
      if v<> GetRoot() AND v<> base AND v<> prot AND v<> nil then
      begin
         Write ("Slot:" && s & ", Value: ");
         Print(v);
      end;
   end;
end with ()
```

Use Throw! (3/6/94)

Using Thow (exception handling) is a very powerful error signalling technique. Developers can signal errors using Throw from any place in their code; that is, you can issue a Throw from anywhere at almost any time. See the "Q&A Application Design" for a discussion of exception handling vs return values in code.