APPLICATION DESIGN

APPLICATION DESIGN 1
Introduction..............................1
“Memory Full” Problems and Size of Entries (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-23)..............................1
Optimizing Root View Functions (9/15/93)..............................1
Code Optimization (9/15/93)..............................2
Nested Frames and Inheritance (10/9/93)..............................2
Screen Sizes (10/9/93) (Obsoleted by the1.0 Newton Programmer's Guide p2-18)..............................3
To protoApp or not to protoApp (10/13/93)..............................4
_parent._parent Assumptions (12/19/93)..............................5
One Copy of Global Functions and Variables (6/7/94)..............................5
Global Name Scope (6/7/94)..............................6
Beware of "self <dot>" (6/7/94)..............................6
Removing Same Package (6/7/94)..............................7
Preventing an Application from opening (screen too small) (6/9/94)7
Exceptions vs return codes, performance? (6/9/94)..............................8
Strategies and Limitations for Digital Ink on the Desktop (9/28/94)10
Splitting an Application into Multiple Packages. (12/7/94)..............................11

Introduction

This document addresses Newton Application Design 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.

“Memory Full” Problems and Size of Entries (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-23)

Q: Is there a way to figure out the size of an entry before it is stored in a Soup? We would like to calculate the sizes before we store entries in order to avoid “memory full” problems.

A: It is impossible to figure out the final size of an entry before storing it, because entries are compressed in various ways. The best approach is to save entries and check for exStoreErr exceptions.

Optimizing Root View Functions (9/15/93)

Q: I've got this really tight loop that executes a "global" function. The function isn't really global, it's defined in my root view and the lookup time to find it is slowing me down. Is there anything I can do to optimize it?

A: If the function does not use inheritance or "`self`", you can speed things up by doing the lookup explicitly once before executing the loop, and using the call statement to execute the function within the body of the loop.

Here's some code you can try inside the Inspector window:

```
f1 := {myFn: func() 42};
f2 := {_parent: f1};
f3 := {_parent: f2};
f4 := {_parent: f3};
f5 := {_parent: f4};
f5.test1 := func () for i:=1 to 2000 do :myFn();
f5.test2 := func() begin
      local fn := myFn;
      for i:=1 to 2000 do call fn with ();
      end

/* executes with a noticeable delay */
f5:test1();

/* executes noticeably faster */
f5:test2();
```

Note: Use this technique only for functions that don't use inheritance or the self keyword.

This trick is analogous to the Macintosh programming technique of using `GetTrapAddress` to get a trap's real address and calling it directly to avoid the overhead of trap dispatch.

Code Optimization (9/15/93)

Q: Does the compiler in the Newton Toolkit reorder expressions or fold floating point constants? Can the order of evaluation be forced (as with ANSI C)?

A: The current version of the compiler doesn't do any serious optimization, such as folding constants, eliminating subexpressions, or reordering functions; however, this may change in future products. In the meantime, you need to write your code as clearly as possible without relying too heavily on the ordering of functions inside expressions.

The current version of the NTK compiler dead-strips conditional statements from your application code if the boolean expression is a simple constant. This feature allows you to compile your code conditionally.

For example, if you define the DEBUG_MODE constant in your Project Data file and have in your application a statement conditioned by the value of DEBUG_MODE, the NTK compiler removes the entire if/then statement from your application code when the value of DEBUG_MODE is NIL.

```
constant DEBUG_MODE := true;        // define in Project Data
if DEBUG_MODE then Print(...);      // in application code
```

When you change Project Data so that the value of the DEBUG_MODE constant is NIL, then the compiler strips out the entire if/then statement.

Nested Frames and Inheritance (10/9/93)

Unlike C++ and other object oriented languages, NewtonScript does not have the notion of nested frames obtaining the same inheritance scope as the enclosing frame.

This is an important design issue, because sometimes you want to enclose a frame inside a frame for name scoping or other reasons. If you do so you have to explicitly state the messages sent as well as explicitly state the path to the variable:

Here's an example that shows the problems:

```
myEncloser := {
   importantSlot: 42,
   GetImportantSlot := func()
      return importantSlot,

   nestedSlot := {
      myInternalValue: 99,

      getTheValue := func()
      begin
         local foo;
         foo :=  :GetImportantSlot();           // WON'T WORK
         /* actually creates an undefined slot */
         foo :=  myEncloser:GetImportantSlot(); // MAY WORK

         importantSlot := 12;                   // WON'T WORK
         myEncloser.importantSlot := 12;        // MAY WORK

      end
   }
};

myEncloser.nestedSlot:GetTheValue();
```

The workaround is to give the nested frame a _parent or _proto slot that references the enclosing frame. Nesting the frame is not strictly necessary in this case, only the _proto or _parent references are used.

Screen Sizes (10/9/93) (Obsoleted by the1.0 Newton Programmer's Guide p2-18)

This is a friendly reminder that your applications should be screen-size indepenedent. There will be Newton products with other screen sizes, such as VGA (640x480) and fractions thereof (240x320).

This means you should develop with smaller and larger screen sizes in mind. This is supported in the current version of Newton software.

There are 2 main ways to make sure you application works at different screen sizes.

1. Develop and design based on the smallest standard screen (320x240)

2. Develop and design fully dynamic resizeable applications

There is support in the system to help you write dynamically sized applications, in the form of this function:

```
GetAppParams()
```

This function call returns a frame describing the current device view bounds and the location of the buttons. The return frame currently looks like this:

```
      {appAreaTop: 0,
       appAreaLeft: 0,
       appAreaWidth: 240,
       appAreaHeight: 320,
       buttonBarPosition: 'bottom}
   // buttonBarPosition is one of 'top, 'bottom, 'left, 'right
```

You can use this call to set the viewbounds of your base application view during at viewSetupFormScript time. The simplest way is to create a function that will resize your view for you and call it from the viewSetupFormScript.

Here is an example for "full screen" using RelBounds with a couple of constants. This example limits the size to no larger than the original Newton MessagePad:

```
constant kMaxAppWidth := 240 ; // original MP width
constant kMaxAppHeight:= 336 ; // original MP height

local b := GetAppParams() ;
// make view no bigger than the original MP
viewBounds := RelBounds(b.appAreaLeft, b.appAreaTop,
                     MIN(b.appAreaWidth, kMaxAppWidth),
                     MIN(b.appAreaHeight, kMaxAppHeight));
```

Once your base application view is sized correctly, you can use the `viewJustify` slot to make other parts of your application parent-relative or sibling-relative justified. Not all parts of your application need to be full- or center- justified, some parts will not be affected by a slightly smaller screen size.

Other parts of your application may need to be dynamically constructed (for instance, a view that has multiple children representing lists of soup entries, like the checkbook).

Your application base view should include a border. That way if your application is run on a screen that is larger than the maximum size you define, the user will be able to see the edges of it.

Warning: Do not use full justification in your base application view. The root view may be larger than the visible area of the screen.

Warning: Do not rely on upper left global coordinate of your application being fixed. On a larger screen it may be possible to move your base application around the screen. If you work with global coordinates in your application, make sure you check the current location of your base view (send a `GlobalBox` message to your base view).

If you follow these simple guidelines, your application should work on future Newton products.

To protoApp or not to protoApp (10/13/93)

Q: Should I use protoApp as my base application view? If not, what do I need to set?

A: It is not required that you use protoApp as your base application view. It does provide a nice encapsulated proto with which to do quick testing, but it has some problems. (It is difficult to add buttons to the statusBar, for instance).

There are some important points to remember about your base application view:

• Use an appropriate template for your base view, generally this will be a clView.

• The viewFlags slot should have the vApplication bit set (You do not need the vVisible flag, NTK will actually turn off the vVisible flag in the topmost view of the layout marked as the "main layout" if you have it set.)

• The base view will generally do a declareSelf: 'base so that a close or cancel box (like those in a statusBar) can close it.

• It may contain a protoTitle for an application title. A protoTitle defaults to drawing itself centered in the top of the parent (you could change this). You also do not need to set the title slot of the protoTitle, you can set it in your base view and it will be found by inheritance.

• It may contain a protoShowBar for filing (though make sure you set appropriate slots in your base view for filing, see the filing chapter in NPG and the appropriate Q&A).

• It may contain a statusBar.

_parent._parent Assumptions (12/19/93)

Be careful when directly using the `_parent` slot, if possible use the `Parent` method instead.

Because of the way functions are implemented in the MessagePad, _parent within a function body will evaluate to the current message context. For example, "`_parent = self`" will be true. If you must use `_parent` (because the `:Parent` method is not available) make sure and use `self._parent`.

The same is true of traversing `_proto` chains; if possible use inheritance rules instead of hard-coded paths concerning proto inheritance.

One Copy of Global Functions and Variables (6/7/94)

Q: Are global functions and variables shared amongst applications? In other words, does the system store a shared library entry for multiple references to a particular function or global?

A: Yes. NewtonScript is fully dynamic, keeping track of entries in memory for you. Thus, calling the same function from multiple instances in your application does not hamper memory performance. However, user defined variables and functions will be compiled into packages created with NTK, so each application package downloaded will have its own copy of user protos and functions or constants defined in the Project Data file for that application.

Global Name Scope (6/7/94)

Note that in NewtonScript, global functions and variables are true globals. This means that you might get name clashes with other possible globals, and as this system is dynamic you can't do any pre-testing of existing global names.

Here are two recommended solutions in order to avoid name space problems:

Use your signature in any slot you create that is outside of the domain of your own application.

Unless you really want a true global function or variable, place this inside the base view. You are actually able to call this function or access this variable from other applications, because the base view is declared to the root level.

For instance, we might have the following function declared inside the base view:

```
|MyBaseView:MySIG| := {
   viewClass: clView,
   declareSelf: 'base,
   TestThisView := func()
   begin
      // bla bla bla
   end
};
```

If you now call the function from another template that is not a child of the base view, you might do this:

```
|MyBaseView:MySIG|:TestThisView();
```

Beware of "self <dot>" (6/7/94)

The notation "`self.someSlot`" does not use `_parent` inheritance, which many people seem to expect. So, if you write code such as:

```
self.myParentSlot := 42;
```

then even if the parent frame has a slot named `myParentSlot`, the code above will create another slot inside the current frame. This might cause all kinds of strange problems.

Removing Same Package (6/7/94)

Q: I want to do a self-deleting package, so that after it is run once it erases itself. This is what I have, but it isn't working. Can you tell me what's wrong?

```
viewQuitScript:func()
begin
   local myDelayedAction := func()
      begin
      foreach package in GetPackages() do
      if (StrEqual(package.title, "packageName:signature")) then
         RemovePackage(package);
      end;
   AddDelayedAction(myDelayedAction, '[], 1000);
end
```

A: Your code is trying to remove its own statements as part of the remove process when the package is removed. That's why you are getting the sudden restarts.

Here's a more appropriate way to remove the same package inside a viewQuitScript:

```
// inside the Project Data file
DefConst('kRemoveMeFunc, func()
begin
   foreach package in GetPackages() do
      if (StrEqual(package.title, "packageName:signature")) then
         begin
         RemovePackage(package);
         break;
         end;
end);

   viewQuitScript: func()
   begin
      AddDeferredAction(EnsureInternal(kRemoveMeFunc), '[]);
   end,
```

The function passed to the `AddDeferredAction` is wrapped inside an `EnsureInternal` so the function doing the removing is guaranteed NOT to be part of the package being removed.

You can also create an "Auto Dispatch Auto Part" to do something like this. This is a package which does not appear in the extras drawer. It has an InstallScript, and once the script is called the package removes itself. Documentation on creating these parts can be found in the "BitParts" article by Mike Engber and in the "Q&A Newton Toolkit" document.

Preventing an Application from opening (screen too small) (6/9/94)

Q: I do not want my application to open if the screen size is too small. What's the best way to prevent it?

A: Check for whatever constraints or requirements you need early, probably in the `viewSetupFormScript` for the application's base view. In your case, you can do some math on the frame returned from `GetAppParams` to see if the screen is large enough to support your application.

If you do not want the application to open, do the following:

- Call `GetRoot():Notify()` to tell the user why your application cannot run.
- Set the base view's `viewBounds` to 0 width/height, so it is invisible.
- Possibly set (and check) a flag so expensive startup things do not happen.
- Possibly set the base view's `viewChildren` and `stepChildren` slots to NIL.
- call `AddDeferredAction(func(v) v:Close(), [self])` to close the view.

Exceptions vs return codes, performance? (6/9/94)

Q: What are the performance tradeoffs in writing code that uses try/onexception vs returning and checking error results?

A: I did a few trials, just to weight the relative performance. Consider the following two functions:

```
   thrower: func(x) begin
```

```
      if x then
         throw('|evt.ex.msg;my.exception|, "Some error occurred");
      end;

   returner: func(x) begin
      if x then
         return -1;  // some random error code,
      0; // nil, true, whatever.
      end;
```

Code to throw and and handle an exception:

```
   local s;
   for i := 1 to kIterations do
      try
         call thrower with (true);
      onexception |evt.ex.msg;my.exception| do
         s := CurrentException().data.message;
```

Code to check the return value and handle an error:

```
   local result;
   local s;
   for i := 1 to kIterations do
      if (result := call returner with (true)) < 0 then
         s := ErrorMessageTable[-result];
```

Running the above loops 1000 times took about 660 ticks for the exception loop, and about 310 ticks for the check the return value loop. So, if the exception occurs often, checking the return value is probably faster. (This could very well be because of the function call to `CurrentException` to get the message, which is probably slower than doing the table lookup.)

Running the same loops, but passing NIL instead of TRUE so the "error" does not occur was interesting. The exception loop and the return value loop both took about 180 ticks. So, when the error doesn't occur, setting up the try/onexception clause is about the same as checking the return value.

However, you can do better things with exceptions than you can with result codes. In the above case, if I want to stop the loop when the exception occurs, I am free to re-code it like this:

```
   local s;
   try
      for i := 1 to kIterations do
         call thrower with (nil);
   onexception |evt.ex.msg;my.exception| do
      s := CurrentException().data.message;
```

This code takes 130 ticks for 1000 iterations, significantly better than the return value case, where I'd have to check the result after each call to the function and stop the loop if an error occurred.

With exceptions, you can also handle the error at any level up the call chain, without having to worry about each function checking for and returning error results for every sub-function it uses. This will produce code that performs much better, and will be easier to maintain as well.

With exceptions, you do not have to worry about the return value for successful function completion. It is occasionally very difficult to write functions that both have a return value and generate an error code. The C/C++ solution is to pass a pointer to a variable that is modified with what should otherwise be the return value of the function, which is a technique best avoided.

As in the above example, you can attach data to exceptions, so there's no need to maintain an error code to string (or whatever) mapping table, which is another boon to maintainability. (You can still use string constants and so on to aid localization efforts. Just put the constant in the `throw` call.)

Finally, every time an exception occurs a message gets printed in the Inspector. (The timings were done with no Inspector connected.) This is also a boon to debugging, because you know something about what's going wrong, and you can set the `breakOnThrows` global to stop your code and look at why there's a problem. With result codes you have a tougher time setting break points. With a good debugger it could be argued that you can set conditional break points on the "check the return value" code, but even when you do this you'll have lost the stack frame of the function that actually had the problem. With exceptions and `breakOnThrows`, all the local context at the time the exception occurred is still available for you to look at, which is an immense aid.

Conclusion: Use exceptions. The only good reason not to would be if your error handler is very local and if you expect it to be used a lot--if that were true we could make a good case for re-coding the function.

Strategies and Limitations for Digital Ink on the Desktop (9/28/94)

Q: I want to capture electronic "ink" and send it to a desktop machine to be displayed. What is the best way to do this? I can't find a way to get a screen shot that doesn't use the NTK Inspector.

A: There is no general API for capturing all or part of the Newton screen. We are aware of the missing parts of the current API with respect to ink, and have plans to address this in future Newton OS releases.

For the current Newton OS, there are three approaches to consider, all of which have strengths and weaknesses.

Do it all myself approach.
Pro: Very simple, lots of control.
Con: Almost no help from Newton OS or other graphics libraries.

The "Different Strokes" sample code illustrates part of this approach. While user input is happening, you can get arrays of points representing the "ink" and turn them into drawing objects. Because these arrays are simple X, Y pairs, they can be easily transmitted to a desktop computer and rendered there.

Use Connection Kit approach.
Pro: Connection Kit does all the work for you.
Con: You have to use clEditViews on the Newton, and Connection Kit for the desktop machine.

You may notice that NCK displays ink from the paper roll. There is in fact some custom code inside the Connection Kit application which can translate the ink or points binary objects that are stored in `clPolygonViews` in a `clEditView`, and render this on the Mac or PC screen. This code is not accessible via MetaData for your soups.

However, when you export a soup to Newton Connection Native Format, the Connection Kit application will create a Mac PICT or Windows Metafile structure that represents the ink or points data. See the Connection Kit chapter of the Newton Programmer's Guide and the Q&A Connection Kit section for more details.

Hybrid approach. (Use a clPolygonView and hope.)
Pro: Can use standard Newton view classes and recognition; gives control over output format.
Con: Doesn't always work.

This technique relies on the `PointsToArray` global function to turn a binary object in the `points` slot of a `clPolygonView` into an array of X, Y coordinates. Like the first approach, you can export the resulting array to a desktop machine and render it.

This is an improvement over the first approach because you can let the Newton do "normal" shape recognition in the view, which allows the drawing of squares, rectangles, circles, ovals, and so on, and this vastly reduces the number of points. Even with shape recognition off, the user input is processed by the recognition system and redundant points are stripped, so the data is reduced.

The problem is that under some circumstances — when the user draws very slowly, for example — the input will not be recognized as a polygon shape and will instead be left as ink, in the `ink` slot. Data in the `ink` slot is in a proprietary and undocumented format, and no API in the current Newton OS can translate this format.

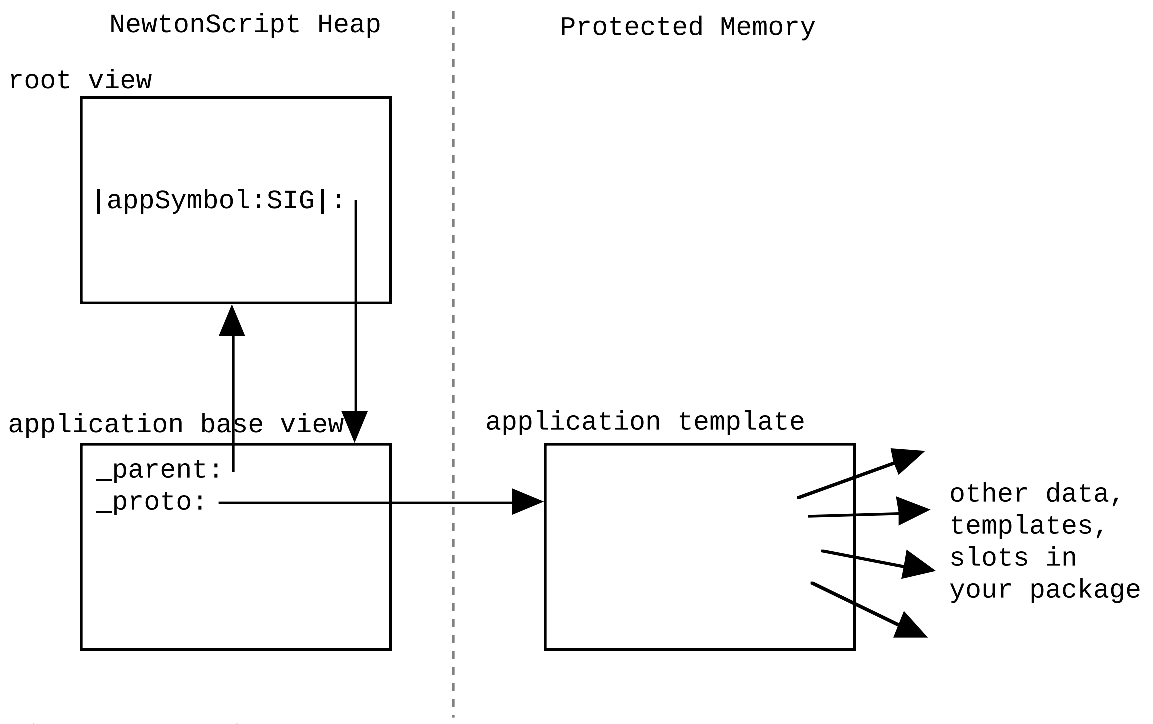
If you take this approach, we recommend watching for data as it's added (in the `viewChangedScript` for a `clPolygonView`, and possibly in the `viewAddChildScript` for a `clEditView`) and deleting new data or children if `ink` data is discovered. We hope that users will quickly learn to avoid drawing such that ink is created.

Splitting an Application into Multiple Packages. (12/7/94)

Q: How can I add templates or data to my application from another application? I'd like to add user protos to my main application from an "installer" app.

A: There are several ways. To understand the tradeoffs, you need to keep in mind what part(s) of your application reside in the NewtonScript heap, and what parts are in protected memory (in the package.)

When an app is installed, a view frame for the application's base view is created in the NewtonScript heap. This frame _proto's to the template in the package. When the app is closed, this base view frame is the only part that is in modifiable memory. (Unless your application's InstallScript registers with system services or does other custom things.)

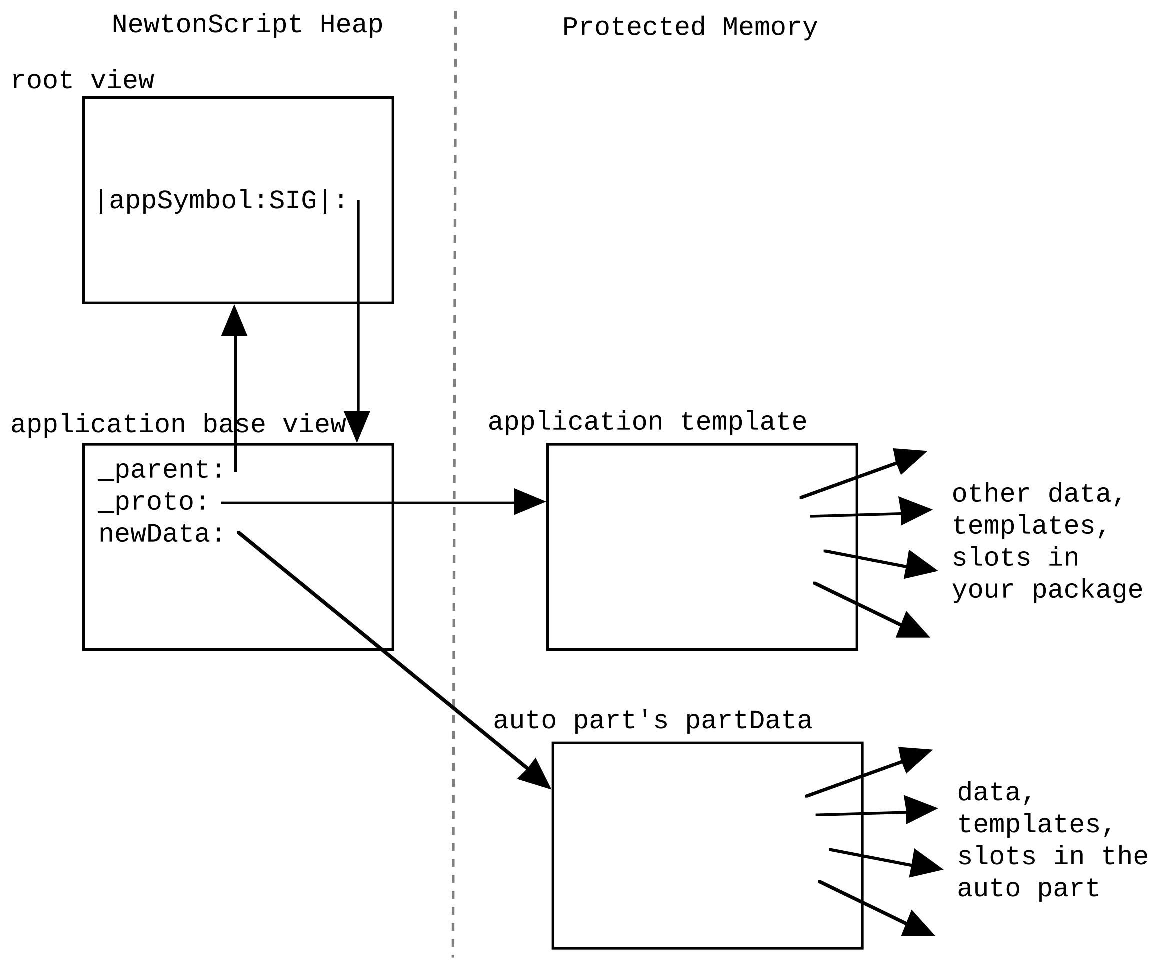


This provides one hook you can use to make data from a second package available to an application. Have the new package create a slot in the original application's base view which references the new package. This could be done for example from the InstallScript of an auto part.

The code would look something like this:

```
    GetRoot().|appSymbol:SIG|.newData := partFrame.partData;
```

which would produce this:

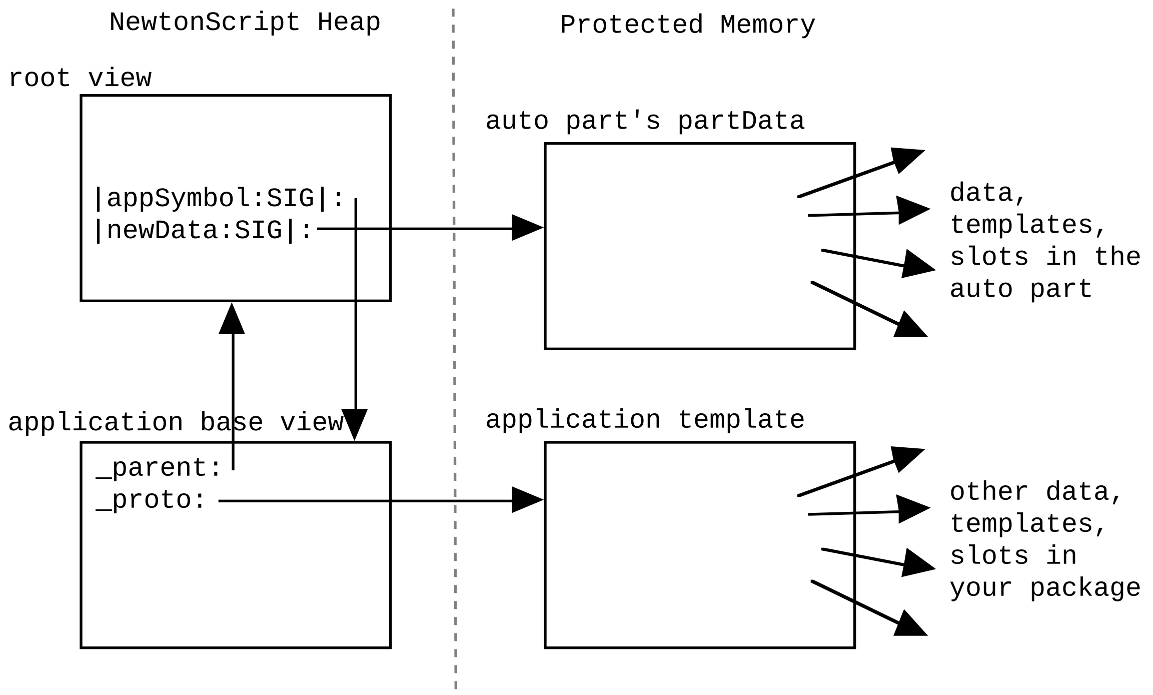


Now code in the application can access data from the second package through the (inherited) reference in the slot newData in the application's base view. The only space used in the NewtonScript heap is the extra slot in the application's base view, which takes very little space. However, the application must be installed before the auto part for this to work, so there is some dependency on the order of the packages which may not be desirable. Also, the auto part can only "install" into one application, so data from the auto part may not be sharable across multiple applications.

Another approach which makes the auto part's data more generally available is to have the reference to the data placed in a slot in the root view. If possible, avoid this approach because it clutters up the root view with application-specific data. The code in the auto part's InstallScript would look like this:

```
GetRoot().|newData:SIG| := partFrame.partData;
```

And the result is this:



The hit on the NewtonScript heap is still minimal. The application can access the data through the (still inherited) variable |newData:SIG|. Your signature must be used here because the root view is a shared frame and the signature prevents two different applications from trying to register with the same frame.

One final alternative is to create a new global variable with the reference to the data in the auto part. There is little difference between creating a new global and registering a new slot in the root view in terms of accessibility. To create a new global, the code is:

```
GetGlobals().|newData:SIG| := partFrame.partData;
```

Note that the signature is still required. Global name space is shared the same way slots in the root view are shared, and so the signature is required to avoid naming collisions.

In all of the above cases, care should be taken when accessing the data to handle the case where the auto part has not been installed and present a suitable error message to the user.

End of DTSQA

COMMUNICATIONS

Table Of Contents:

COMMUNICATIONS 2
Introduction..1
Newton Remote Control IR (Infra-red) API (6/9/94)..............................1
Stripping the High Bit from Incoming Bytes (10/8/93)..........................3
IR Transmissions and Power Drain (11/11/93)..3
Modem Initialization Strings (12/19/93)..3
Send Request Limitations with the Comms Architecture (2/21/94) 3
MNP Endpoint Issues (2/24/94)...4
Partial Scripts Don't Work (6/9/94)..4
SetInputSpec versus nextInputSpec (6/9/94)...5
Communications With No Terminating Conditions (6/9/94).................5
Modem Compatibility (3/8/94)...5
Flush the Buffers (3/8/94)...6
System Preferences and Endpoint Options (3/8/94)...............................7
Polling Endpoint Functions Undocumented..7
Disconnecting an ADSP Endpoint (3/8/94)...7
Dispose with No Connect Problem (3/8/94)...8
Wizard Communication Programming (3/8/94).......................................8
Maximum Speeds with the Serial Port (3/8/94)..8
Frame Transfers and the NewtonScript Heap (3/8/94).........................9
What is Error Code -18003? (3/8/94)...10
Do Not Pass View Frames To OutputFrame (3/16/94)..........................11
OutputFrame Requires An 8-bit Data Pipe (3/16/94)............................11
A Dynamic byteCount inputForm (6/7/94)..11
Connected endpoints and the powerOffScript (6/7/94).......................12
OutputFrame fails after Connect (6/7/94)...12
What Really Happens During Instantiate & Connect (6/14/94).........13
Endpoints & The ISO-OSI Reference Model (6/14/94)...........................15
IR Port Hardware Specs (6/15/94)..16
Serial Port Hardware Specs (6/15/94)...16
Unicode-ASCII Translation Issues (6/16/94)..17
The ioBusy Flag (7/8/94)..18
Timeouts and ADSP Connections (8/25/94)...18
Five States In The Life Of An Endpoint (8/25/94).................................18
Endpoint I/O Data Buffers & Flow Control (10/5/94)............................19
Newton Serial IR Endpoint (12/29/94)..21
Sharp IR Protocol (12/2/94)..23
Why is waitForCarrier ignored? (1/10/95)...29
IR Protocol Speeds (1/16/95)...29

FlushOutput and the nested event loop problem (1/16/95)................30
NEW
Using Serial PCMCIA Cards (04/26/95)...31

Introduction

This document addresses Newton Communications 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.

Newton Remote Control IR (Infra-red) API (6/9/94)

NTK 1.0.1 and future NTK development kits contains the needed resources to build applications that control infrared receive systems, consumer electronics systems and similar constructs.

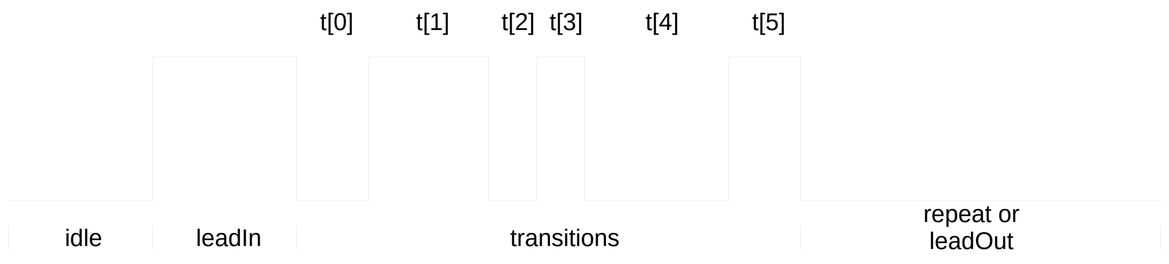
This development kit is fairly robust, and will produce send-only applications.

Note: The NTK 1.1 platforms file is required to produce code that will execute correctly on the MessagePad 100 upgrade units.

```
cookie := OpenRemoteControl();
```
Call this function once to initialize the remote control functions. It returns a magic cookie that must be passed to subsequent remote control calls, or nil if the initialization failed.

```
CloseRemoteControl(cookie);
```
Call this function once when all remote control operations are completed, passing cookie returned from OpenRemoteControl. Always returns nil. cookie is invalid after this call returns.

```
SendRemoteControlCode(cookie, command, count);
```
Given the cookie returned from OpenRemoteControl, this function sends the remote control command (see below for format of data). The command is sent count times. count must be at least 1. Returns after the command has been sent (or after the last loop for count > 1).



Each command code has the following structure:

```
struct IRCodeWord {
      unsigned long name;
      unsigned long timeBase;
      unsigned long leadIn;
      unsigned long repeat;
      unsigned long leadOut;
      unsigned long count;
      unsigned long transitions[];
};
```

name	identifies the command code; set to anything you like
timeBase	in microseconds; sets the bit time base
leadIn	duration in timeBase units of the lead bit cell
repeat	duration in timeBase units of the last bit cell for loop commands

leadOut	duration in timeBase units of the last bit cell for non-loop commands
count	one-based count of transitions following
transitions[]	array of transition durations in timeBase units

Note that the repeat time is used only when the code is sent multiple times.

See Remote.π, Sony.r, RC5.r, and RemoteTypes.r files for examples. The .rsrc files have templates for ResEdit editing of the Philips and Sony resources. See Remote IR Sample code for more details.

Things To Know Before You Burn The Midnight Oil:

If the Newton goes to sleep, the IR circuits are powered down, and any subsequent sends will fail. If you want to override this, you need to have a powerOffhandler close the remote connection, and when Newton wakes up the application could re-open the connection.

If two applications are concurrently trying to use the IR port (beaming and remote control use for instance), this will cause a conflict.

Sample Code:

The Remote IR Sample is part of the DTS Sample code distribution, you should find it on AppleLink and on the Internet ftp server (ftp.apple.com).

By way of a quick summary: the sample has an array of picker elements with the resource definitions bound to the index (ircode inside the application base view).

You specify the constant that is an index to the array, get the resource using GetNamedResource (see global data) and when you send data, use the constant as the resource used.

OpenRemoteControl is called in viewSetupFormscript, and closeRemoteControl is called in viewQuitScript. Note that these are methods, not global functions; same is true of SendRemoteControlCode.

More Information

Consult the IR samples available on AppleLink, ftp.apple.com (Internet) and on the PIE Developer CDs.

The following sites have more information about other infrared protocols:

nada.kth.se:home/d89-bga/hp/remote/remotes (Internet, ftp)
flash.ecel.uwa.edu.au (Internet, ftp)

Stripping the High Bit from Incoming Bytes (10/8/93)

Receiving ASCII strings over a 7-bit serial connection may cause problems in pre-1.3 system software. To overcome this problem change the inputForm to 'bytes, and pass the received array of bytes to this function for conversion to string format:

```
func(data)
begin
      local len := Length(data);
      local s := SmartStart(len);
      for i:= 0 to len - 1  do
```

```
            SmartConcat(s, i, chr(BAND(data[i], 0x7F)));
        return s;
end;
```

Future communications system software will provide a faster, more transparent mechanism to handle this situation.

IR Transmissions and Power Drain (11/11/93)

Q: Is there a power cost in turning the IR receive on and then off periodically?

A: In the MessagePad 100 and 110, the "Receive beams automatically" option powers on the SCC and IRDCU for 50ms every two seconds, resulting in a duty cycle of about 2.5%. The power drain this creates is negligible. In general receiving information is fairly efficient, but transmitting is approximately three times harder on the battery.

Modem Initialization Strings (12/19/93)

Currently there's no way to send modem initialization strings using the modem endpoint. One workaround is to use the serial endpoint and send initialization strings this way. Note however that the modem endpoint is handling many of the modem states, such as "No Carrier" exceptions and similar things, so you will need to re-implement most of these features.

Engineering will provide a solution for handling modem initialization strings; stay tuned for more information later.

Send Request Limitations with the Comms Architecture (2/21/94)

Each Output or OutputFrame call queues the data to be sent by the underlying comms architecture. In the case of the serial and modem endpoints, the current comms architecture allocates data structures for up to five send requests totalling 1024 bytes before blocking occurs. In the case of the ADSP endpoint space is allocated for as many as three, but because of a bug in the ADSP endpoint, additional output calls deadlock the system instead of blocking. The solution to this dilemma is to return control to the NewtonScript main event loop after three successive output calls, and as often as possible to allow the NewtonScript task to process the internal messages from the ADSP communications task.

Calling FlushOutput from NewtonScript blocks the NewtonScript task until the underlying system has more space for output requests. However, because this is independent of the underlying data transmission, you should check for any error messages from Output and OutputFrame, in order to narrow down problems with send requests and internal buffers. FlushOutput does not block the ADSP endpoint.

Note that this is only the case for outgoing requests. The buffer size for incoming requests is defined by the discardAfter slot value of the endpoint, which is a separate, generally larger buffer than the SCC interrupt-level one used by the low-level communications subsystem and is responsible for flow-control activation.

The BytesAvailable call will not be supported by the future communications architecture, as there will be a much better mechanism for handling asynchronous send requests.

MNP Endpoint Issues (2/24/94)

When using MNP options for an endpoint, it does not matter if you specify opSetNegotiate or opSetRequired; MNP always negotiates. The key options for MNP endpoint support are really specified in the data slot.

In addition, there is no need to specify the kCMOMNPAllocate option because kMNPDoAllocate is the default. The same is true for kMNPCompressionMNP5 if kCMOMNPAllocate is kMNPDoAllocate. By setting the compression option to kMNPCompressionV42bis, the system will "fall back" to MNP5 if V42bis is unavailable, and to None if MNP5 is unavailable. In general, however, it is better to state the desired options explicitly rather than relying on the system's default behaviour.

Partial Scripts Don't Work (6/9/94)

Q: Partial scripts don't seem to work. If I stick with input scripts, life is good. What am I doing wrong?

A: Partial scripts work just fine, but you must understand the mechanism completely in order to take advantage of it.

Let's say you Connect and SetInputSpec with inputScript, partialScript, and partialFrequency options.

The endpoint begins searching the input stream for the terminating condition (either a matching endCharacter value or for byteCount number of bytes). When the termination criteria is met, the inputScript is triggered. SetInputSpec must then be called again (usually from the inputScript) to begin another search.

Now, because you've specified partialScript and partialFrequency options in the endpoint configuration, a completely detached mechanism is also at work. During idle time, the endpoint checks to see if the partialFrequency has elapsed and if there is data in the input buffer. If so, the partialScript is triggered, and a copy of the data received is passed to the partialScript. The real data remains in the input buffer, however, which can be flushed via FlushPartial if desired. This process occurs independently of the inputScript mechanism.

Something to keep in mind:

If all your inputSpecs are partialScript driven (that is, no inputScripts) then you must post an Abort before disconnecting the endpoint. Without the abort, there will be a pending receive request sitting in the endpoint, even though no more inputSpecs are active. This is bad. If you have mixed inputScripts and partialScripts, then you can simply SetInputSpec(nil) before disconnecting.

SetInputSpec versus nextInputSpec (6/9/94)

Q: Setting the inputSpec is confusing. Under what conditions can I simply set the slot value nextInputSpec and under what conditions do I have to call SetInputSpec? I do both for safety.

A: You should always call SetInputSpec to guarantee future communications system software compatability.

Currently the endpoint calls ep:SetInputSpec(ep.nextInputSpec) automatically before returning from Connect and Listen, and after your inputScript returns. However, it is important that you explicitly call SetInputSpec to set the nextInputSpec slot, rather than simply setting the nextInputSpec yourself, to guarantee compatability with future communications system software.

Communications With No Terminating Conditions (6/9/94)

Q: How do I handle input that has no terminating characters and/or variable sized packets?

A: Remember that input specs are specifically tied to the receive completion mechanism. To deal with the situations of no terminating characters or no set packet sizes, you need only realize that one receive completion is itself a complete packet. Set the byteCount slot of your input spec to the minimum packet size. In your input script, call Partial to read in the entire packet, and then call FlushInput to empty everything out for your next receive completion.

If this is time-delay-based input, you may be able to take advantage of partialScripts with partialFrequencies. Call the Ticks function if necessary to determine the exact execution time of apartialScript.

Modem Compatibility (3/8/94)

Q: I would like to test fax and modem support in my Newton application. What modems work with Newton?

A: For fax, we require a "Class 1" modem. The modem must be "Hayes compatible", and capable of accepting commands over the serial line at 19.2kbps. You can customize the behavior of the modem endpoint using the NewtonScript modem options. At this time you must use SetOptions after Connect to set flow control options such as XON/XOFF if you are using no other error control (such as MNP) in your endpoint.

The built-in modem tool supports the following Rockwell chip sets:

Chip set	ID String
Rockwell RC224ATF	"242"
Rockwell RC96V24AC-W	"RC96V24AC-W !!!"
Rockwell RC9624AC-W2	"RC9624AC-W2!!!!"
Rockwell RC96ACL	"RC32ACL"
Rockwell RC144ACL	"RC32ACL"

You can determine if a modem contains any of these chips by sending the ATI4 command to the modem with a serial terminal emulator. If the response string is one of the above ID strings, the default modem endpoint will probably support the modem. Support for other types of modems is underway; for details, contact Newton DTS.

IMPORTANT:
While it may be possible to communicate with certain modems using a serial endpoint and "AT" commands, developers are strongly encouraged to use the provided modem endpoint. The built-in modem tool automatically reroutes to communicate with PCMCIA modems, whereas the serial tool does not. Using the modem tool also provides a coherent user experience-- it makes no sense for a user to be able to use your application to connect to a remote service with a Brand X modem, but not be able to fax with that same modem.

Flush the Buffers (3/8/94)

Q: I have a serial endpoint and when I call the output method, after a while I suddenly get strange -8007 errors. If I change what I'm sending over the serial port the error appears in other output messages I'm sending. What is going on?

A: This error is caused by limitations on the number of Output calls the endpoint can handle. The endpoint will allow at most five asynchronous Output calls (three in the case of the ADSP endpoint) before it will raise an exception with a -36008 error message .

A good general rule is to call ep:FlushOutput after large amounts of data are Output. Another good idea is to combine data before calling Output. For instance, instead of calling:

```
ep:Output("myName",nil);
ep:Output(unicodeCR,nil);
ep:Output("myPassword",nil);
ep:Output(unicodeCR,nil);
```

Do this:

```
ep:Output("myName" & unicodeCR &  "myPassword" & unicodeCR);
```

If you have further trouble with this, try calling FlushOutput after each Output call. Calling FlushOutput blocks until one or more outstanding outputs complete.

Note that the underlying send system will only buffer 1024 bytes at the current time, so if the total amount of send requests is bigger, the system will block.

System Preferences and Endpoint Options (3/8/94)

User preference settings are NOT reflected in the endpoint options slots. In other words, if you want to set these same options in endpoints you create, you need to fetch the values from the "System" soup (or from the userconfiguration global) and configure your own endpoints with the user-defined values.

For example, you may need to look at the user setting for "Require dial tone."

Polling Endpoint Functions Undocumented; Will Disappear (3/8/94)

Note that the following calls are no longer documented, and will be phased out in later ROM versions.   The reason for removing these calls is either that they are redundant, or that they are part of an early and ill-fated suite of calls intended to support polling.   A future API will provide much better functions to implement polling where required, as well as functions that will enable you to avoid polling altogether in most cases. If you must poll, try using partialScripts, which do not trigger your inputScript, to look for incoming data.

```
endpoint:InputAvailable()
endpoint:ReadyForOutput()
endpoint:OutputDone()
endpoint:BytesAvailable()
endpoint:Reject()
```

If you are using any of these calls at present, phase out their use to ensure compatability with future system software releases.

Disconnecting an ADSP Endpoint (3/8/94)

Due to a bug in the ADSP Disconnect routine, the endpoint must call CloseAppleTalk after Disconnecting an ADSP endpoint (either before or after Disposing of that endpoint). Also after a failed Connect if you dispose of the endpoint in the failure handler. Forgetting to do so will prevent you from printing to network printers, for example.

Dispose with No Connect Problem (3/8/94)

Q: I get -8007 errors if I dispose an endpoint that has not been used for a Connect statement. What is the problem?

A: This is a bug in Dispose that will be fixed in a future system release. One possible workaround is to attempt a Connect or Listen in a deferred action, then call Abort to stop the attempt in progress. Note that Aborting a Listen or Connect in progress will cause an exception to be thrown, which you should be prepared to handle.

Wizard Communication Programming (3/8/94)

Q: How does one use the Sharp Wizard import and export communication functions in an application?

A: The Sharp Wizard functionality in Newton MessagePad is limited to simple import functionality of Wizard data into the Newton environment, the Note soup, or the Card soup. There is no API for importing or exporting data. You need to write this functionality using the communications API.

Note that there is a specific unicode translation table defined for you by the system which will allow you to talk to the Wizard through an endpoint. To use this table, you must add the following slot to your endpoint frame:

```
encoding: kWizardEncoding,
```

This will allow you to properly import unicode strings from the Wizard. You can, of course, choose to use other data types, but in that case you'll have to provide your own scripting to convert bytes into useful data.

Maximum Speeds with the Serial Port (3/8/94)

Here are some rough estimates of the speeds attainable with the Newton serial port in combination with various kinds of flow control. These numbers are rough estimates, and depending on the protocol and amount of data (burst mode or not) you might get higher or lower transmission speeds. Experiment until you have found the optimal transmission speed.

- **0** to **38.4** Kbps
  No handshaking necessary for short bursts, but long transmissions require flow control (either hardware or XON/XOFF).

- **38.4** Kbps to **115** Kbps
  Require flow control, preferably hardware, but XON/XOFF should also work reasonably reliably.

- **115** Kbps +
  You will encounter problems with latency and buffer sizes. Speeds in this range require an error correcting protocol.

Both hardware and XON/XOFF flow control can be set with the kCMOInputFlowControlParms and kCMOOutputFlowControlParms options. In the case of hardware handshaking (RTS/CTS) you should use the following options:

```
{     label:      kCMOInputFlowControlParms,
        type:       'option,
        opCode:     opSetRequired,
        data:       {     xonChar:    kDefaultXonChar,
                          xoffChar:   kDefaultXoffChar,
                          useSoftFlowControl:     NIL,
                          useHardFlowControl:     TRUE, }, },

{       label:      kCMOOutputFlowControlParms,
        type:       'option,
        opCode:     opSetRequired,
        data:       {     xonChar:    kDefaultXonChar,
                          xoffChar:   kDefaultXoffChar,
                          useSoftFlowControl:     NIL,
                          useHardFlowControl:     TRUE, }, },
```

Frame Transfers and the NewtonScript Heap (3/8/94)

The Newton has a NewtonScript heap for all the RAM frame objects; it is designed to be a place for short-lived frame objects needed for visual display, temporary frames, cloned frames, frames put together for beaming, faxing and so on.

The default NewtonScript heap in the current Newton system is 90K, and drops down to about 70K after you start the system. Other message pads, such as MP 110, have a bigger NewtonScript heap (120K). Do not assume that the heap will always have this number; other Newton platforms may have different sizes. By using the Stats function you can find out the current heap size.

A heap of 90K may sound small, but the size value was designed with the idea that frame objects for display (and similar objects) do not stay around for long, and it is assumed that garbage collection will make room for more objects when needed.

Problem can arise, however, when sending large temporary frames to Newtons (via beaming for example). A special object reader will read in and create the incoming flattened frames sent by the OutputFrame comms method. If the frame heap is too small the reader can't reconstruct the frame, and the end result is usually unreliable transmissions.

A statistical max number for a transmitted frame is about 16K. Sometimes it may be possible to transmit 20K, if 16K is not enough.

Currently we recommend not sending large frame constructs in the same transmission. If possible, break down the transmitted frames to smaller chunks, and send them over one at a time. Also try to zero out unneeded information in slots, and remove unneeded sub-frames (_Proto, _Parent, and so on) and similar information from the transmitted frame.

See the SendCards sample for an example of how to send multiple items when the user routes a single item (that is, multiple beams for one beam).

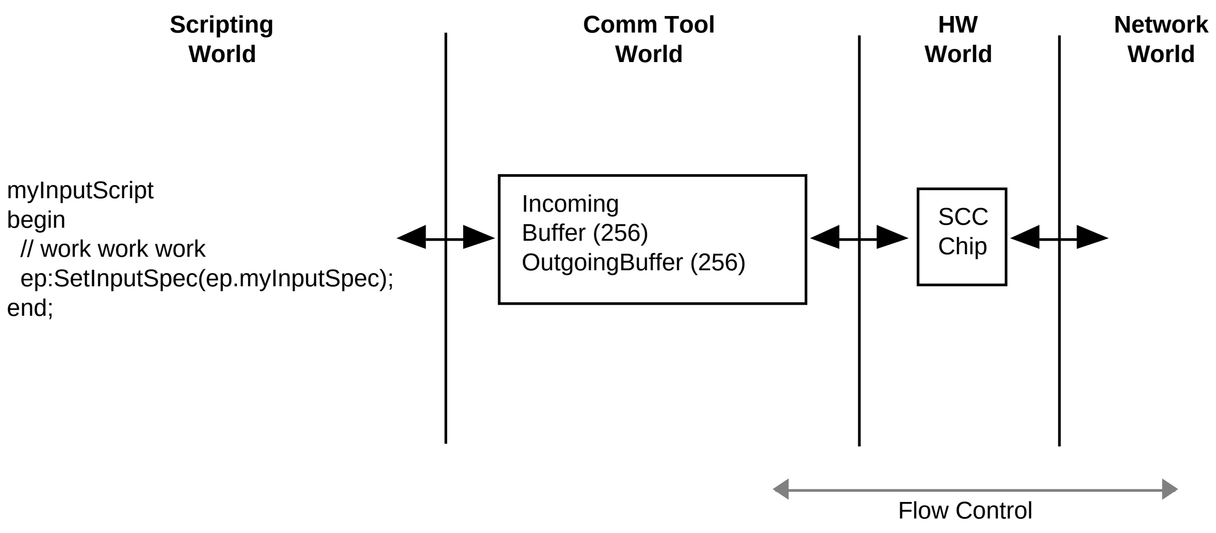
What is Error Code -18003? (3/8/94)

Q: What is error code -18003?

A: This signal is also called SCC buffer overrun; it indicates that the internal serial chip buffer filled, and the NewtonScript part didn't have time to read the incoming information. You need to either introduce software (XON/XOFF) or hardware flow control, or make sure that you empty the buffer periodically.

You will also get -18003 errors if the underlying comms tool encounters parity or frame errors. Note that there's no difference between parity errors, frame errors, or buffer overruns; all these errors are mapped to -18003.

Here's an explanation of what is going on concerning the serial chip, the buffers and the scripting world:



The SCC chip gets incoming data, and stores it in a 3-byte buffer. An underlying interrupt handler purges the SCC buffer and moves it into a special tools buffer. The comms system uses this buffer to scan input for valid end conditions (the conditions which cause your inputSpec to trigger). Note that you don't lose data while you switch inputSpecs; it's always stored in the buffer during the switch.

Now, if there's no flow control (XON/XOFF, HW handshaking, MNP5), the network side will slowly fill the tool buffer, and depending on the speed the buffer is handled from the scripting world sooner or later the comms side will signal a buffer overrun. Even if flow control is enabled, you may still receive errors if the sending side does not react fast enough to the Newton’s plea to stop sending data. In the case of XON/XOFF, if you suspect that one side or the other is not reacting or sending flow control characters correctly, you may want to connect a line analyzer between the Newton and the remote entity to see what is really happening.

If you have inputScripts that take a long time to execute, you might end up with overrun problems. If possible, store the received data away somewhere, quickly terminate the inputSpec, then come back and process the data later. For instance, you could have an idleScript which updates a text view based on data stored in a soup or in a slot by your inputSpec.

Do Not Pass View Frames To OutputFrame (3/16/94)

You can send almost any frame to OutputFrame, as long as it isn't a view. The reason you currently can't pass a view is that it contains both _Parent and _Proto slots. These slots cause OutputFrame to choke as it tries to flatten the entire inheritance chain.

If you are sending frames constructed from views, be careful to remove any references to data (such as _Parent) that you don't wish to send.

One way to do this is to only send Soup entries with OutputFrame. Note that when you put a frame in a Soup, the data storage system strips away the _Proto slot but not the _Parent slot for you.

We are working on ways to extend OutputFrame so that you will be able to send a view directly, without pulling in the entire ROM into the data transfer.

OutputFrame Requires An 8-bit Data Pipe (3/16/94)

In order for OutputFrame to operate correctly, it needs a guaranteed 8-bit binary transfer pipe. Simple flow control (XON/XOFF or hardware) is not sufficient. You must use some form of error control such as MNP.

A Dynamic byteCount inputForm (6/7/94)

Q: I want to use the byteCount inputForm, but I need to set the value of the byteCount slot dynamically. Currently I store this value in a slot of my application's base view. How do I set up the inputSpec to look there, and how do I access my application's base view from within my inputScript?

A: This is less obvious than it may seem, but is actually very easy to do:

The frame given to SetInputSpec should be thought of as "a function with static variables." In order to modify the frame's byteCount slot, the entire frame must exist in RAM at runtime.

Let's assume your inputSpec frame is a template (defined at compile time using NTK). Simply Clone the frame when you are setting up your endpoint. This will duplicate the inputForm and byteCount fields, as well as the reference to your inputScript. Alternately, you could build the frame from within a function at run time.

To access slots of your application base view, the simplest method is to add a `_parent` slot to your endpoint frame (for example, during initialization, before instantiation) which points to your application base view. When your inputScript is triggered, `self` will be the input spec frame, and you will have a reference to the endpoint frame in your first parameter. You may then use syntax such as:

```
byteCount := endpoint._parent.appBaseViewByteCountSlot;
```

or

```
byteCount := endpoint:GetByteCount();
```

If you prefer to use parent inheritence and an accessor method from within your inputScript. Note however that a complete path expression will yield the best performance.

Also be aware that, unless you're using a reliable protocol such as MNP, any error in the input stream can cause the byteCount method of input to fail. It is essential that the data be validated using a checksum or other such mechanism. Where possible, rely on MNP or ADSP for error correction and flow control, especially at high speeds.

Connected endpoints and the powerOffScript (6/7/94)

Q: How should I disconnect my endpoint when my powerOffScript is called? The delayed action used to disconnect and dispose of the connection doesn't execute because the Newton falls alseep.

A: You have essentially three choices:

1. Prevent the Newton from sleeping by calling AddPowerOffHandler before connecting, and RemovePowerOffHandler after disconnecting.
2. Veto the power off request until the endpoint is no longer connected.
3. Disconnect the endpoint in the powerOffScript, and use a deferred action to dispose of the endpoint when the Newton wakes up.

A fourth option may become available in a future release which will allow you to put the Newton to sleep procedurally. This would give you the ability to veto the power off request, confirm with the user that they wish to disconnect, and if so, disconnect, dispose, and request sleep from within the delayed action.

jfs: It is not currently documented, but calling GetRoot():GotoSleep() (preferably from a delayed action???) will do the trick. Anyone know if this function will become public?

OutputFrame fails after Connect (6/7/94)

Q: OutputFrame always seems to throw an exception, and the exception data appears to be garbage. What am I doing wrong?

A: This is the result of a bug in OutputFrame. It will fail if called immediately after the connection is established unless one or more bytes are sent using Output first. A future system software release should not have this restriction.

The good news is that you can actually use this to your advantage! For example, you could send a version string before transmitting the first frame, to let the receiver know the format of what's coming next. Another possibility involves breaking up your frames into smaller "subframes" to reduce the size of the NewtonScript heap required to flatten each frame during transmission and reassembly. Using Output to send the additional control information required to reassemble the subframes between each frame transmission is a powerful and fast approach to complex Newton-to-Newton communications.

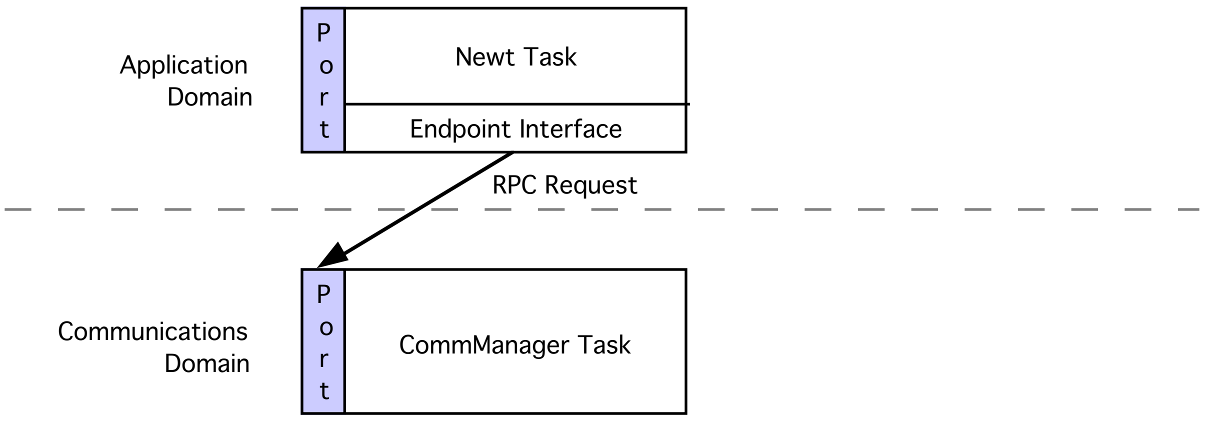
What Really Happens During Instantiate & Connect (6/14/94)

Q: Does Instantiate or Connect touch the hardware? Why can't I open more than one endpoint at a time?

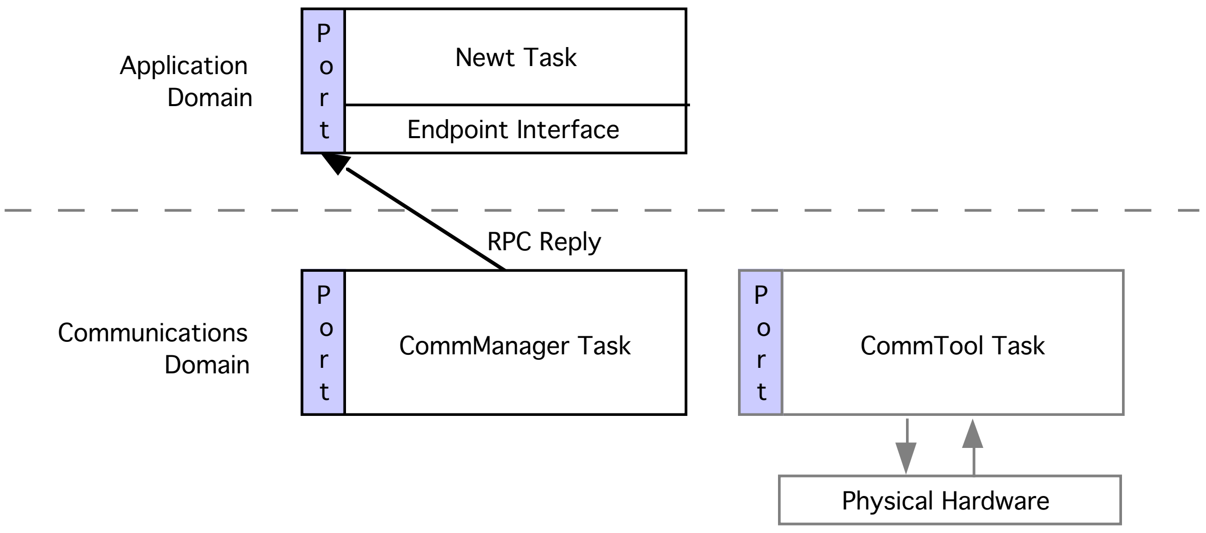
A: Exactly what happens depends on the type of endpoint being used. In general:

The endpoint requests one or more communications services using the configOptions array of the endpoint frame like this:

```
{
  type:     'service,
  label:    kCMSAsyncSerial,
  opCode:   opSetRequired
}
```

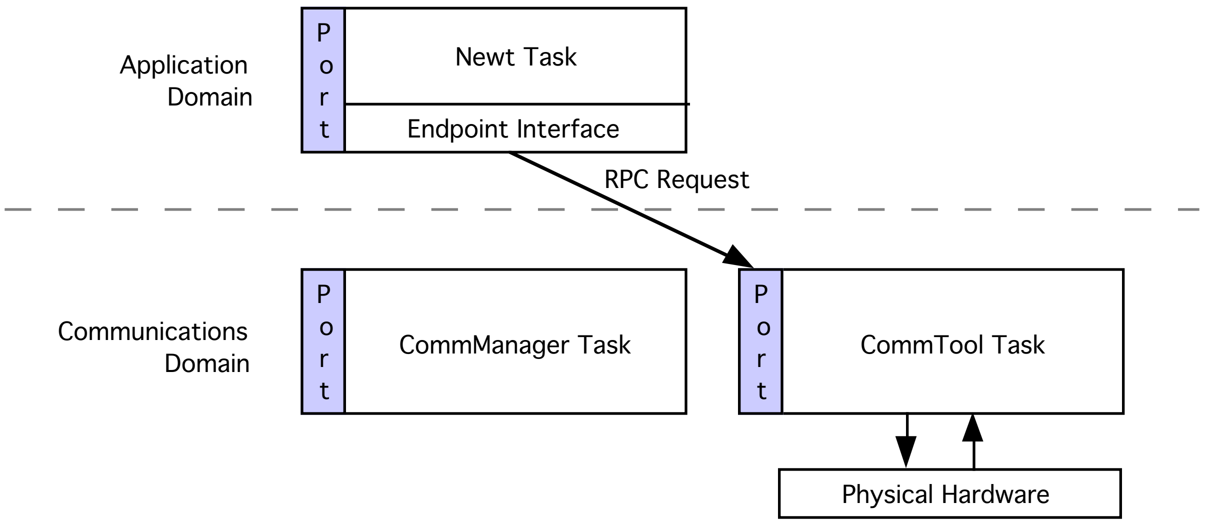


The CommManager task creates the appropriate CommTool task(s) and replies to the communications service request. Each CommTool task initializes itself and acquires access to the physical hardware, such as powering up the device. The endpoint is ready-to-go.

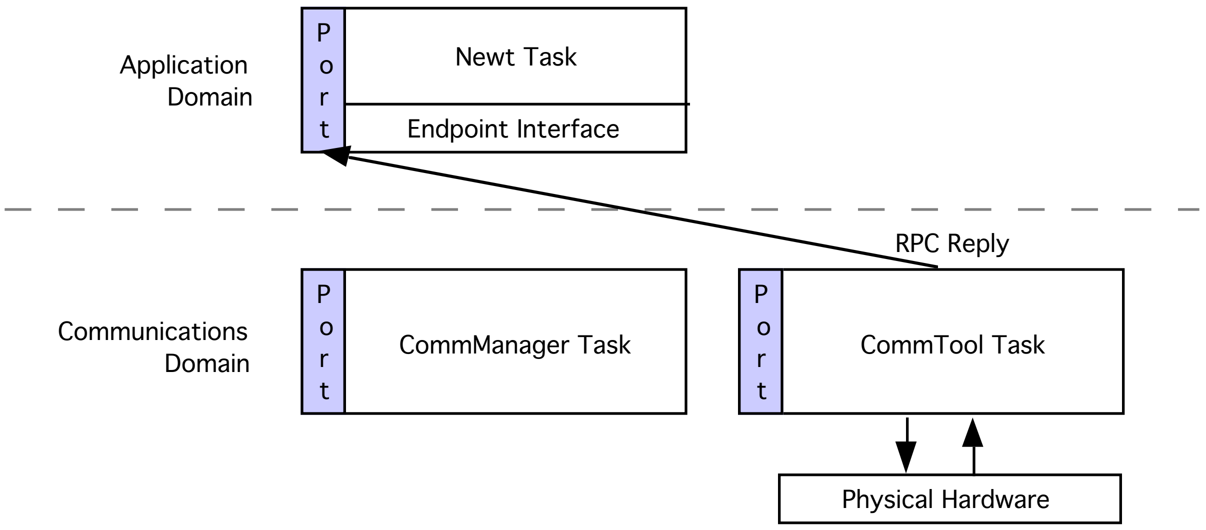


An endpoint may use multiple CommTool tasks, but there will always be a single NewtonScript endpoint reference for them. Multiple endpoints are currently unsupported, but future communications system software will remove that restriction (architecturally this is not a problem).

When the endpoint requests a connection, the CommTool interacts wih the physical hardware as necessary to complete the connection, depending on the type of communications service. For example, ADSP will use the endpoint address frame to perform an NBP lookup and connection request. MNP will negotiate protocol specifications such as compression and error correction.



The CommTool completes the connection and replies to the connection request. Note that if this is done from within a deferred action, the Newt task will continue execution while the deferred function blocks, giving the user an option to abort the connection request.

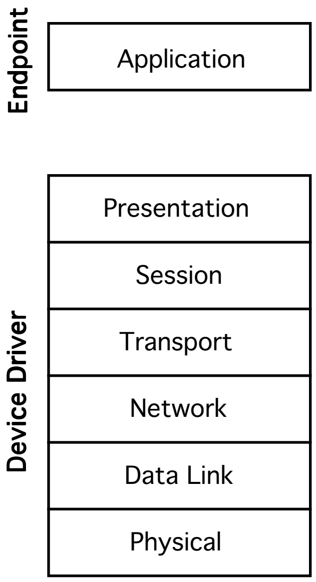


Disconnect functions similarly to Connect, moving the endpoint into a disconnected state. Dispose turns off hardware and deallocates the CommTool task.

Endpoints & The ISO-OSI Reference Model (6/14/94)

Q: How does the Newton endpoint architecture fit into the ISO-OSI standard reference model for communcations?

A: The Newton contains an implementation of AppleTalk which conforms to the standard seven-layer ISO-OSI reference model. Currently ZIP, PAP, and ADSP session protocols, the DDP network protocol, and the LLAP data link layers have been implemented. Future system software may feature session-independent AppleTalk protocols, TCP/IP, and PPP protocols for example.



IR Port Hardware Specs (6/15/94)

Q: What are the hardware specifications for the Newton IR port?

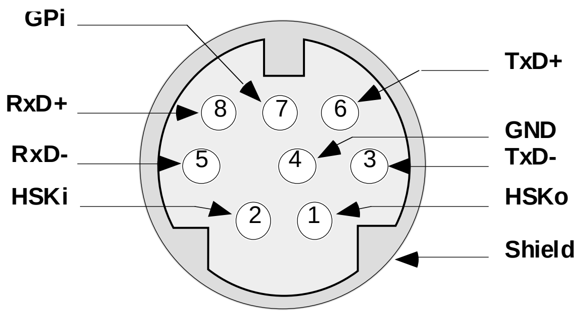
A: In the MessagePad 100 and 110, the IR transmitter/receiver is a Sharp Infrared Data Communication Unit model RY5BD11 connected to channel B of a Zilog 85C30 SCC. Data is communicated along a 500 KHz carrier frequency at 9600 or 19200 baud, 8 data bits, 1 stop bit, odd parity. The IR hardware requires a minimum of 5 milliseconds settling time when transitioning between sending and receiving. Sharp's CE-IR2 wireless interface unit may be used to connect the Newton to Macintosh or DOS machines, with the appropriate software.

The Newton supports three IR software data modes:
Sharp encoding, NewtIR protocol (specifications are NOT releaseable)
Sharp encoding, plain serial (specifications available from Sharp)
38 KHz encoding ("TV Remote Control")

Serial Port Hardware Specs (6/15/94)

Q: What are the hardware specifications for the Newton serial port?

A: The Newton serial port is an EIA standard RS-422 port with the following pinout (as viewed looking at the female Mini-DIN-8 socket on the side of the Newton):



Pin **1**	HSKo	/DTR
Pin **2**	HSKi	/CTS
Pin **3**	TxD-	/TD
Pin **4**	GND	Signal ground connected to both logic and chassis ground.
Pin **5**	RxD-	/RD
Pin **6**	TxD+	(see below)
Pin **7**	GPi	General purpose input received at SCC's DCD pin.
Pin **8**	RxD+	(see below)

All inputs are:	Vih = 0.2V	Vil = -0.2V	Ri = 12k ohms
All outputs are:	Voh = 3.6V	Vol = -3.6V	Rl = 450 ohms

Pins 3 & 6 tri-state when SCC's /RTS is not asserted.

The EIA RS-422 standard modulates its data signal against an inverted (negative) copy of the same signal on another wire (twisted pairs 3/6 & 5/8 above). This differential signal is compatable with older RS-232 standards by converting to EIA standard RS-423, which involves grounding the positive side of the RS-422 receiver, and leaving the positive side of the RS-422 transmitter unconnected. Doing so, however, limits the usable cable distance to approximately 50 feet, and is somewhat less reliable.

Unicode-ASCII Translation Issues (6/16/94)

Q: How are out-of-range translations handled by the endpoints? For example, what happens if I try to output "\u033800AE\u Apple Computer, Inc."?

A: The first Unicode character (0338) is mapped to ASCII character 255 because is it out of the range of valid translations, and the second Unicode character (00AE) is mapped to ASCII character A8 because the Mac character set has a corresponding character equivalent in the upper-bit range.

All out-of-range translations, such as the 0338 diacritical mark above, are converted to ASCII character 255. However, **the reverse is not true**! ASCII character 255 is converted to Unicode character 02C7. This means you will need to escape or strip all 02C7 characters in your strings before sending them if you want to use ASCII character 255 to detect out-of-range translations. Character 255 was picked over character 0 because 0 is often used as the C-string terminator character.

The built-in Newton Unicode-ASCII translation table is set up to handle the full 8-bit character set used by the Macintosh. Although `kMacRomanEncoding` is the default encoding system for strings on most Newtons, you can specify it explicitly by adding one of the following encoding slots to your endpoint:

```
encoding:  kMacRomanEncoding; // Unicode<->Mac translation
encoding:  kWizardEncoding ;     // Unicode<->Sharp Wizard
                        // translation
encoding:  kShiftJISEncoding ;    // Unicode<->Japanese ShiftJIS
                        // translation
```

For `kMacRomanEncoding`, the upper 128 characters of the Macintosh are sparse-mapped to/from their corresponding unicode equivalents. The map table can be found in Appendix B of the NewtonScript Programming Language reference. The upper-bit translation matrix is as follows:

```
short gASCIIToUnicode[128] = {
  0x00C4, 0x00C5, 0x00C7, 0x00C9, 0x00D1, 0x00D6, 0x00DC, 0x00E1,
  0x00E0, 0x00E2, 0x00E4, 0x00E3, 0x00E5, 0x00E7, 0x00E9, 0x00E8,
  0x00EA, 0x00EB, 0x00ED, 0x00EC, 0x00EE, 0x00EF, 0x00F1, 0x00F3,
  0x00F2, 0x00F4, 0x00F6, 0x00F5, 0x00FA, 0x00F9, 0x00FB, 0x00FC,
  0x2020, 0x00B0, 0x00A2, 0x00A3, 0x00A7, 0x2022, 0x00B6, 0x00DF,
  0x00AE, 0x00A9, 0x2122, 0x00B4, 0x00A8, 0x2260, 0x00C6, 0x00D8,
  0x221E, 0x00B1, 0x2264, 0x2265, 0x00A5, 0x00B5, 0x2202, 0x2211,
  0x220F, 0x03C0, 0x222B, 0x00AA, 0x00BA, 0x2126, 0x00E6, 0x00F8,
  0x00BF, 0x00A1, 0x00AC, 0x221A, 0x0192, 0x2248, 0x2206, 0x00AB,
  0x00BB, 0x2026, 0x00A0, 0x00C0, 0x00C3, 0x00D5, 0x0152, 0x0153,
  0x2013, 0x2014, 0x201C, 0x201D, 0x2018, 0x2019, 0x00F7, 0x25CA,
  0x00FF, 0x0178, 0x2044, 0x00A4, 0x2039, 0x203A, 0xFB01, 0xFB02,
  0x2021, 0x00B7, 0x201A, 0x201E, 0x2030, 0x00C2, 0x00CA, 0x00C1,
  0x00CB, 0x00C8, 0x00CD, 0x00CE, 0x00CF, 0x00CC, 0x00D3, 0x00D4,
  0xF7FF, 0x00D2, 0x00DA, 0x00DB, 0x00D9, 0x0131, 0x02C6, 0x02DC,
  0x00AF, 0x02D8, 0x02D9, 0x02DA, 0x00B8, 0x02DD, 0x02DB, 0x02C7
};
```

The ioBusy Flag (7/8/94)

The current 1.x ROM environment has a global flag called ioBusy. This flag is enabled (set to true) when a communications session is going on. Examples of such sessions are when the Inspector window is open, or when the Newton is sending a fax.

Developers could make use of this flag to ensure that they could open a communications session, and that no other application would be concurrently using the communications services. In other words, do not start a communications session unless ioBusy is nil; if you start a communications session, you set ioBusy to true. The current system can run into serious problems if multiple communications sessions are enabled.

Note, however, that this flag will not be supported in the forthcoming Newton releases. You should always check if the ioBusy global exists before you read it or write to it.

Timeouts and ADSP Connections (8/25/94)

Q: We have seen problems with ADSP endpoints when we attempt to connect to an entity that resides in another zone. In many cases the connection fails after a certain time interval. We suspect the problem is timing related, so we want to adjust any possible timeout values associated with an ADSP connection phase.

A: The problem you encounter usually happens in AppleTalk networks with slow routers or other gateways.

Currently there is no way to adjust the ADSP connection timeouts. The workaround is to check for any results from the Connect method, and retry. After N attempts, signal the user that the connection failed, and ask if the user wants to try again or not.

Five States In The Life Of An Endpoint (8/25/94)

Q: I want to call Connect or Listen in a deferred action so that the user can cancel the connection request, but I'm having a lot of problems managing the deferred and delayed actions. It's too easy for the user to crash the Newton. For example, what do I do if the user starts an asynchronous connect, closes the application, opens it again, and tries to connect before my previous delayed action has had a chance to execute and dispose of the previous endpoint?

A: Basically you need a way to manage the flow of control in your application external to your endpoint, or perhaps to the application itself. A very simple solution, which only requires an integer of overhead when your application is closed, is to use a persistent integer slot in your application's base view frame. Its value is never initialized at application startup, but rather it is set by a carefully controlled sequence of deferred and delayed actions during the connect and disconnect process. Of course, it has an initial ROM value of "disconnected" assigned to it by NTK.

The five states you need to manage in the life of an endpoint are:

Disconnected	Endpoint frame may or may not exist. If it exists, it has not yet been instantiated or will be disposed of soon. This is the normal "start-up" state. You may connect only if the endpoint is in this state.
Connect	Endpoint is beginning the connect process, instantiating the endpoint, setting up variables, and so on. The deferred action to perform the connect has not yet been queued.
Connecting	Endpoint is connecting. The deferred action is blocked, waiting for the connect to complete, while the thread that added the deferred action continues execution. You may disconnect the endpoint in this state (same steps as when connected, except don't call the endpoint disconnect function because it's not yet connected).
Connected	If the connect was successul, this becomes the normal operating state of the endpoint for I/O. If the connect was unsuccessful, the disconnect process should begin to recover from the failed connect attempt. You may disconnect the endpoint in this state.
Disconnecting	Endpoint is aborting any pending I/O. A delayed action has been queued to perform the disconnect and dispose.

Refer to the sample code titled **ACME Modem** and **ACME Serial** for more information.

Endpoint I/O Data Buffers & Flow Control (10/5/94)

If the communication dies mysteriously, it sometimes helps to increase the communication tool's incoming buffer size, from the default 256 bytes to 1K. Note that this should be done only if you suspect connection problems related to buffer overruns, otherwise you'll just waste memory. Symptoms of a problem with buffer sizes include sudden connection drops, mysterious hangs in the middle of a large transaction, lost or invalid data, and so on.

Below is a snippet showing how this is done:

```
// setup a 512 byte output buffer and 1024 byte input buffer
local aSerialBuffersParam :=
{
   label:   kCMOSerialBuffers,
   type:    'option,
   opCode:  opSetRequired,
   data:    [ 0, 0, 2, 0, 0, 0, 4, 0, 0, 0, 0, 8 ],
};

// add an option to increase the size of the serial buffer
// when we instantiate the endpoint
if (not instantiated) then
   begin
      AddArraySlot(  modemEndPoint.configOptions,
                     aSerialBuffersParam);
      modemEndPoint:Instantiate( modemEndPoint,
                                 modemEndPoint.configOptions);
      instantiated := true;
   end;
// connect
modemEndPoint:Connect(itsAddress, nil);
```

Note that we will provide this as an option later, in order to ensure forward compability with future Newton platforms. In the meantime, how do we change the buffer size to other values?

The buffer data is actually three long words. The first one defines the send buffer, the second one the receive buffer, and the third one defines the error marker size per message. Always keep the error marker size 8. You'll very seldom need to increase the send buffer, and in most cases you'll only need to modify the receive buffer.

So looking at the example:
`[ 0, 0, 2, 0, 0, 0, 4, 0, 0, 0, 0, 8 ]`

We see that `[0,0,2,0]` defines the send buffer, `[0,0,4,0]` defines the receive buffer, and `[0,0,0,8]` defines the error marker size.

Let's say we want a 4982- byte input buffer. In practice, you should round your input buffer sizes to 256 byte units. This isn't required, and 4982 is used here to aid in the explanation.

First we convert this number to binary:

```
4982 / 2 = 2491 remainder 0 = bit 0
2491 / 2 = 1245 remainder 1 = bit 1
1245 / 2 =  622 remainder 1 = bit 2
 622 / 2 =  311 remainder 0 = bit 3
 311 / 2 =  155 remainder 1 = bit 4
 155 / 2 =   77 remainder 1 = bit 5
  77 / 2 =   38 remainder 1 = bit 6
  38 / 2 =   19 remainder 0 = bit 7
  19 / 2 =    9 remainder 1 = bit 8
   9 / 2 =    4 remainder 1 = bit 9
   4 / 2 =    2 remainder 0 = bit 10
   2 / 2 =    1 remainder 0 = bit 11
   1 / 2 =    0 remainder 1 = bit 12
```

Lay the remainders out flat, with bit 0 as the rightmost least significant bit:
`1001101110110`

Pad it with leading zero bits to form a 32-bit number:
`00000000000000000001001101110110`

Chop it into 4 pairs of nibbles:

```
0000 0000   0000 0000   0001 0011   0111 0110
```

Convert each nibble to its corresponding hex value:

```
0 0   0 0   1 3   7 6
```

Now each nibble pair specifies an element of the array as follows:
`[0x00, 0x00, 0x13, 0x76]`

Which is equivalent to the following decimal description:
`[0, 0, 19, 118]`

And if you're particularly lazy, you could use this handy little function instead:

```
func(x)
begin
   local s := BAND(x, 0xFF);
   for i := 0 to 2 do
      s := BAND(x := x >> 8, 0xFF) & ", " & s;
   "[" & s & "]";
```

```
end;
```

Serial buffer overruns are often related to a lack of error control or flow control. For instance, the modem tool may connect to a remote service with no flow control if your modem endpoint is set up to allow connections without flow control. The workaround in the case of the modem endpoint is not to set this serial buffer parameter, but rather to set XON/XOFF flow control using SetOptions after Connect. In this case you will want to pass the kCMOInputParms and kCMOOutputParms as follows:

```
{  label:   kCMOInputFlowControlParms,
   type:    'option,
   opCode:  opSetRequired,
   data:    {  xonChar:    kDefaultXonChar,
               xoffChar:   kDefaultXoffChar,
               useSoftFlowControl:  TRUE,
               useHardFlowControl:  NIL,  }, }

{  label:   kCMOOutputFlowControlParms,
   type:    'option,
   opCode:  opSetRequired,
   data:    {  xonChar:    kDefaultXonChar,
               xoffChar:   kDefaultXoffChar,
               useSoftFlowControl:  TRUE,
               useHardFlowControl:  NIL,  }, }
```

Note that hardware flow control should be used whenever possible, as it is a much more reliable mechanism than XON/XOFF at high speeds (with the proper cables, of course). Do not set XON/XOFF flow control when using an MNP connection, as the flow control characters will interfere with the MNP data stream. MNP is by definition its own flow control protocol. XON/XOFF may also interfere with binary data transfers, such as OutputFrame, so it's best to rely on MNP or hardware handshake in that case.

Newton Serial IR Endpoint (12/29/94)

Included in the 1.1 platform file are two functions which enable developers to open the IR port as if it were a half-duplex asynchronous serial port. This may help alleviate the need for the SharpIR or NewtIR protocol specifications. However, it is up to the developer to implement the packetizing and error correction necessary for reliable serial IR communication. Their use is as follows:

1. Turn on the IRDCU:

```
call kEnableIRModuleFunc with (true);
```

2. Instantiate the endpoint with an option slot for SCC port B:

```
ep := {
    _proto:         protoEndpoint,
    _parent:        self,   // or whatever you want to inherit from
    configOptions:  [
            {   label:  kCMSAsyncSerial,
                type:   'service,
                opCode: opSetRequired, },

            ... <various other option frames> ...

            {   label:  kCMOSerialHardware,
                type:   'option,
                opCode: opSetRequired,
                data:   call kGetSCCSideBFunc with (), }, ],
    };
local err := ep:Instantiate(ep, nil);
```

Note that kGetSCCSideBFunc can **only** be called at run time! Any attempt to place the above kCMOSerialHardware option frame in an NTK evaluate slot will crash the NTK compiler.

3. Set up your exception handler to deal with -18003 errors. You'll get a lot of them because of parity and bit-frame errors.

```
    if exception.data = -18003 then...
```

4. Do your I/O (remember you can't SetInputSpec and Output simultaneously):

```
    if passive then
      ep:SetInputSpec(ep.myInputSpec);
    else
      ep:SetInputSpec(nil);
    ...
    // wait at least 5ms before outputting
    ep:Output(data, nil);
    ep:FlushOutput();
```

5. Turn off the IRDCU after the endpoint Dispose in your delayed action:

```
    ep:Disconnect();
    ep:Dispose();
    call kEnableIRModuleFunc with (nil);
```

The code which packetizes and error-corrects should transmit 5 or so 0xFF characters before each packet, to give the receiver a chance to synchronize with the sender. One-bits are "clock-only" bits, whereas zero-bits are "double-clock" bits (they have an extra phase transition). When the receiver is trying to phase-lock to the sender's clock, all it really wants is the clock signal.

The low-level SCC driver will actually attempt to receive and buffer the data being transmitted, even if the current input spec is nil. You should, therefore, flush the input buffer after transmission, and before setting the next input spec.

These functions are intended to give developers a head start at implementing their own IR protocol. However, there are some anomalies in the 1.x operating system that you should be aware of when using the IR port as an ordinary serial port:

1. While transmitting data, the SCC receiver remains enabled. This may cause the SCC to destablize its transmissions as a result of the high frequency IR reflections caused by the transmissions. Likewise, the SCC transmitter remains enabled while receiving data, which will cause the receiver to destablize if any data is transmitted during the receive.

2. It is preferable to enable the IRDCU only while transmitting, rather than for the duration of the endpoint, but the asynchronous nature of endpoints makes this way of using the IRDCU more difficult than keeping the unit on at all times.

3. There are a small number of buffer management and exception handling conditions that may cause data to be transmitted and received incorrectly regardless of the above conditions. These issues are not correctable in the 1.x operating system, but will be addressed in a future OS release.

4. Reliable communications are best obtained by implementing a "ping-pong" protocol. In other words, unit A sends a single packet to unit B, then unit B sends a single packet to unit A, and so forth. Neither unit attempts to send or receive more than one packet per turn, or at the same time,

and transmissions errors are detected and corrected during each turn.

Refer to the **ACME Serial** sample code for more information.

Sharp IR Protocol (12/2/94)

(Distilled from source dated 10/14/1992)

1 Serial Chip Settings

Baudrate	9600
Data bits	8
Stop bits	1
Parity	Odd

2 Hardware Restrictions

The IR hardware used in the Sharp Wizard series (as well as Newtons and other devices) require a brief stablizing period when switching from transmitting mode to receiving mode. Specifically, it is not possible to receive data for two milliseconds after transmitting. Therefore, all device should wait three milliseconds after completion of a receive before transmitting.

3 Packet Structure

There are two kinds of Packets: "Packet I" and "Packet II". Because the IR unit is unstable at the start of a data transmission, DUMMY (5 bytes of null code (0x00)) and START ID (0x96) begin both packet types. At least two null bytes must be processed by the receiver as DUMMY before the START ID of a packet is considered. After this (DUMMY, START ID) sequence the PACKET ID is transmitted. Code 0x82 is the packet ID for a PACKET I transmission, and code 0x81 is the packet ID for a PACKET II transmission.

3.1 Packet I

This packet type is used to transmit the following **control** messages:

3.1.1	Request to send	ENQ (0x05)
3.1.2	Clear to send	SYN (0x16)
3.1.3	Completion of receiving data	ACK (0x06)
3.1.4	Failed to receive data	NAK (0x15)
3.1.5	Interruption of receiving data	CAN (0x18)

The format of this packet type is as follows:

	Byte length	Set value in transmission	Detection method in reception
DUMMY	5	0x00 * 5	Only 2 bytes are detected when received.
START ID	1	0x96	
PACKET ID	1	0x82	
DATA	1	above mentioned data	

Packet I example:

DUMMY	0x00, 0x00, 0x00, 0x00
START ID	0x96
PACKET ID	0x82
DATA	0x05

3.2 Packet II

This packet type is used to transmit **data**. The maximum amount of data that may be transmitted in one packet is 512 bytes. If more than 512 bytes is to be transmitted, it is sent as several consecutive 512-byte packets. The last packet need not be padded if it is less than 512 bytes and is distinguished by a BLOCK NO value of 0xFFFF.

The format of this packet type is as follows:

	Byte length	Set value in transmission	Detection method in reception
DUMMY	5	0x00 * 5	Only 2 bytes are detected.
START ID	1	0x96	
PACKET ID	1	0x81	
VERSION	1	0x10	Judge only bit 7-4
BLOCK NO	2 (L/H)	0x0001 ~ 0xFFFF	
CTRL CODE	1	0x01	Don't judge
DEV. CODE	1	0x40	Don't judge
ID CODE	1	0xFE	Don't judge
DLENGTH	2 (L/H)	0x0001 ~ 0x0200	
DATA	1 ~ 512		
CHKSUM	2 (L/H)		

BLOCK NO in last block must be set to "0xFFFF".

CHKSUM is the two-byte sum of all of the data bytes of DATA where any overflow or carry is discarded immediately.

Send all two-byte integers lower byte first and upper byte second.

Packet II example:

DUMMY
START ID
PACKET ID
VERSION
BLOCK
NO
CTRL CODE
0x00, 0x00, 0x00, 0x00
0x96
0x81
0x10
Low
High
0x01

DEV CODE
ID CODE
DLENGTH

data
CHECKSUM

0x40
0xFE
Low
High
????
Low
High

4 Protocol

Data will be divided into several blocks of up to 512 bytes each. These blocks are transmitted using type I and II packets as follows:

4.1 Transmission Protocol

4.1.1 The initiating device (A) begins a session by sending an ENQ (type I) packet. The receiving device (B) will acknowledge the ENQ by transmitting a SYN packet.

4.1.2 When (A) receives a SYN packet, it goes to step 4.1.4 below.

4.1.3 When (A) receives a CAN packet, or when 6 minutes have elapsed without a SYN packet reply to an ENQ packet, (A) terminates the session. If (A) receives any other packet, no packet, or an incomplete packet, it begins sending ENQ packets every 0.5 seconds.

4.1.4 When (A) receives a SYN packet, it transmits a single type II data packet, then awaits an ACK packet from (B).

4.1.5 When (A) receives an ACK packet, the transmission is considered successful.

4.1.6 If no ACK packet is received within 1 second from completion of step 4.1.4, or if any other packet is received, (A) goes to step 4.1.1 and transmits the data again. Retransmission is attempted once. The session is terminated if the second transmission is unsuccessful.

4.2 Reception Protocol

4.2.1 The receiving device (B) begins a session by waiting for an ENQ (type I) packet. If no ENQ packet is received after 6 minutes (B) terminates the session.

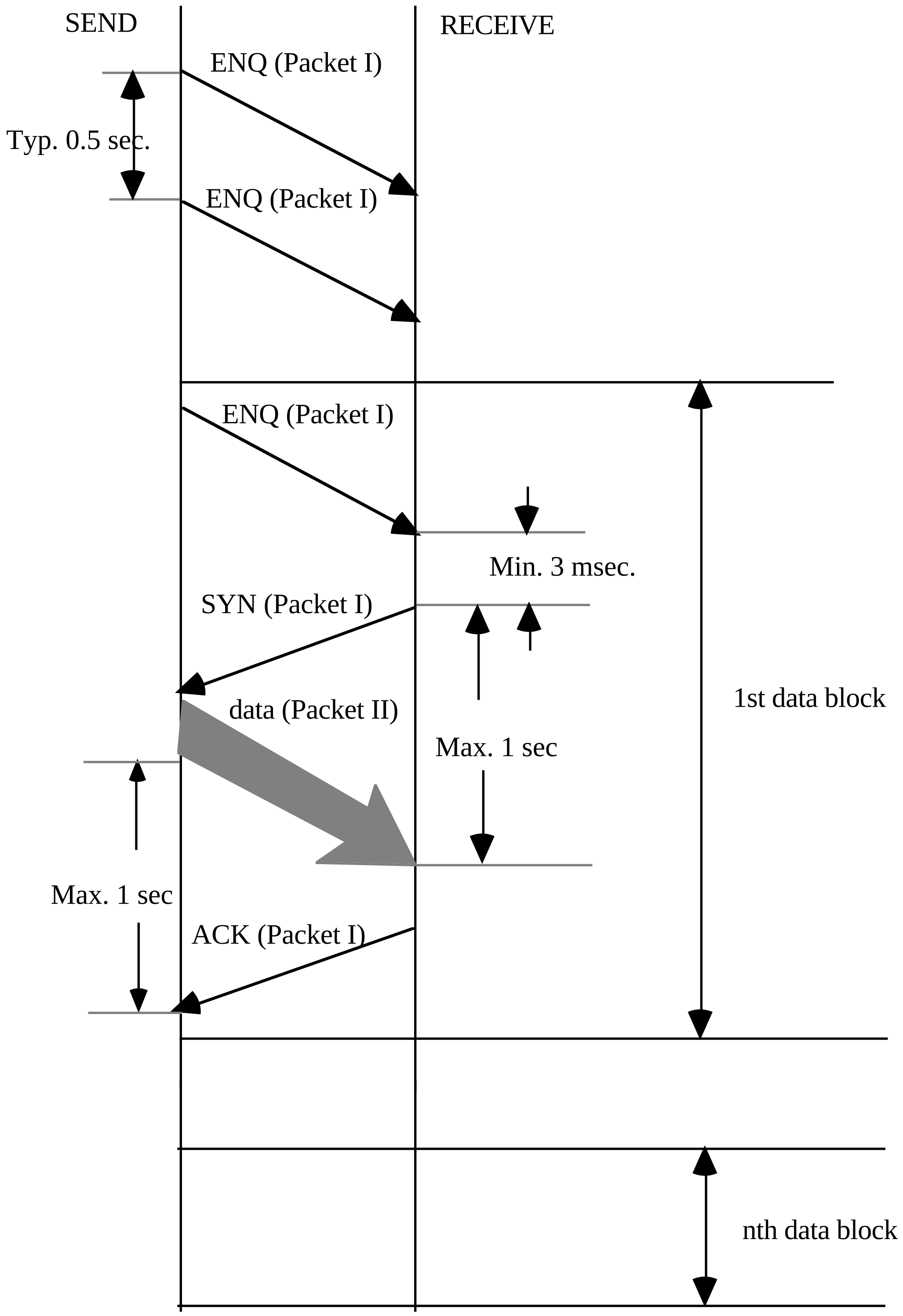
4.2.2 When (B) receives an ENQ packet, (B) transmits either a SYN packet to continue the session or a CAN packet to terminate the session.

4.2.3 When (B) receives a valid type II packet (eg. the checksum and all header fields appear to be correct), (B) transmits an ACK packet.

4.2.4 If one or more header fields of the data packet are not correct, or if the time between data bytes is more than 1 second, (B) goes to step 4.2.1 and does not transmit the ACK packet (this will cause (A) to retransmit the packet after a one second delay).

4.2.5 If the header fields of the data packet appear to be correct but the checksum is incorrect, (B) transmits a NAK packet (this will cause (A) to retransmit the packet immediately).

Because of the restriction in hardware mentioned in item 2 above, it is not possible to receive data for two milliseconds after a data transmission. Please wait three milliseconds before transmitting a response to the other device.



Why is waitForCarrier ignored? (1/10/95)

In the configOptions for a modem endPoint, one of the options that can be set is: kCMOModemDialing. The correct way of using this option is:

```
{  label:   kCMOModemDialing,
   type:    'option,
   opCode:  opSetRequired,
   data:    {  speakerOn:        TRUE, // or NIL
               waitForCarrier:   55,   // or a time in seconds
            },
}
```

Unfortunately, the speakerOn and waitForCarrier flags are ignored. This is a bug in the underlying code which interprets the options (instead of looking in the data frame, the interpreter looks in the option frame). This means that the kCMOModemDialing option must be specified as:

```
{  label:   kCMOModemDialing,
   type:    'option,
   opCode:  opSetRequired,
   data:    {  speakerOn:        TRUE, // or NIL
               waitForCarrier:   55,   // or a time in seconds
            },
   speakerOn:        TRUE,
   waitForCarrier:   55,
}
```

Note that the data frame is kept to ensure that this frame will work correctly when the bug is fixed.

IR Protocol Speeds (1/16/95)

Q: Data sheets indicate Newton IR communications occur at speeds up to 38.4 kbps. Communications documentation implies that 19.2 kbps is the top speed. Which is correct?

A: IR communications occur at 9600 baud when using the Sharp IR protocol (the kCMSSlowIR). When two Newtons are beaming to each other, a slightly faster protocol is used, which communicates at 19.2K baud. PIE engineering reserves the right to modify this protocol (including transmission speed) in current and future products and versions of the Newton operating system. This protocol is intended only for internal Newton communications. Developers should use the kCMSSlowIR endpoint in their applications.

The 38.4 kbps speed (actually 38.4 kHz) refers to a mode of communication in which the Newton's serial communcations controller chip is not involved. This system is used by "TV remote controls" and is a send-only mechanism. Data is "bit-blasted" as a 38.4 kHz square-wave through the IR port by a software loop, bypassing the SCC entirely. Besides being a very inefficient method of data transmission, this mechanism has no provision for error detection/correction. The data to be transmitted must be described as a large binary object of square-wave state transitions and is therefore specific to each vendor's remote control protocol.

FlushOutput and the nested event loop problem (1/16/95)

Q: I've noticed that sometimes my inputScripts are triggered during a call to FlushOutput. If I try to abort or disconnect the endpoint from within that inputScript the Newton hangs. Why is this, and how can I avoid it?

A: This is currently a most subtle and heinous design problem with the Newton's communication architecture. The reader may wish to refer to the Q/A Communications article "What Really Happens During Instantiate & Connect " to better understand the following answer.

FlushOutput suspends the current NewtonScript function until all data in the output buffer has been transmitted. Because data to be output is sent to the communications task via RPC calls, the RPC reply from that task is used to signal the completion of the output. In order to receive an RPC reply, control must be given to the main NewtonScript event loop. Herein lies the problem.

To halt the current NewtonScript function and wait for an RPC reply, FlushOutput enters a "nested event loop" with a filter on the event type. This causes only events of a certain type to be received by the event dispatcher. When the RPC reply is received from the communications task, the filter is removed and the nested event loop is popped, thus returning control to the NewtonScript statement following FlushOutput.

Unfortunately, the RPC reply which indicates the data output has completed is (nearly) identical to the RPC event which indicates data has been received. The main event dispatcher is unable to distinguish the "output completed" event from the "input completed" event and therefore applies the input data to the inputSpec, potentially triggering your inputScript if the terminating conditions of the inputSpec are met.

If the inputScript is triggered in this way, care must be taken to ensure the state of the endpoint is unchanged until the nested event loop has been exited and the FlushOutput call has completed. Otherwise, the Newton will hang (in the nested loop). Calling Abort, Release, Disconnect, or Dispose are therefore forbidden.

There are two basic solutions to this problem, neither of which is ideal.

The first (and preferred) method is to set a flag in your endpoint just before calling FlushOutput and then to clear the flag when FlushOutput has returned. In your inputScripts and endpoint exception handlers, check the state of this flag. If it is set, defer the behavior until the flag has cleared. The method for deferring the action depends on your code and on the state machine you have created to handle your communications; that is, you may be able to simply use AddDeferredActions or a viewIdleScript as appropriate.

The second method is to simply avoid the use of FlushOutput altogether, and avoid the use of an active inputSpec while outputting data. This will slow your communications somewhat by forcing you to adopt a half-duplex system of communications. This solution still requires that you defer any actions which may change the state of the endpoint until your endpoint exception handler has returned.

A future version of the Newton communications system should behave differently.

NEW: Using Serial PCMCIA Cards (04/26/95)

Q: Can I use a Serial PCMCIA card in a Newton?

A: If the Newton indicates a communications card has been inserted, then yes. But the trick is to use the Modem comm tool in a special way which bypasses the "AT" commands and the modem enabler if it's installed. There are four basic steps:

First, create a basic no-frills modem configuration.

```
[   {   label:   kCMSModemID,
        type:    'service,
        opCode:  opSetRequired },

    {   label:   kCMOModemECType,
        type:    'option,
        opCode:  opSetNegotiate,
        data:    kModemECProtocolNone },

    {   label:   kCMOMNPAllocate,
        type:    'option,
        opCode:  opSetRequired,
        data:    kMNPDontAllocate },

    {  label:   kCMOMNPCompression,
        type:    'option,
        opCode:  opSetNegotiate,
        data:    kMNPCompressionNone }, ]
```

Second, append a modem preference option. This will put the modem comm tool into a state which avoids the "AT" commands.

```
   {  label:   kCMOModemPrefs,
      type:       'option,
      opCode:  opSetRequired,
      data: [  1, 0, 0, 0,
                  0, 0, 0, 0,
                  1, 0, 0, 0,
                  0, 0, 75, 0,
                  0, 0, 0, 3,
                  0, 0, 0, 5 ], }
```

Third, append a modem profile option. This will prevent the modem enabler from trying to take control of the endpoint during Instantiate (it overrides whatever profile the user has selected in Modem Prefs).

```
   {  label:   kCMOModemProfile,
      type:       'option,
      opCode:  opSetRequired,
      data: [  0, 0, 0, 1,
                  0, 0, 63, 255,
                  0, 0, 75, 0,
                  0, 0, 3, 232,
                  0, 0, 0, 40,
                  0, 0, 0, 25,
                  0, 0, 0, 6,
                  0, 0, 0, 0, 0, 0 ], },
```

Now Instantiate the endpoint. You can either put all of the above options into the configOptions array of the endpoint, or, pass them in the second parameter to Instantiate.

Fourth, set any serial comms parameters after a successful endpoint Connect. This is necessary because the modem comm tool always sets up the serial port as though it were going to send "AT" commands.

```
   DefConst('kMySerialOptions,
               [  {  label:   kCMOSerialIOParms,
                        type:    'option,
                        opCode:  opSetNegotiate,
                        data: {  bps:          k9600bps,
                                 dataBits:     k8DataBits,
                                 parity:       kNoParity,
                                 stopBits:     k1StopBits, }, },

                  {  label:   kCMOInputFlowControlParms,
                        type:    'option,
                        opCode:  opSetNegotiate,
                        data: {  useHardFlowControl:  nil,
                                 useSoftFlowControl:  true,
                                 xOnChar:     unicodeDC1,
                                 xOffChar:    unicodeDC3, }, },

                  {  label:   kCMOOutputFlowControlParms,
                        type:    'option,
                        opCode:  opSetNegotiate,
                        data: {  useHardFlowControl:  nil,
                                 useSoftFlowControl:  true,
                                 xOnChar:    unicodeDC1,
                                 xOffChar:   unicodeDC3, }, }, ] );

    func(ep)
    ep:SetOptions(kMySerialOptions);
```

There are two minor side-effects to be aware of when using the modem tool in this way: A single newline character will be transmitted when the connection is established, and more system heap memory is used by the modem comm tool than by the async serial tool. A better method of using Serial PCMCIA cards will exist in a future implementation of the Newton communications architecture. When that happens, this method will no longer be recommended and may not be supported.

End of DTSQA

NEWTON CONNECTION KIT

TABLE OF CONTENTS:

Introduction..1
Display PICTs in Newton Connection Kit (11/11/93)..............................1
PC 9-pin D-Type Connectors (11/11/93)...1
Synchronization of Soups and UnionSoups (11/11/93) (Obsoleted by 1.0 Newton Programmer's Guide p16-51)..1
Soup Info, Synchronization (12/19/93)...2
Where to put MetaData (1/4/94) (Obsoleted by 1.0 Newton Programmer's Guide p16-2)..2
Getting Symbols From Text (3/9/94)..2
StringToTime problems in 1.0 (3/9/94)...3
Limitations, Error Messages, Memory Constraints (6/7/94).................3
exportSoup slot in ExportFilter() (6/8/94)..4
GetStores function in Connection Kit (6/8/94)..4
Adding custom translators to Connection Kit 2.0 (6/9/94)..................4
"New Paragraph" button for my soups? (6/9/94)...................................5
Preventing New Entry in Connection? (6/9/94)...5
Connection caches MetaData frames. (8/26/94)..5
INDEXES in Native File Format do not work. (8/26/94)..........................6
Dangers of UniqueID and Connection Restore. (8/25/94)....................6
Creating _exportpict slots in native export (9/16/94)...........................6
No hook into browser deletion (9/28/94)..7
Limit on number of items for tab export. (10/13/94)...........................7
validType
note formatProto slot incorrect (11/18/94)....................................7
NEW
soupInfo is not synched (3/14/95)...8

Introduction

This document addresses Newton Connection Kit 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.

Display PICTs in Newton Connection Kit (11/11/93)

Q: How could I display PICT resources and digital ink in Newton Connection Kit? What about polygon data?

A: Connection Kit 2.0 does not support PICT information or ink in its browsers. Stay tuned for more information concerning future Connection Kit products.

Concerning polygon data, the best approach just now is to use the scripting function to render the polygon data as a series of points, and then export these as X,Y coordinate pairs. These can be read and possibly decoded later by other Macintosh applications.

The Native file export format for Connection Kit 2.0 includes clPolygonView data rendered as PICT data.

PC 9-pin D-Type Connectors (11/11/93)

Q: I would like to connect a Dell PC, but I need information about the serial cable specification (8-pin mini-din connector).

A: Here's the information:

```
MD8M               DB9F
 1                 1,6
 2,7               4
 3                 2
 4,8               5
 5                 3
 6                 NC
```

The MD8M is the Newton male 8-pin (same as Mac serial 8-pin). The DB9F is the PC 9-pin female.

Synchronization of Soups and UnionSoups (11/11/93) (Obsoleted by 1.0 Newton Programmer's Guide p16-51)

Q: It seems that Newton Connection assumes that it will always synchronize union soups. Even though there's only a single instance of the Directory soup on the internal store, Newton Connection shows an instance of all soups with MetaData in this directory for a card in the MessagePad, even if we have not created a soup on this card.

Is there a way to prevent this behavior; that is, to indicate that a soup is not a union soup and that it exists only on a card or in the internal store?

A: Newton Connection indeed assumes union soups, and when you synch, it makes efforts to ensure that a soup is created on each store. This happens with all soups, not just the ones with MetaData.

The idea was that this would allow users to enter data into cards and internal stores with ease.

MetaData written for Connection Kit 2.0 can disable this behavior, preventing Newton Connection from creating a given soup where none existed previously.

Soup Info, Synchronization (12/19/93)

As the soup information frame (specified by the SetInfo call) is unconditionally synchronized with Newton Connection, you should avoid putting large amounts of data in the soup info frame. The data in this frame will get copied over the connection every time a synchronization is performed, which can hurt performance.

The soup information frame is only sent back to the Newton during a restore, so you should not write MetaData function which modify this data.

Where to put MetaData (1/4/94) (Obsoleted by 1.0 Newton Programmer's Guide p16-2)

Q: Where do I put my MetaData to get Newton Connection to use it?

A: MetaDatas go in the MetaData soup, also called the Directory soup. This soup only exists on the internal store. To add your MetaData, do something like this:

```
metaSoup := GetStores()[0]:GetSoup(ROM_MetaSoupName);
metaSoup:Add(myMetaData);
```

Care should be taken to ensure that only one copy of your MetaData record exists in the MetaData soup, and that that record is removed if/when your soup is deleted.

Getting Symbols From Text (3/9/94)

Q: I use MakeSymbol to get the folder symbol in my Import script but it appears to break in 2.0?

A: Connection 2.0 does not support MakeSymbol (in fact, MakeSymbol is not supported in the ROM). You should use Intern. However, Connection 1.0 did not support Intern (it supported MakeSymbol.)

Since Connection 2.0 is a free upgrade, compatibility with the 1.0 version should not be required. However, if it is, you need to define a local function in your MetaData specification that looks like:

```
GetSymbol: func(string)
      if functions.Intern then
            Intern(string)
      else
            MakeSymbol(String),
```

Use your new function `:GetSymbol(string)` where you would have used either MakeSymbol or Intern.

StringToTime problems in 1.0 (3/9/94)

`StringToTime` was broken in 1.0; it returned a frame instead of an integer. In Connection 2.0, this has been corrected.

Since Connection 2.0 is a free upgrade, compatibility with the 1.0 version should not be required. However, if it is, you can define a local function in your MetaData specification that looks like:

```
GetTime: func(string)
            if ClassOf(StringToTime("11:00 am")) = 'frame then
            StringToDate(string)
      else
            StringToTime(string),
```

Use `:GetTime(string)` where you would have used `StringToTime`.

Limitations, Error Messages, Memory Constraints (6/7/94)

Q: There appears to be some restriction on the number of items and/or the size of items that a soup entry can contain, in the Connection Kit.

I built an application that uses the Connection Kit and it worked pretty well at first. When I added a few more items to the soup, and to the MetaData, I got an error message from the Connection Kit. When I tried to open the soup, it said, 'Could not create a new document because there is not enough memory' ! I increased the multifinder partition for the Connection Kit to 2500K. I reduced the number of items in the MetaData back down toward the number I had when my application first worked, and this helped.

A: Newton Connection 1.0 might (as you may have guessed) be displaying the out-of-memory error in response to any throw that happens from inside your export script.

I'd recommend testing your MetaData scripts on the Newton before trying them out in Newton Connection. Take a look at the Sync Another One sample for an illustration of how to do this. Note that the environment on the Newton isn't quite like the environment on the Mac inside Newton Connection, but it's close.

Newton Connection 2.0 has much better control over import/export, as well as better error reporting. Newton Connection 2.0 for Developers will be available shortly; this has an inspector window available for even more detailed debugging of MetaData.

exportSoup slot in ExportFilter() (6/8/94)

Q: I can't make `ExportFilter` cause a different soup to be exported.

A: The `exportSoup` slot returned in the `exportFrame` must contain an actual reference to a soup, not just a string with the name of the soup. Further, you cannot return a reference to a union soup. A different soup on the same store that the entry being exported is on can be gotten with:

```
EntryStore(exportFrame.exportEntry):GetSoup("TheSoupName");
```

GetStores function in Connection Kit (6/8/94)

Q: What does the `GetStores` global function return when called from a MetaData function running in Newton Connection?

A: `GetStores` in Connection Kit returns an array with one element for each synch file that is currently open in Newton Connection. The order of the elements in this array is dependent on the order that the sync files were opened, and elements are removed from the array when the synch file is closed.

Normally when you synchronize a Newton with a card installed, two synch files will be opened, one for the Newton internal store and one for the card. However, the user can open just the sync file for a card, or open that file first. In that case, `GetStores()[0]` will NOT represent the Newton internal store, but will be the card's synced store.

Scripts for MetaData should be written to avoid calling `GetStores`. Instead, use the store argument passed to some of the MetaData functions, or use the `EntryStore` global on a soup entry to determine which store is being operated on.

Adding custom translators to Connection Kit 2.0 (6/9/94)

Q: How do I add a translator to convert my soup to Excel (Word, FileMaker, etc) format?

A: DataVis provides the translators shipped with Newton Connection 2.0, and can be contacted for more information about extending the translator services. Contact Dick Fontana of DataVis Inc., 55 Corporate Drive, Trumbill, CT, USA, (203) 268-0030.

In the (near) future, documentation on how to write translators that integrate with Connection Kit will be available from Apple. Stay tuned.

"New Paragraph" button for my soups? (6/9/94)

Q: When editing the Notepad in NCK, there's a button and menu item for creating new paragraphs. How can I use these in my own metadata?

A: You can't. The Notepad and part of the Calendar soup browsers for Newton Connection are custom-code-built into connection. They are not run solely via MetaData. The card file is the only built-in application that uses only MetaData functionality.

Newton Connection provides no hooks into the functions used to run these browsers. Further, there are no scripting hooks that can be used while the user is editing the text frame produced with an export function.

Preventing New Entry in Connection? (6/9/94)

Q: I don't want the user to be able to add records to the soup in the Connection Kit, but I want them to be able to edit the existing entries. How can I do this?

A: You cannot. You might consider setting `newEntry` to `NIL`, or some similar trick. The problem is that when the user clicks on the New Entry button, Connection Kit clones whatever is in the newEntry slot in the MetaData, then adds that frame to the soup before calling any scripts in MetaData.

Your best approach would be to set the `newEntry` to something that is easy to find and delete later, and have your `editExport` and/or `editImport` warn the user that they cannot add new entries to this soup.

Connection caches MetaData frames. (8/26/94)

Q: My soup design is such that when a `TabImport` is done I need to create a new soup. I create MetaData for the new soup, then use the `metaDataSoupRef` slot in the frame returned from `TabImport` to add new entries to the new soup. However, the `ApplyCommands` method in the newly created MetaData for the new soup never gets called.

A: Connection maintains a cache of MetaData frames for each synch file, and loads that cache only when the synch file is first opened. Because of this Connection won't "see" the new entry in the MetaData soup until the synch file is closed and re-opened.

Instead of using `metaDataSoupRef`, you could explicitly add the entry to the new soup in the main soup's `EntryValidation`, `ApplyCommands` or `TabImport` methods.

INDEXES in Native File Format do not work. (8/26/94)

Q: We want to create an index on a soup that results from importing from a Native File format import. How do we get the INDEXES part of the Native File specification to work? (p16-93/16-93 of the 1.0 Newton Programmer's Guide)

A: There is a bug in the 2.0 Newton Connection Kit which causes indexes described in the INDEXES part of the Native File format to be skipped. A suggested workaround is to detect the missing index on the Newton and send the `AddIndex` message to the soup from your application.

Dangers of UniqueID and Connection Restore. (8/25/94)

Q: When I restore a Newton from a Connection synch file, all the uniqueIDs for my soup entries are different. How can I create cross-entry links?

A: Unfortunately, a bug in the Connection Kit (including the current 2.0.2 version) causes uniqueIDs for soups to change after a restore. This will eventually be fixed, but in the meantime, you cannot reliably use uniqueID to "link" two entries together. Since Connection Restore recovery process is used only rarely, we recommend providing some means to "reconnect" the soup entries through potentially slower means, and advising users that after a restore this relinking process will be required.

PCMCIA backup/restore does not exhibit this problem.

Creating _exportpict slots in native export (9/16/94)

Q: I have an array of points that I want to export as a useful picture on a Mac or PC. How do I do that?

A: You can use the Native Export feature of Connection to export the points as a PICT (Mac) or

Metafile (Windows). You need to create an entry in your exported soup that has the following slots:

For Polygons:
```
{viewBounds: <viewbounds of the shape>,
 viewStationery: 'poly,
 points: <a valid points array binary object>}
```

For Ink:
```
{viewBounds: <viewbounds of the ink>,
 ink: <a valid ink binary object>}
```

When Connection exports a frame that meets the above conditions, it will create a slot called _exportpict as documented in the Newton Programmers Guide 16-87.

No hook into browser deletion (9/28/94)

Q: Is there a function that is called when the user deletes an item from the Browser?

A: Unfortunately, Connection 2.0 has no such function. We will add such a function in later versions, but currently there is no workaround.

Limit on number of items for tab export. (10/13/94)

Q: When I try to do a tab export of a soup entry with very many slots, why don't some show up?

A: When defining a tab export configuration, NCK uses an array to hold the selection and order of slots to export. Because of limits in the software, no more than 128 entries in the array are used to export the data, so no more than 128 fields can be output via tab export.

validType: 'note formatProto slot incorrect (11/18/94)

Q: I use a modified version of the Names metadata from the Built-In Metadatas sample. In some circumstances, changing one of the entries in connection causes an error. The Inspector indicates the problem is in `functions.VerifyParagraphScript`. This is not part of my code; what is the problem?

A: It turns out that the documentation in NPG 1.0 (16-71) on `formatProto` is incorrect. The actual slot should be called `paraFormat`. It **must** be a frame and can contain two slots:
`maxRight`: an integer. the maximum width of the paragraph
`defaultFontSpec`: a font specification (integer or frame) for a default font

So the entry for the notes slot in the `namesValidation` frame should be:

```
notes: {validType: 'note, paraFormat: {maxright: 237}, nilOK: 'delete}
```

Note, you **must** include a `paraFormat` slot if your `validType` is of type `'note`.

NEW: soupInfo is not synched (3/14/95)

Q: I am trying to use a slot in the soup's info frame to assign my own unique id to items. I have the metadata update the soupInfo on the desktop, but the new value never makes it to the Newton.

A: The info frame is not really an 'entry' in the soup. Connection does not synchronize the soup info, it is instead copied from the Newton to the desktop on each synchronize. If you need to do things like counters or other things that are synchronized, we recommend you use an entry in the system preferences soup.

End of DTSQA

DATA STORAGE AND RETRIEVAL

TABLE OF CONTENTS:

DATA STORAGE AND RETRIEVAL..........1
Introduction..........1
Using GotoKey (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-51)..........1
Hand-Rolling an Entry (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-50) 1
EntryChange, EntryUndoChanges, and Autosaving (9/15/93)(Obsoleted by 1.0 Newton Programmer's Guide p7-59)..........1
Cursors are fully dynamic! (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-35) 2
RemoveAllEntries Problems (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-44)..........2
When to Remove Unused Soups? (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-31)..........2
Creating Soups Properly (10/5/93) (obsoleted by 1.0 Newton Programmer's Guide p7-24) 3
Count Soup Entries, How? (10/14/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-48) 3
Getting/Setting the Default Store (11/10/93) (obsoleted by 1.0 Newton Programmer's Guide p7-65)..........4
Null Union-Soups? (11/10/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-66) 4
UnionSoup Dangers (11/10/93)..........4
UniqueIDs (11/10/93)..........4
Soup Removal Problems, Indexing, Error -48022 (11/10/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-32)..........4
Soup Sizes - Generate Test Soup (11/30/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-32)..........5
Be Careful What you Store in Soup (12/6/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-23)..........5
If EntryChange fails, what happens? (12/9/93) 6
Performance
Cursors and EntryRemoveFromSoup (12/11/93)..........6
Always use Constants instead of Strings for Built-in Soups (12/12/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-23)..........7

Do Not Clone Cursors with Clone/TotalClone/DeepClone (12/19/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-50)....7
What Store is Removed? (12/19/93)....7
Store
GetInfo, SetInfo, GetAllInfo Functions (12/19/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-12)....7
Store
SetInfo and Soup
SetInfo (3/9/94)....8
Magic Pointers in soups (3/10/94)....8
Application Preferences and the internal store (1/26/94) (Obsoleted by 1.0 Newton Programmer's Guide p7-33)....8
Correct name of _uniqueID slot (NEW 5/31/94)....9
Nil Index Slot, Adding Entries (6/7/94)....9
Indexes in decreasing order? (6/7/94)....9
startKey and endTest and cursor limits (6/9/94)....10
Interraction of EntryChange and MapCursor (6/9/94)....10
Use GC with RemoveIndex (6/10/94)....10
Beware of Indexes inExternal Soups in Combination with RemoveIndex and AddIndex (6/10/94)....11
AddToDefaultStore catches exceptions (7/5/94)....11
FrameDirty is deep, but can still be fooled. (8/19/94)....12
Dangers of UniqueID and Connection Restore. (8/25/94)....12
Single Unicode Characters and Soups/NTK (10/11/94)....13
store
UsedSize and card 'applet' memory size varies (10/13/94)....13
Potential Problems with Soup Entry _proto Slots (11/10/94)....13
AddWithUniqueID documentation bug (12/7/94)....14
CHANGED
Limits on sizes of soup entries (3/7/95)....14

Introduction

This document addresses Newton Data Storage and Retrieval 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.

Using GotoKey (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-51)

Q: How does `cursor:GotoKey(key)` work?

A: This only works with cursors generated with a query of type index, and you can only supply a key from the slot against which the cursor was generated. For example, if you generate a cursor based on an index in a slot called `name`, and supply a key that is a number in the slot called `extension`, the `GotoKey` function will not work with that key, and an exception will be thrown, because the cursor is on a string key and a number was provided.

Hand-Rolling an Entry (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-50)

Q: Is there a way to us a non-entry frame as the paramater for `cursor:Goto(entry)`?

A: No, there is no way to "hand roll" an entry by creating a frame with the appropriate attributes. Only entries that come from soups are valid. For example, you could extract the entry from a cursor by assigning a variable to the result of cursor:entry and passing that variable as input to `cursor:Goto`.

EntryChange, EntryUndoChanges, and Autosaving (9/15/93)(Obsoleted by 1.0 Newton Programmer's Guide p7-59)

Q: EntryChange(entry) and EntryUndoChanges(entry) are at present unclear. Does the latter exist/work or not? Is there autosaving as in HyperCard ? If you don't callEntryChange(entry), and something major happens, are the changes lost?

A: EntryUndoChanges exists and works as documented. This is the preferred way to perform an undo operation on soup data.

There is no autosaving as with Hypercard. You are responsible for updating data that has been modified in soups.

Where "something major" is defined as catastrophic power loss, program error, hitting the reset button, and so on, data that was changed in an entry and not passed to EntryChange will be lost.

Because the entry is cached and shared by all cursors, other parts of an application may see a "changed" entry even though EntryChange has not yet been called.

Cursors are fully dynamic! (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-35)

Q: Are the results of a query that returns a cursor dynamic or static? Consider this scenario: I have a cursor that points to a list of entries that meet the criteria specified in my query. In the course of my work, I add more entries to the soup against which I made the original query, some of which meet the criteria of my original query. If I call cursor:next enough times will I eventually point to some of my new entries? Conversely, if I modify an entry in my cursor's "list" such that it no longer matches the query criteria, will the cursor no longer point to it?

A: The result pointed to by the cursor is completely dynamic!

If you delete entries which previously met the criteria in the Query call, the cursor will skip over them in the course of traversing the soup entries. In other words, it never points to entries that don't meet the criteria of the original query.

Similarly, if you subsequently add to the soup entries meeting the criteria of the original Query, the cursor will point to them as it traverses the entries in the soup.

As a peripheral note, if you delete an entry while the cursor is pointing to it, then cursor:Entry will return the symbol 'deleted.

RemoveAllEntries Problems (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-44)

Q: `RemoveAllEntries` doesn't seem to exist, at least for union soups—it works on independent soups. What's the problem?

A: No problem. `RemoveAllEntries` is not implemented for union soups, only for "plain" soups.

When to Remove Unused Soups? (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-31)

Q: I have an application that creates and uses a rather specialized soup. This soup will be of no use to any other applications, so when a user decides to remove my application from the Newton, my specialized soup should get thrown out too. How do I get my application to clean up after itself in this way? Is there a message sent to the application when it is being removed that can be caught and made to trigger a "clean up" method?

A: RemoveScript is called when the application is removed. However, you can't tell if this happened as a result of "remove software" or the user removing a PCMCIA card with your application on it.

Implement your application's removeScript script in the Project Data file. Be careful: the application has already been removed when the RemoveScript is called so you can't reference the application itself (or its slots, functions, variables, etc.)

To remove your soups, your `removeScript` could do something like this:

```
    AddDeferredAction(func() foreach store in GetStores() do begin
local soup := store:GetSoup("mySoupName");
if soup then
   soup:RemoveFromStore();
   // might also want to handle case when store:IsReadOnly() = true
end);
```

`GetStores` has problems if the `removeScript` is called as the result of a card removal event. The `AddDeferredAction` works around this problem.

Creating Soups Properly (10/5/93) (obsoleted by 1.0 Newton Programmer's Guide p7-24)

Use `RegisterCardSoup` and `UnRegisterCardSoup`. These are provided in the latest NTK platforms file via the constants `kRegisterCardSoupFunc` and `kUnRegisterCardSoupFunc`. See the platform file release notes for more information.

Count Soup Entries, How? (10/14/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-48)

Q: How do I determine the number of entries in a soup (or the number of entries returned by a Query)? I figured out how to do it with MapCursor, but is there an easier way?

A: This is not a simple case. Here's an example of a function that you could use instead of using MapCursor. This function may be slower though, but it does not bring soup entries into the frame heap, so memory performance may be better.

```
func CountCursorEntries (cursor)
begin
        cursor := cursor:Clone();
        cursor:Reset();
        if cursor:Entry() = NIL then
                return 0;
        local count := 1;
        while cursor:Next() do
                count := count + 1;
        count
end
```

Note that using MapCursor or iterating through the cursor will only determine the number of entries at the time of iteration.

Soups are fully dynamic; I can add and delete items to/from the soup as you are iterating and the cursor will take this into account. In other words, you have to iterate to find the number of entries, though they can change. Cursors are dynamic!

Getting/Setting the Default Store (11/10/93) (obsoleted by 1.0 Newton Programmer's Guide p7-65)

The latest NTK platforms file provides `SetDefaultStore` and `GetDefaultStore` via the constants `kSetDefaultStoreFunc` and `kGetDefaultStoreFunc`. See the platform file release notes for more information.

Null Union-Soups? (11/10/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-66)

The latest NTK platforms file provides `UnionSoupIsNull` via the constants `kUnionSoupIsNullFunc`. See the platform file release notes for more information.

UnionSoup Dangers (11/10/93)

Note that unique IDs of a soup entry are only valid within the same soup. If you use Union Soups you could get "unique" IDs that are the same from two different member soups. It takes the store signature, soup signature, and entry's unique ID to uniquely specify an entry in a union soup.

UniqueIDs (11/10/93)

Use the `EntryUniqueID` function instead of using the `_uniqueID` slot from the entry. The function is more efficient because it does not require that a cached copy of the entry be created, and is preferable because the unique ID implementation might (or might not) change in the future.

Soup Removal Problems, Indexing, Error -48022 (11/10/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-32)

Q: On a Newton MessagePad, when tring to remove a soup from a PCMCIA Flash card store, I received a "Internal error-something unexpected happened." message, error number -48022. What did really happen?

A: This error is returned when any of the following things happen.

- There is no space for more nodes while indexing.
- A slot is internally out of range (creating a problem finding it using offset)
- The system is unable to create index resources when creating a soup, or when adding or deleting indexes.

If you receive this error, there is a strong possibility that the soup's index definitions are or were corrupted. You might also check to see whether you are attempting to delete an index that is no longer present.

With MessagePads before the 1.05 system update, it was possible to "corrupt" soup indexes by adding entries with NIL in an indexed slot. (Not having the slot present at all is no problem, but having the slot with NIL was.) If you have a 1.05 or later MessagePad, NIL in indexed slots is perfectly legal.

Soup Sizes - Generate Test Soup (11/30/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-32)

Because of compaction and other internal issues, it's very hard to predict what a final soup size will be. One way to measure this is to generate a fake soup, add fake data to it, and measure the soup size.

Below is sample code showing how to create a text soup quickly using the Inspector:

```
call func() begin
   local s, i, j;
   for i := 1 to Random(10, 20) do begin
      s := GetStores()[0]:CreateSoup("Soup"&i&":PIEDTS", {});
      for j := 1 to Random(100, 1000) do
         s:AddEntry({
             string: GetRandomWord(5, 10) && GetRandomWord(6, 12),
             real: Random(1000) + 1/Random(1000),
             integer: Random(10000)});
   end;
end with ();
```

If you want "richer" soups with other data types, look at the Sync Another One sample; it creates a random field of nearly every data type that is used in the Newton environment.

Be Careful What you Store in Soup (12/6/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-23)

It is important to know what you store in a soup. A good example would be the references you get from various Toolbox calls, such as the GetStores function. If you save the reference, it will contain _parent references, and this will both make the entry quite large, and will make it contain references to other outside things. Soup entries should not contain any such references. They should be self contained.

In other cases, a references may be pointers to data outside the NewtonScript world (in the case of recognition units or viewCObjects, for instance). You can't save these because they only make sense during a particular time period.

If EntryChange fails, what happens? (12/9/93)

Q: Suppose I add entries in a soup and call `EntryChange`, and it turns out there's no space in the store, and an exception is thrown. If I call `EntryUndoChanges`, do I get the original, unmodified entry back?

A: Yes. The entry will be OK because soups are transaction based, and the "update" transaction was incomplete. `EntryUndoChanges` will revert the local copy to the same data stored in the soup, which has not been modified.

Performance: Cursors and EntryRemoveFromSoup (12/11/93)

`EntryRemoveFromSoup` has a side-effect of calling `Move(1)` on any cursors pointing to that entry. For poorly written queries, this can cause `validTest` to be called on all remaining items in the soup. Here's an example of a possible scenario:

```
local c := query(GetStores()[0]:GetSoup("soupName"), {
       type: 'index,
       indexpath: 'somePath,
       startKey: xSymbol,
       validTest: func(item) item.id = msg.id,
      });

local e := c:entry();
if (e) then
   begin
     EntryRemoveFromSoup(e);
   end;
```

When `EntryRemoveFromSoup(e)` is executed, the cursor `c` is "pointing at" that soup entry. In the process of deleting it, the system must figure out which entry the cursor should point to next , so it does the equivalent of `c:Move(1)`. Because of this, the `validTest` function will be called until a new "valid" entry is found, which could be slow.

When possible, always use an `endTest` in your queries to stop the cursors as soon as possible. Avoiding keeping around unused cursors will also help.

Always use Constants instead of Strings for Built-in Soups (12/12/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-23)

Always use defined constants instead of string names for built-in soups! This is preferable because it guarantees your software to run on other Newton devices where the soup names may have changed. It also saves a few bytes in your application.

Here's a list of the currently available built-in soup constants:

```
ROM_calendarsoupname
ROM_cardfilesoupname
ROM_inboxsoupname
ROM_outboxsoupname
ROM_paperrollsoupname
ROM_systemsoupname
ROM_todosoupname
```

Do Not Clone Cursors with Clone/TotalClone/DeepClone (12/19/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-50)

Do not clone the cursor using the `Clone` functions (`Clone`, `TotalClone`, `DeepClone`). Use the cursor method called `cursor:Clone`. This will do the right thing. Cursors are not like normal frames, they are special.

What Store is Removed? (12/19/93)

Q: Is there a way in my `RemoveScript` to find out which store is being removed so that I don't try to access anything on it?

A: Currently there is no good way to determine which store is being removed. Engineering is working on a new way of treating stores and soups. Stay tuned for more information later.

Store: GetInfo, SetInfo, GetAllInfo Functions (12/19/93) (Obsoleted by 1.0 Newton Programmer's Guide p7-12)

```
store:GetInfo(symbol)
```

This method takes a symbol that will specify a specific symbol frame in the store, and will return the value for this symbol.

```
store:SetInfo(symbol, value)
```

This method takes a symbol and a value, and creates a slot inside the info frame of the Store.

```
store:GetAllInfo()
```

This function will return all information stored in the info frame of the Store.

Store:SetInfo and Soup:SetInfo (3/9/94)

You need to use `EnsureInternal` on both arguments to `SetInfo` (at least args that aren't immediates). This applies to both the soup method and the store method.

Example:

```
store:SetInfo(EnsureInternal('piSymbol), EnsureInternal(3.1415));
```

Magic Pointers in soups (3/10/94)

Related to the previous topic, you should be careful not to try to put magic pointers (for example, `ROM_fontsystem10`) into soups. You can't actually put a magic pointer into a soup. If you try, the magic pointer is followed before being copied into the soup. However, the RAM-based frame for a soup entry (or soup info or store info) will still have a magic pointer in it. If at this point you try to sync using Newton Connection, you may get a -48201 error from the Connection software (depending on the Newton ROM).

You can detect if something is a magic pointer using the function `IsMagicPtr`. If you `Clone` a magic pointer, you get back a copy of the object, not a magic pointer. Thus you can use`Clone` to get something safe to put in a soup. Alternately, you could add the entry, then immediately throw away the RAM copy, force a garbage collection, and query the soup to get a "fresh" copy of the entry which will no longer contain a magic pointer.

Although the problem described here is subtle and tricky to understand, it rarely arises in practice. Once it occurs, if you reset the Newton, it will usually not occur again.

Application Preferences and the internal store (1/26/94) (Obsoleted by 1.0 Newton Programmer's Guide p7-33)

Q: Is there any reason the information we put into the "system" soup (our preferences) or the information we put into the "directory" soup (our docker info) needs to be on the internal store (store 0)? Can we use a unionsoup and AddToDefaultStore, so they can follow the rest of the user's data around with the card?

A: There are a couple of reasons why you want to put your preferences on store zero. One reason is if your application is on a card and is moved from Newton to Newton, you want the application to look first to see if the current Newton already has preferences for your application -- that way, that Newton's owner knows what to expect.

Also, if you merely add the preferences to the default store, you could get into a situation where your preferences are on the card but your application is in internal memory. In this case, if the card is pulled you're out of luck.

Correct name of _uniqueID slot (NEW 5/31/94)

Q: I'm unable to use the `uniqueID` slot to manipulate soup entries. I see it in the Inspector, but my code keeps breaking. What's wrong?

A: The correct name of this slot, `_uniqueID`, begins with an underscore. Unfortunately, the Data Storage and Retrieval chapter of NPG 1.0 omits the underscore in several places; this error will be corrected in the next version of the NPG. You should use the `EntryUniqueId` function instead of accessing the slot directly, as the function is more efficient and will always return the correct ID even if the slot value in the cached entry is modified.

Nil Index Slot, Adding Entries (6/7/94)

The original MessagePad had a problem which caused soup indexes to become corrupted if an indexed slot in any soup entry existed and contained the value `NIL`. The system would not warn of this, and you would end up in trouble later when doing Queries.

This was fixed in System Update 1.05 and later releases.

`NIL` in an indexed slot now behaves the same as if the slot were not present. That is, that soup entry will not be findable using the specified index. (All entries can always be found with the default index.)

Indexes in decreasing order? (6/7/94)

Q: I'm doing a little database app, and in my overview, I want to be able to do a "sort by" control. Choosing the column is easy (different indices, just switch cursors). But choosing ascending/descending is trickier. How do you make cursors go backwards from their "normal direction" ?

A: Indexes are always sorted in ascending order, but your cursor does not have to only move in the "natural" direction. If you use the `Move` method and have a "direction" constant/global that is either 1 or -1, you can effectively make an index appear to be increasing or decreasing by doing:

```
cursor:Move(gDirection * amount);
```

However, the built-in behaviors for moving to the nearest key, such as with the `startKey` in an index, or `cursor:GotoKey(key)`, will still operate in ascending order. You can fix this by calling `Prev` in appropriate places. For example:

```
cursor:GotoKey(theKey);   // moves the cursor to the key specified
                                // or to next highest key if not found.
if gDirection = -1 and cursor:Entry().key <> theKey then
      cursor:Prev();  // now cursor is at next lowest key from theKey
```

startKey and endTest and cursor limits (6/9/94)

Q: If I create a cursor with `startKey:"Bruce"`, then do `cursor:Reset` and `cursor:Prev`, it seems like I should get `NIL`, but I get the entry `"Allan"` instead. What's wrong?

A: `StartKey` is used to position the cursor initially (and after a `cursor:Reset`.) It is correct to think of `StartKey` as an implicit `cursor:GotoKey`.

`EndTest` is called on a candidate entry whenever the cursor moves, either forward or backward. If the `EndTest` function returns true, the cursor is assumed to be at the "end" of the soup, similar to `prev`'ing or `next`'ing past the last entry.

It's perfectly correct to have an `endTest` that limits the soup to entries between "C" and "G", and a `startKey` that initially positions the cursor at the first "E".

Your `endTest` should be written so that it will return `TRUE` when the cursor moves off either end of the soup. For example:

```
func(e) StrCompare(e.sortOn, "D") > 0 or StrCompare(e.sortOn, "B") < 0)
```

When the `endTest` returns `TRUE`, you can think of it as "moving" the cursor off the beginning or end of the soup, depending on whether you were moving forward or backward. When you then try to `:Next` or `:Prev` to the last/first entry, the actual last/first entry in the soup is tested with the `endTest` function. If (as in the above case), the last or first entry causes `endTest` to return `TRUE`, the cursor will again be positioned at the beginning/end of the soup, and the cursor's entry will be `NIL`. Your cursor is "stuck." at the end. If you expect this to happen, you can use the `:Goto` or `:GotoKey` methods to "unstick" the cursor.

Interraction of EntryChange and MapCursor (6/9/94)

You should avoid updating a soup entry with `EntryChange` inside a `MapCursor`. Unexpected results can occur if you update the value of an indexed slot. The `EntryChange` will cause the cursor to do a `Goto(Entry)` before the `MapCursor` executes the `cursor:Next`. This can result in skipping cursor entries.

We recommend that you avoid using `EntryChange` or otherwise modifying soup entries inside of a `MapCursor`.

Use GC with RemoveIndex (6/10/94)

Currently the system has a bug where `RemoveIndex` does not succeed in all instances, for instance:

```
c := Query(s, {type: 'index, indexpath: p };
c := nil;
// GC();

s:RemoveIndex(p);
```

Expectatation is that this should succeed; in fact, an error is generated unless the GC is executed.

```
{name: |evt.ex.fr.type;type.ref.frame|, data: {errorCode: -48400, value:
<array>...
```

The workaround at this point is to do the following steps before a RemoveIndex:
1. Do a GC
2. Check for cursors, if they are still in use, signal an error (the index is still used).

Beware of Indexes inExternal Soups in Combination with RemoveIndex and AddIndex (6/10/94)

`AddIndex` and `RemoveIndex` work fine with union soups. However, if the user inserts a card, and the card has a soup (that will become a member of the union) which has indexes missing, you will have problems.

If your application is registered for soup changes, then when the new soup joins the union you will be notified. However it's up to your code to ensure that all necessary indexes are present on all union members before using the index in a query.

Note that this will be addressed in a future Newton products.

AddToDefaultStore catches exceptions (7/5/94)

Q: `AddToDefaultStore` returns `NIL` if the memory is full. I've added an exception handler as indicated by the docs, but it never gets called. What's wrong?

A: The implementation of `AddToDefaultStores` catches the exception, posts the notification for the user, then returns `NIL`. This may be fixed in future releases. If you need to catch the exception, you can use the `kGetDefaultStoreFunc` provided in the NTK platform file to get the appropriate soup and use the `Add` method on that soup, e.g.:

```
try
   (call kGetDefaultStoreFunc with
()):GetSoup(kMyAppSoupName):Add(theEntry);
onexception '|evt.ex.fr.store| do
   // handle the exception somehow...
```

Note: be sure to get the default store for each entry or batch of entries, as the user may change it (by inserting a PCMCIA card, for example) while your application is running.

FrameDirty is deep, but can still be fooled. (8/19/94)

Q: Does the global function `FrameDirty` see changes to nested frames?

A: Yes. However, `FrameDirty` is fooled by changes to bytes within binary objects. Since strings are implemented as binary objects, this means that `FrameDirty` will not see changes to individual characters in a string. Since `clParagraphViews` try (as possible) to work by manipulating the characters in the string rather than by creating a new string, this means that `FrameDirty` can be easily fooled by normal editing of string data.

Here is an NTK Inspector-based example of the problem:

```
s := GetStores()[0]:CreateSoup("Test:PIEDTS", []);
e := s:Add({slot: 'value, string: "A test entry", nested: {slot:
'notherValue}})
#4410B69  {slot: value,
           String: "A test entry",
           nested: {slot: notherValue},
           _uniqueID: 0}
FrameDirty(e)
#2        NIL

e.string[0] := $a;      // modify the string w/out changing its reference
FrameDirty(e)
```

```
#2        NIL

EntryChange(e);
e.string := "A new string";   // change the string reference
FrameDirty(e)
#1A       TRUE

EntryChange(e);
e.nested.slot := 'newValue;   // try a nested change, FrameDirty is deep.
FrameDirty(e)
#1A       TRUE

s:RemoveFromStore()  // cleanup.
```

Dangers of UniqueID and Connection Restore. (8/25/94)

Q: When I restore a Newton from a Connection synch file, all the uniqueIDs for my soup entries are different. How can I create cross-entry links?

A: Unfortunately, a bug in the Connection Kit (including the current 2.0.2 version) causes uniqueIDs for soups to change after a restore. This will eventually be fixed, but in the meantime, you cannot reliably use uniqueID to "link" two entries together. Since Connection Restore recovery process is used only rarely, we recommend providing some means to "reconnect" the soup entries through potentially slower means, and advising users that after a restore this relinking process will be required.

PCMCIA backup/restore does not exhibit this problem.

Single Unicode Characters and Soups/NTK (10/11/94)

Q: When I store a frame containing a single Unicode character in a soup, I get back invalid values. What is happening?

A: There is a bug in the code that reads single characters from stores; this bug is encountered only if the high byte of the character is non-zero; (that is, any page besides page 0 of the Unicode standard.) This causes invalid NewtonScript objects to be created when a character of this type is read from the soup.

A simple workaround is to store the character as a string instead of as a single Unicode character. Storing the `Ord` value of the character as an integer will also work and will be slightly smaller, but requires less convenient code.

This bug can also be encountered when typing Unicode characters into NTK's Inspector window. For example, `$\u2022` can produce an error. Use `"\u2022"[0]`or `Chr(0x2022)` instead, to produce the same character.

store:UsedSize and card 'applet' memory size varies (10/13/94)

Q: The numbers returned by the card 'applet' for the size free and used on the card seem to be inconsistent. Similarly, the store method `UsedSize` doesn't seem to reflect the real used size. How can I find out exactly how much space is available on a store?

A: There is no way to find out exactly how many bytes are available on a store. The numbers shown in the card dialog or those returned from `store:UsedSize` are only approximate. The exact amount of free space is difficult to determine because of overhead used to manage the store. How much space an object will require is also difficult to determine to the last byte because of this overhead.

The value displayed or returned is normally accurate to within a few K.

Potential Problems with Soup Entry _proto Slots (11/10/94)

Q: The Newton Programmers Guide (7-35) states that `_proto` slots in soup entries are not touched when we add items to soups, but it appears that the system is trying to stuff them into the soup. Why would the following code crash in the Inspector?:

```
// crashes (runs out of frames heap)
mySoup:Add({x: 42, _proto: GetRoot()});
```

A: The current implementation of the soup entry cache performs an `EnsureInternal` on the frame when an entry is written to a soup. Even though the `_proto` slot is never added to the soup, any frame that directly or indirectly points to objects in non-system ROM will be duplicated, taking up tremendous space in the NewtonScript heap.

In your example, GetRoot will reference many things which are not already in the NewtonScript heap or in ROM, and will easily overfill the NewtonScript heap. (Even if you were lucky and everything was either internal or in ROM, tracing the entire tree of references in NewtonScript could be very time-consuming.)

The `_proto` slot will never actually be written to the soup, as the Newton Programmers Guide states. In practice, you should clip the `_proto` slot when adding a soup entry. Here is some code to work around this problem if you need to beware of your `_proto` slot:

```
local temp := f._proto;
f._proto := nil;  // the _proto slot never goes into the soup anyway!
soup:Add(f);      // Add() might do an EnsureInternal...
EntryChange(f);   // EntryChange might also do an EnsureInternal...
f._proto := temp;.
```

AddWithUniqueID documentation bug (12/7/94)

Q: Why can't I use `AddWithUniqueID` to replace an existing soup entry with one of my own design that has the same `_uniqueID`? It throws an exception instead of replacing the entry that has the same `_uniqueID`.

A: `AddWithUniqueID` is working properly; the documentation for `AddWithUniqueID` on page 7-37 through 7-38 of the Newton Programmer's Guide is incorrect. `AddWithUniqueID` throws an exception if the entry you pass to it has the same `_uniqueID` as an entry already present in the soup to which you are adding the new entry.

The correct way to replace a soup entry is to get the entry using a cursor, modify the entry, and call `EntryChange`. `AddWithUniqueID` should not be used in the course of normal NewtonScript programming; it is only meant for specialized soup restoration applications.

CHANGED: Limits on sizes of soup entries (3/7/95)

Q: How big can I make my soup entries?

A: In practice, entries larger than about 16K will significantly impact performance, and 8K should be considered a working limit for average entry size. No more than 32K of text (total of all strings, keeping in mind that one character is 2 bytes) can go in any soup entry.

There is no size limit built into the NewtonScript language; however, another practical limit is that there must be space in the NewtonScript heap to hold the entire soup entry.

There is a hard upper limit of 64K on Store object sizes for any store type. With SRAM-based stores there is a further block size limit of 32K. Trying to create an entry larger than this will result in `evt.ex.fr.store` exceptions. These limits are for the encoded form that the data takes when written to a soup, which varies from the object's size in the NS heap.

End of DTSQA

DEBUGGING

TABLE OF CONTENTS:

DEBUGGING.....1
Introduction.....1
Path Failed 16 Error (9/15/93).....1
References to Child Frames Causes Memory Problems (9/15/93).....1
Nil = Nil returns True (9/15/93) (Obsoleted by NTK User's Guide, Chapter 5, Debugging).....1
Stack UnderFlow Errors (9/15/93).....2
Memory Statistics and Stats (11/11/93) (Obsoleted by Newton Programmer's Guide Documentation, Utility Functions chapter).....2
Print Tricks (12/19/93).....2
Making Trace take effect immediately. (1/10/94).....3
Don't Forget to Clean Up the Prints (2/21/94).....3
Check for Application Base View Slots (3/6/94).....4
Use Throw! (3/6/94).....4

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.

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)

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.

End of DTSQA

DIGITAL BOOKS

TABLE OF CONTENTS:

DIGITAL BOOKS 1
Introduction 1
BookMaker Page Limitations? (11/19/93) 1
Format Information (11/24/93) 1
Printing Ranges of Pages (11/29/93) 1
BookMaker Requires XTND 1.3.6 (12/6/93) 2
Searching Non-Text Content Items (12/13/93) 2
Scrolling in a Book (12/16/93) 3
TurnToPage (12/17/93) 5
Opening a Book to a Specific Page (12/18/93) 5
Espy Font in Bold (12/26/93) (Obsoleted by 1.0 Newton Book Maker User's Guide pB-2) 6
Revised Scripts Caution (inherit viewSetupFormScript!) (12/30/93) 6
Dot Commands in Story Blocks (1/4/94) (Obsoleted by 1.0 Newton Book Maker User's Guide pB-2) 6
Quote Pathnames in Chain Command (1/17/94) (Obsoleted by 1.0 Newton Book Maker User's Guide pA-25) 7
Adding Custom Book Icon to Extras drawer(1/26/94) (Obsoleted with NTK 1.0 Final) 7
PICT2 Resources, Color Drawing Calls, BookMaker (2/25/94) 7
Opening Views from Book Scripts (3/14/94) 7
Books, MP and MP110 Differences? (3/24/94) 8
Default message for .script (5/26/94) 9
Curly quotes don't compile (5/31/94) 9
Create Function Libraries for Book (6/7/94) 9
Controlling how Find results are displayed. (6/9/94) 10
.preamble command outputs NewtonScript before content (6/30/94)10
.postamble command outputs NewtonScript after content (6/30/94)11
Do not duplicate Layout names in a book (6/30/94) 11
Only use .option once for each option (6/30/94) 11
How to jump to a particular content item (6/30/94) 11
Update to documentation of FindPageByContent (6/30/94) 12
.title not updated on loading updated book (10/5/94) 12
No scripts in page .headers, use .running story (11/22/94) 13

Introduction

This document addresses Newton Digital Books 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.

BookMaker Page Limitations? (11/19/93)

Q: Does BookMaker have any limitations concerning the size of books or page count?

A: The current page limitation of BookMaker is 16 million pages, a very unlikely size to be exceeded. However, since the entire book is held in memory during processing, you need to have enough application heap space allocated to BookMaker.

There is also a limitation on the size of a single storyto 32K. If you have larger story sections, the workaround is to break bigger stories into smaller ones using the .story command. Also, if this is a problem, you could consider breaking the book into more (and smaller) chapters and subjects.

Format Information (11/24/93)

Q: Even though we say that you should not rely on specific content being on a particular page (larger screens may lay out the book somewhat differently, thus moving things around) there may be times when, for speed, you need to hard-code a page. How can you tell easily on which format your book is running ?

A: There is a slot called curRendering that can tell you. If curRendering is 0, your format is that of the MessagePad.

Here's an example of how to use it:

```
if (curRendering = 0) then :TurnToPage(destPage);
  else :TurnToPage(:FindPageByContent(dest, 0, NIL));
```

This snippet, which might be used in a .script, says that if the curRendering is 0, then do a fast turntopage (You might have figured out the page with some compile-time scripts.) If not, use the FindPageByContent method.

Printing Ranges of Pages (11/29/93)

Q: I am having trouble updating a book. If I change the .title to be a different date, the change is not reflected when I load the new book. What am I doing wrong?

A: The book reader engine on Newton caches certain parts of the book like the title and the current page a user is on. To get the engine to notice the title has changed, you have to change the .isbn of the book before you load the updated one.

If you are likely to update your book, you should use a dash in the .isbn to indicate version or date. An example would be:

```
.title My Info Book of 7/94
.isbn  IB-7-94:PIEDTS
```

BookMaker Requires XTND 1.3.6 (12/6/93)
(Obsolete with NTK 1.0, which includes XTND and translators)

To use any document format besides TEXT as input, BookMaker requires Claris XTND 1.3.6 and the appropriate XTND translator for that file format. The XTND translators are included with On Location, FileMaker Pro and several other products from Claris. NTK 1.0.1 includes the XTND engine and a few translators.

Searching Non-Text Content Items (12/13/93)

Q: Our book uses layouts to display the names of hotels, restaurants, and so on, as opposed to using text. This is done to provide a certain graphical look. The problem is that the BookReader in the Newton ROM does not search into frames. It only searches text. This causes a problem when the user wants to use Find to find a particular place.

A: Here are some alternatives:

1) Use text and not layouts. Use a two-column layout with 1 grid unit on the left and 11 on the right. In this way you can have the glyph on the left and the text next to it on its right as you have it now, with the benefit of the text being searchable (and highlighted on find). (Note: a 2-10 layout might work better - not sure of your glyph size).

Advantage: Easy to do, little overhead
Disadvantages: Must use New York, Geneva or Espy for name. Might conflict with the page layout already in use (unless it's just a 12-column layout).

2) Add a story to the page with the name of the location, so that it can be found with a search, but have the story live underneath the frame you have now, so the story is hidden from a user's view.

Advantage: Graphic design maintained
Disadvantages: Requires writing some NewtonScript. Found text will not be highlighted (but you will get to the right page).
Here's what the NewtonScript would look like:

```
.story
Joe's Place
.attribute myForm: layout_myForm
.script viewSetupDoneScript
local theform, theview;
theForm := {_proto: item.myForm, viewBounds: viewBounds};
theView := GetView(:InsertForm(theform));
theView:show();
SetValue(self, 'text, "");
.endscript
```

Note that you might have to tweak the right bounds of the frame since the story paragraph view extends all the way across its column. A clever thing to do might be to use the story's text as the text in the form so as not to waste those bytes.

A suggested alternative was to make the frame a child of the paragraph view at viewsetupchildren time. Either of these should work for you. In the future, we will add to BookReader automatic searching of frames.

Scrolling in a Book (12/16/93)

Q: Book Reader handles the scroll arrows for paging through a book. However there are times when I want to intercept the scroll arrows for scrolling a view that is currently being displayed on top of the book. Or I might add a pane to the browser and want scrolling in it. How do I override the scrolling?

A: You'll need the following methods in your scrolling layout:

viewScrollDownScript
viewScrollUpScript
viewShowScript
viewQuitScript
viewHideScript
RestoreScrollHack

At viewShowScript time we replace Book Reader's (copperfield) viewScroll methods with our own:

```
viewShowScript: func()  // Replace Copperfield's scrolling
copperfield.viewShowScript := func()
  begin
    copperfield.viewScrollUpScript := func()
      begin
      // If the browser is visible we need to pass scrolling to it
       if (Visible(browser)) then
         browser:viewScrollUpScript();
       else
         scrollStealer:viewScrollUpScript();
     end;
    copperfield.viewScrollDownScript := func()
     begin
      // If the browser is visible we need to pass scrolling to it
       if (Visible(browser)) then
            browser:viewScrollDownScript();
       else
          scrollStealer:viewScrollDownScript();
      end;
      // Now set a slot so the bookreader (copperfield)
      // knows who scrollStealer is.
     copperfield.scrollStealer := self;
  end;
```

At viewQuitScript and viewHideScript time we'll restore Book Reader's viewScroll methods:

```
viewQuitScript: func()
  begin
    :RestoreScrollHack();
  end

viewHideScript: func()
  begin
    :RestoreScrollHack();
  end
```

Here's the method that does the restore:

```
   RestoreScrollHack: func()   // Restore Copperfield's scrolling
     begin
       copperfield.scrollStealer := NIL;
       bookReader.viewScrollUpScript := copperfield._proto.viewScrollUpScript;
       bookReader.viewScrollDownScript :=
copperfield._proto.viewScrollDownScript;
```

```
        end
```

Important Note: Do not take over the scrolling arrows unless the use of your scrolling view is a mode. In other words you may NOT have scrolling views as part of a page in a book. Your view must be one that floats on top of the book.

Note: Future Newtons will provide a better way for your views to receive scroll messages.

Scrolling in a browser pane is accomplished much the same way.

You'll need the following methods in your browser pane layout:

viewScrollDownScript
viewScrollUpScript
viewSetupDoneScript
viewQuitScript

The viewScrollUpScript and viewScrollDownScripts are the standard methods. In your viewSetupDoneScript you override the receiver of the scroll arrows:

```
   viewSetupDoneScript: func()  // Hack the browser so we get scrolling
     begin
       browser.viewScrollUpScript := func()
         begin
           broScrollStealer:viewScrollUpScript();
         end;
       browser.viewScrollDownScript := func()
         begin
           broScrollStealer:viewScrollDownScript();
       end;
         // Now set a slot so the bookreader (copperfield) knows who
         // broScrollStealer is.
       copperfield.broScrollStealer := self;
     end,
```

Be sure to have a viewQuitScript to remove the scroll hack and restore normal scrolling:

```
   viewQuitScript: func()  // Remove our scroll hack
     begin
     // Restore the old scroller
       browser.viewScrollUpScript := browser._proto.viewScrollUpScript;
       browser.viewScrollDownScript := browser._proto.viewScrollDownScript;
         // Setting paneIndex to 0 works around a bug with alternate       //
browser panes
       browser.paneIndex := 0;
       copperfield.broScrollStealer := NIL;
     end,
```

Note: Future Newtons will correctly send to browser panes the scroll messages. The above workaround is temporary.

TurnToPage (12/17/93)

Q: My books have problems if my script calls `TurnToPage`.

A: Book Reader has a problem if Book Reader itself is not the thing that turns pages. You can word around this by hiding the current storyCard (if any). Assuming you had a button which caused a `TurnToPage`, you could do this:

```
         buttonClickScript: func()
```

```
            begin
                  // get rid of story card before turning away...
                  if storyCard then
                        :HideStoryCard();
                  // your code here...
            end;
```

Opening a Book to a Specific Page (12/18/93)

Q: How do I open a book to a specific page?

A: You first open it, then use one of the book functions to turn it to the proper page.

In order to open a book you need to know its ISBN. Every book must be uniquely identified with a .isbn command in the source. To see the ISBNs for all your installed books type this in the Inspector:

```
call func() begin
local i;
foreach i in extras do
      if i.app = 'copperfield then
            print(i.isbn);
end with ()
```

For the 800#s demo the ISBN is "800-PENSEE". To open that book you' do this:

```
cu := GetRoot().Copperfield;
cu:OpenBook("800-PENSEE");
```

If you happen to know something about the structure of the book, you can also turn to a specific page like this:

```
cu:TurnToContent('name, "hotels");
```

Espy Font in Bold (12/26/93) (Obsoleted by 1.0 Newton Book Maker User's Guide pB-2)

Q: I use Espy Bold and my document looks wrong on the Newton.

A: We recommend against using Espy for normal books, since it does not print well (it has no equivalent font on most PostScript printers) but if you insist... Be sure you use the Espy Sans Bold font. Do not use Espy Sans with bold style. Espy Sans Bold and Espy Sans with bold style each measure differently on the Mac. The Newton uses Espy Sans Bold when it sees Espy Sans with bold style.

Revised Scripts Caution (inherit viewSetupFormScript!) (12/30/93) (May be Obsoleted by 1.0 Newton Book Maker User's Guide p4-3)

Because there's an important `viewSetupFormScript` method in the `contentArea` view that is called when the page is imaged, try to avoid using `viewSetupFormScript` method at the book or page level. Instead, use the `viewSetupDoneScript` method, which is called every time a page is imaged (except when printing). If you must use the `viewSetupFormScript` method, be sure that your `viewSetupFormScript` method calls the `inherited:viewSetupFormScript` method.

Dot Commands in Story Blocks (1/4/94) (Obsoleted by 1.0 Newton Book Maker User's Guide pB-2)

Q: Where do I put dot commands (like .attribute) that go with a content?

A: Dot commands can technically be placed anywhere after the .story or .subject line that begins the content and before the dot command that starts the next content. However, the beta Book Maker can lose track of style information if dot commands (like .attribute) are placed in the middle of runs of text. For stylistic reasons, we recommend placing .attribute (and other commands like .script, .mark) immediately after the .story or .subject lines.

Quote Pathnames in Chain Command (1/17/94) (Obsoleted by 1.0 Newton Book Maker User's Guide pA-25)

The `.chain` command requires you to enclose in double quotation marks the pathname passed as its argument. For example,

```
.chain "My Disk:My Folder:My File"
```

Note that you can include spaces in the pathname as long as its quoted properly.

Adding Custom Book Icon to Extras drawer(1/26/94) (Obsoleted with NTK 1.0 Final)

Q: How can I replace the default book icon in the Extras drawer with one of my own?

A: With NTK 1.0 and later versions, you can set the icon for a book in the NTK "Project Setting" dialog just as you would for any other package built with NTK.

With earlier (released) versions, use the following code in your book's preamble or postamble:

```
.postamble
book.icon = GetPictAsBits("myIcon", true);
```

You can put this icon in a `.rsrc` file that you've included as part of the NTK project or you can use the `OpenResFileX` function to open the resource file yourself.

We recommend setting up NTK to use the icon, which will ensure that the resource file is part of the project and that it is open at compile time; add the .postamble as a temporary measure.

PICT2 Resources, Color Drawing Calls, BookMaker (2/25/94)

Only the black and white QuickDraw instructions in PICT2 resources will work in the current Newton environment. The Newton view system skips over any drawing calls it does not support; no color calls will be made.

Youc ould fix this by editing the .f book file. Change the GetResource calls to GetPictAsBits calls. The BookMaker will then image the PICT2 resources into black and white bitmaps at compile time. However, this usually means the package produced will be larger, because PICT is a more efficient storage format than bitmaps.

Opening Views from Book Scripts (3/14/94)

Q: How can I have a book script open a new layout?

A: Use the InsertForm book command. InsertForm takes as an argument a template to add. With NTK, you can use layout_<layoutName> here.

However, you should be careful. In early versions of NTK, the layout_<layoutName> symbols are not constants, but rather compile-time variables. The big difference is that if they are constants, you can use the symbol within a script and it will get the right thing. If it's a compile-time variable, the reference to the slot will be left in the script and you'll get run-time errors.

However, using DefConst can help here. In your book, you can do something like DefConst('klayout_<layoutName>, layout_<layoutName>). This will cause NTK to create a constant from the compile-time variable, which you can then reference from within your script.

None of this is necessary with NTK 1.0 and later. The layout_<fileName> symbols are constants, and are available after NTK has compiled the layout. (See the NTK docs and Q&A for compilation order issues.)

Another issue is that InsertForm doesn't return the view created; instead it returns the template. In this case, use GetView to get the view frame.

The created view may not be visible. Send it the Open message.

Finally, NTK turns off the vVisible bit in the top level view of the "main" layout. (This is usually good, otherwise applications would open when installed.) If you use user protos instead of layouts NTK won't change the proto's vVisisble bits. User protos can still be referenced as layout_<protoFileName> If you use print formats, NTK will compile the print format (and produce the constant) before the book source is processed.

The complete script to open the view looks like this:

```
.story
Joe's Place
.script viewClickScript
      GetView(:InsertForm(layout_<protoFileName>)):Open();
.endscript
```

Books, MP and MP110 Differences? (3/24/94)

Q: I created a simple telephone directory using BookMaker and NTK. Everything works fine on the original Newton (MP), but when running it on the MP 110 I have found that the text is not paged correctly. It should be showing one less line of text. It obviously thinks it has those extra 16 pixels.

Why is the text not being formatted properly for the screen?

A: The Newton Book Reader in the MessagePad 110 is slightly different from the one in the original MessagePad.

To make room for the entire book content area, the status bar on the MessagePad 110 has been 'collapsed' into the little clock icon you see in the lower right corner.

The status bar happened to be 16 pixels high. Removing it left just enough space for books to keep the same page breaks and other layout attributes on both platforms.

So, you are not seeing a bug, or other side-effect. What is happening is by design.

Default message for .script (5/26/94)

The default value for the event parameter to the .script command is buttonClickScript, not viewClickScript, as stated in the Book Maker User's Guide command reference pp. A-16 & 17.

Also the code example on page A-17 is slightly incorrect. Here is the corrected version:

```
.#  by default, its a buttonClickScript
.script
playSound(ROM_click):
inherited:buttonClickScript
.endscript
```

Alternatively, you can correct the example to make the inherited call match the message specified in the original .script command, as in this example.

```
.script viewClickScript
playSound(ROM_click):
inherited:viewClickScript // call the correct inherited method!
.endscript
```

Curly quotes don't compile (5/31/94)

Q: When I add Print statements to a script in my digital book, I get "illegal character" error messages from NTK at compile time. What is happening?

A: NewtonScript is restricted to 7-bit ASCII, and you could have inadvertently added characters that are outside this range. For example, the problem can be caused by the "smart quotes" feature in the word processor you're using to write your book source file. Curly Quotes are not part of the ASCII character set, they are part of the Macintosh extension. If you use only straight quotation marks in NewtonScript statements and those irritating messages will go away. For example:

```
.script
Print(“Whoops”) // won't compile
Print("Whoops") // will compile
.endscript
```

Create Function Libraries for Book (6/7/94)

Q: I would like to create a library of functions that I could use inside the book package at run time. How do I tell the compiler to include the function or functions into my book package so that I can call it during run time?

A: Here's one solution:

```
.preamble
book.data.myFunction := func(arg) begin … end

.script buttonClickScript
local f := :BookData().myFunction;
// See book chapter for BookData warning
call f with (someArg);
```

```
    .endscript
```

Another, possibly better solution is to simply use a constant for the function or functions. Define them in your NTK Project Data file and call them later. An equivalent example:

```
    // in project data
    DefConst('kMyLibraryFunc, func(arg) begin ... end;

    // in book source
    .script buttonClickScript
    local f := call kMyLibraryFunction with (someArg);
    .endScript;
```

Controlling how Find results are displayed. (6/9/94)

Q: When the user perfoms a search with the Find button and more than one item is found, some text is displayed in the find overview to show the item "in context." How can I control what text is displayed?

A: The Newton BookViewer application in the ROM generates the "in context" text from your source document. It chooses a few words before and after the word that was found as the context. When looking, spaces, tabs, and carriage returns are each treated as a word. However, if there is a borderline between two content areas (e.g. separate .story items), text from the adjoining area will not be displayed in the find overview.

To avoid nearby text being displayed as part of the overview, you can add additional separation in the form of extra spaces, tabs, or carriage returns, or you could make each piece of text a single .story content item.

.preamble command outputs NewtonScript before content (6/30/94)

Q: How do I define NewtonScript constants and code that can be referenced in the content of my book?

A: You can use the .preamble command: `.preamble`

This allows NewtonScript to be copied directly into the Bookmaker output file. The text is placed before any definitions of content. The .endpreamble is optional.

```
    .preamble
    book.data.myStuff := specialProto;
    .endpreamble
```

.postamble command outputs NewtonScript after content (6/30/94)

Q: How do I define NewtonScript constants and code that will appear in the final book output by NTK?

A: You can use the .postamble command:`.postamble`

This allows NewtonScript to be copied directly into the Bookmaker output file. The text is placed after all definitions of content and internal book frames. The.endpostamble is optional.

```
    .postamble
    book.data.specialContent := c1;
    .endpostamble
```

Do not duplicate Layout names in a book (6/30/94)

Layout names cannot be duplicated. Book Maker gets very unhappy if you try to define a layout with the same name as one previously defined.

Only use .option once for each option (6/30/94)

Q: BookMaker sometimes seems to ignore options when I set them.

A: The .option command should only be used once for each applicable option.

How to jump to a particular content item (6/30/94)

Q: I would like to provide a help button that jumps to a particular piece of content, then jumps back again when the user has finished.

A: Often you want to give yourself a reference to a particular content item so that later you can turn to its page or access its data. For example, you might have a help button in a page header. If this page header was a view (.form), it might be difficult to use Book Maker's `goto` to give you that reference. Thus you can use this trick.

The technique is to use the `.index` command to build references to these content items.

For example if I have this:

```
.story
.index !crumbs a
Hansel & Gretel
```

This will create an array in `bookdata` called `crumbsIndex`. Everytime the `.index` command appears on a story, a reference to this content (story) gets added to the array. By knowing a little about which items are indexed this way, the array can be used to get a reference to the content.

If speed is a major concern you might want to store the page numbers that these content items have showed up on. To do so, turn on the `indexedPage` option by putting at the top of the file:

```
.option indexedPage
```

This will add a slot to each indexed content item (indexed with the `.index` command) called `firstPage` which contains the page number that this content first appears on. (Note: this slot maybe an array of page numbers in the case of a book layed out for different sized screens).

Then to get to the page number for a particular content use:

```
page := :?BookData().crumbIndex[which].firstPage;
```

Where `which` is the indexed content wanted. If you prefer you could post-process the index output with NewtonScript in the postamble in order to put specific slots into bookdata. For example:

```
.postamble
book.data.helppage := book.data.crumbIndex[0].firstPage;
book.data.glossary := book.data.crumbIndex[0].firstPage;
book.data.helpIndex := NIL;
```

```
    .endpostamble
```

This will add slots to bookdata with the relevant page numbers in them and then nil the index afterwards so it takes no space in the compiled package.

Update to documentation of FindPageByContent (6/30/94)

`FindPageByContent` finds the page that the characters of `aContent` at `anOffset` can be found on. (Note that Contents are frequently split between pages.)

```
    :FindPageByContent(aContent, anOffset, NIL)
```

.title not updated on loading updated book (10/5/94)

Q: I am having trouble updating a book. If I change the .title to be a different date, the change is not reflected when I load the new book. What am I doing wrong?

A: The Book Reader application on Newton caches certain parts of the book, like the title and the current page. To get the engine to notice the title has changed, you have to change the .isbn of the book before you load the updated one.

If you are likely to update your book, you should use a dash in the .isbn to indicate version or date. An example would be:

```
    .title My Info Book of 7/94
    .isbn  IB-7-94:PIEDTS
```

No scripts in page .headers, use .running story (11/22/94)

Q: I am trying to add a script to the header of my pages, but it is never called. What's wrong?

A: You cannot use scripts in `.header` content. Use a `.running` content item instead. To turn off the default header, use the `NoTitle` flag in the .layout line.

For example:

```
    .layout Default 12 NoHeader
    .running story Centered
    Tap here to do something cool.
    .script viewClickScript
    ...  // do something cool
    .endScript
```

End of DTSQA

DRAWING

TABLE OF CONTENTS:

DRAWING..1
Introduction..1
Drawing Text on a Slanted Baseline (9/15/93)..1
Opaque Drawing and Clipping (9/15/93) 1
MakeText and Multiple Lines? (10/11/93)..1
80-dpi Resolution and PICT Resources (10/20/93).................................2
Font Style Creation (11/10/93) 2
LCD Contrast and Grey Texture Drawing (11/10/93)..............................2
Fonts in Picture Shapes (Scaling non-default fonts) (1/12/94)...........3
MakeText Doesn't Draw Text (1/12/94) 3
Destination Rectangles and ScaleShape (3/11/94)...................................4
Making CopyBits Use a Mask (6/3/94)..4
PtInPicture Bug (6/3/94) (obsoleted by NTK 1.0.1 platforms file)....4
Making CopyBits Scale its Output Bitmap(6/3/94) 4
The Newton Bitmap Format and Additional Bitmap Functions (6/9/94) 5
TextBox function for drawing formatted text (6/9/94)..........................8
No way to convert shapes to bitmaps (6/9/94) 9
A RoadMap for BitMap, PICT 1, and PICT 2 objects (7/27/94)...........11
PICT Swapping During Run-Time Operations (7/27/94)........................12
MakeRegion arguments and text, also hit testing. (9/26/94).............13

Introduction

This document addresses Newton Drawing 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.

Drawing Text on a Slanted Baseline (9/15/93)

Q: Is it possible in NewtonScript to draw text on a slanted baseline? I don't mean italics, but actually drawing a word at a 45 or 60 degree angle and so on. For example, can text be drawn along a line that goes from 10,10 to 90,90 (45 degrees)?

A: Like QuickDraw on the Macintosh, the drawing package on Newton supports no calls for rotating text. On the Macintosh, the workaround is to draw the text into a bitmap and then rotate the bits; unfortunately, this probably won't be possible on the Newton, so you need to design your product to draw text on a straight line. It's possible to design a font having characters that are pre-rotated to common angles (such as 45 degrees or 90 degrees) so that applications could just draw characters rather than actually having to rotate a bitmap. Currently, the Newton Toolkit cannot embed Macintosh fonts.

Opaque Drawing and Clipping (9/15/93)

Q: I'm opening a protoFloatNGo that is displayed on top of the screen. Behind the floater, I do a DrawText, which then overwrites the floater. I also notice that any views that have borders often can draw their borders over the floater. However, text in paragraphs don't seem to have this problem; that is, such text is always properly obscured by the floater.

What do I need to do to make the floater always opaque?

A: The preferred way to use the script drawing functions is to call Dirty, which causes the system to call your view's viewDrawScript script; the system sets clipping properly before calling your viewDrawScript script.

If you want to call the drawing functions at some other time, you can pass the name of your script to the function DoDrawing. This will set up clipping properly, call your script, and then restore clipping to its previous state. (This is really only useful for animation-style drawing -- your view still needs a viewDrawScript script in case it needs to be redrawn when the view on top of it is closed.)

MakeText and Multiple Lines? (10/11/93)

Q: Can MakeText be used to create more than one line of text at a time, or does the bound rectangle always clip excess text?

A: MakeText can only create text shapes with one single line.

80-dpi Resolution and PICT Resources (10/20/93)

Q: What is the best approach to creating PICT resources for use in a Newton application? I'm mostly worried about the 72-85-dpi screen resolution (Macintosh) versus 80-dpi resolution (Newton) in combination with development tools.

A: The way to scan an image correctly for 80-dpi resolution is as follows.

Scan the image at 72 dpi and 110% of the size you want to display. At 80 dpi on the Newton screen, you'll get the correct number of horizontal and vertical dots (that is, a full screen would be 240 pixels / 72dpi = 3.33", by 320 pixels / 72dpi = 4.44"). This approach has one immediate and very nice side effect: You can edit the images on a normal monitor with the usual paint tools, yet you'll still see the right thing on the Newton screen.

Font Style Creation (11/10/93)

Q: What is the recommended way of creating font styles, with regard to performance and size:

```
styles: [StrLen(mytext), tsSimpleFont10+tsBold];
```

or

```
styles: [StrLen(mytext), '{font: Espy; face: 1; size: 10}];
```

A: Integer codes, like tsSimpleFont10+tsBold, take up less space in your package (4 bytes). Unfortunately, integer codes are only useful for specifying the built-in fonts (Espy, NewYork, Geneva).

Font frames have the advantage of being human-readable. There are certain font frames already in the ROM, like ROM_fontsystem10underline. (See NTK's GlobalData specification file for a complete list.) If you use these pre-defined font frames, then space is not an issue. Finally, under the current implementation, font frames are processed slightly faster.

LCD Contrast and Grey Texture Drawing (11/10/93)

Q: An artist working with me did a wonderful job rendering a 3D look using several different grey textures. The problem is that when her image is displayed on the MessagePad everything on the screen dims. Is it possible that the image causes too much display current to maintain contrast?

A: What you're seeing is apparently a well-known problem with LCD displays in general, and there's not a lot you can do about it. It's especially aggravated by large areas of 50% gray (checkerboard) patterns, but the light gray and dark gray patterns also seem to cause some of it.

The user interface of the MessagePad deliberately avoided 50% grays as much as possible for this reason. If you know your application is going to display large gray areas, you could adjust the contrast yourself. There's a NewtonScript call, SetLCDContrast, to do just that. However, changing the contrast with no end user control is not considered a good user-interface practice.

Fonts in Picture Shapes (Scaling non-default fonts) (1/12/94)

Q: How can I scale a font other than the default font? A text shape in a PICT always seems to be the default font.

A: The important thing is to put the font specifier style frame "close enough" to the text shape so that it won't get overridden by another style frame.

There are several places where it might seem reasonable to put the style frame with the font specifier. (By the way, the font frame looks like this: {font: '{family: someFont, face: 0, size: 9 }})

The call to DrawShape takes a style argument, so it might go there:

```
  :DrawShape(myText, {font: '{family: someFont, face: 0, size: 9 }});
```

The call to MakePict has many opportunities (this is from the Font Scaling example):

```
   myText := MakePict([{penpattern: 0, (1)font: ...}, rect, {(2)font:
       ...}, txtshape], {(3)font: ...});
```

Setting the style frame in DrawShape won't work, because the style frame in the picture shape array (for penpattern) will override that one. Not having a font slot in a style frame means "use the default font".

Setting the style frame for the font in location (3) in the MakePict call has the same problem.

Setting the font in locations (1) or (2) in the MakePict call work fine. That font gets "encapsulated" into the PICT, and is safe.

If the {penpattern} frame was not present in the picture shape, any of the above places should suffice to set the font.

MakeText Doesn't Draw Text (1/12/94)

Q: I'm passing a valid string to MakeText but the text doesn't get drawn. Why not?

A: You may have specified a rectangle that's too small for the text. Try specifying a larger rectangle in your arguments to MakeText.

Note that shapes may not necessarily be complete copies of input data. Shapes made from bitmaps with MakeShape will keep the same reference to the bits as in the bitmap frame.

Destination Rectangles and ScaleShape (3/11/94)

Q: What is a valid destination rectangle for the 2nd argument to ScaleShape?

A: Like Macintosh QuickDraw, the destination rectangle must be at least 1 pixel wide and 1 pixel high. Each element of the bounds frame must have values that fit in 16 bits, -32768...32767. 0-width/height and negative width/height bounding boxes may appear to work in some cases, but are not supported.

Making CopyBits Use a Mask (6/3/94)

Q: How can I get `CopyBits` to use the mask of the bitmap passed to it?

A: You can pass modeMask as the argument to the mode parameter to *view*:`CopyBits(`*picture*, *x*, *y*, *mode*`)`.

PtInPicture Bug (6/3/94) (obsoleted by NTK 1.0.1 platforms file)

Q: I'm trying to use the function PtInPicture but it doesn't seem to work. The bitmap I'm passing to it is the icon slot of a clPictureView which has a mask.

When it didn't work as expected, we wrote some code to loop through the entire bitmap area and call PtInPicture for every point and draw a dot at each point where it returns true. The result was very strange. As it looped from left to right, there were vertical bands that looked like the left side of the mask, but then were blank (PtInPicture returned nil). These bands were of various widths but all looked like the left side of the mask.

Is there a bug in PtInPicture? Am I using it incorrectly?

A: Yes, there was a bug in the PtInPicture function. Use the PtInBitMap function included in the NTK 1.0.1 Platforms File.

Making CopyBits Scale its Output Bitmap(6/3/94)

Q: How can I cause `CopyBits` to scale the bitmap it draws?

A: `CopyBits` uses the `viewBounds` slot of the bitmap passed to it to scale the bitmap that it draws; hence, by changing the bounds of the bitmap passed to *view*:`CopyBits`, you can cause this method to scale the bitmap it draws. If you want to scale the output bitmap without changing the `viewBounds` of the original, modify the `viewBounds` slot of a clone of the original bitmap and pass the modified clone bitmap to the `CopyBits` method.

The Newton Bitmap Format and Additional Bitmap Functions (6/9/94)

Q: Is there a way to create bitmaps at run time?

A: Yes. However, the information in this answer describes only the current bitmap format. In the future different formats may be used, but the existing format will still be supported. You should not attempt to modify other application's bitmap structures or assume that all bitmap objects will be this format. You can use this info to create and manipulate your own bitmaps - preferably at build time.
You can use this information to create and manipulate your own bitmaps - preferably at build time. Don't assume too much about future Newton platform ROMs, they might have other bitmap specifications.

1.Terse Descripton of Newton 1.x ROM BitMap Format

```
{
 bounds: <bounds frame>,
 bit:    <raw bitmap data>,
 mask:   <raw bitmap data for mask - optional>
}

<raw bitmap data> - class 'bits
 Binary object
 bytes    data-type descr
 0-3      long      ignored
 4-5      word      #bytes per row of the bitmap data
 6-7      word      ignored
 8-15     bitmap rectangle - portion of bits to use--see IM I
 8-9      word      top
 10-11    word      left
 12-13    word      bottom
```

```
    14-15    word      right
    16-*     bits      pixel data, 1 for "on" pixel, 0 for off
```

The bitmap rectangle and bounds slot should should normally be in agreement as to the size of the bitmap, or other functions (for instance, hit testing) won't work properly. It can occasionally be useful to change the bounds slot (only) to get CopyBits for scaling purposes.

2. Utility Functions for Bitmap Manipulations
These functions are not ROM functions - nor will they be. They're designed for the Inspector, "Project Data" code and perhaps inclusion in your package.

```
   WriteBitMap(bitMap,mode)
     return value - unspecified
     bitMap - a bitMap frame
     mode - symbol - one of: 'draw    - draw the bitmap
                             'mask    - draw the mask
                             'test    - draw the "hot" area (wrt hit
                        testing)
   Writes a bm to the Inspector as ASCII - wow! Useful for debugging your
   code concerning small bitmaps.


   BitmapFromRawData(rawData, byteOffset, byteCount, pixelWidth,
   rowByteMultiple)
     return value    - bitmap frame {bits: <binary obj>, bounds:
                       <bounds                       frame>}
     rawData         - binary object containing pixel data
     byteOffset      - offset where pixel data begins in rawData
                       (nil -> default 0)
     byteCount       - number of bytes of pixel data in rawData
            (nil -> use Length(rawData) or StrLen(rawData))
     pixelWidth      - number of pixels per row in the bitmap
     rowByteMultiple - specifies byte alignment of rawData. I.e.
            rows of rawData are padded to be a multiple of
            rowByteMultiple bytes (nil -> default 1)
            (assume rawData is byte-aligned at the very least)

   Builds a bitMap up from raw data.

   */


   //constants for useful spots in raw-bitmap-data
   constant kBMRowBytesOffset  := 4;
   constant kBMTopOffset       := 8;
   constant kBMLeftOffset      := 10;
   constant kBMBotOffset       := 12;
   constant kBMRightOffset     := 14;
   constant kBMPixelOffset     := 16;


   func WriteBitMap(bitMap,mode)
   begin
    if mode = 'mask then
      bm := {bits: bitMap.mask, bounds: bitMap.bounds};
    else if mode = 'test then
      bm := bitMap;
    else //mode = 'draw
      bm := {bits: bitMap.bits, bounds: bitMap.bounds};

    local x,y;
    for y := 0 to bm.bounds.bottom - 1 do
    begin
```

```
  for x := 0 to bm.bounds.right - 1 do
    write(if call kPtInBitMapFunc with (x,y,bm) then "* " else ". ");
  write($\n);
 end
  true;
end;


func BitmapFromRawData(rawData, byteOffset, byteCount, pixelWidth,
rowByteMultiple)
begin
  if not rawData then return nil;

  if not byteOffset then byteOffset := 0;
  if not rowByteMultiple then rowByteMultiple  := 1;
  if not byteCount then
    begin
      byteCount := Length(rawData) - byteOffset;
        //try to be smart and ignore null
        // terminating bytes on strings
      if IsSubClass(ClassOf(rawData),'string) then
            byteCount := byteCount - 2;
    end;


  //calculate rawData's rowBytes
  local rbm8 := 8*rowByteMultiple;
  local rawBytesPerRow := rowByteMultiple*
                  ((pixelWidth + rbm8-1) DIV rbm8);

  //#bytes they gave us must be a multiple of rawBytePerRow
  if (byteCount MOD rawBytesPerRow) <> 0 then
    Throw('|evt.ex.msg|,"pixelWidth (" & pixelWidth & "), byteCount ("
&
byteCount & "), and rowByteMultiple (" & rowByteMultiple & ")
inconsistent");

  //calculate our rowBytes (always a multiple of 4)
  local rowBytes  := 4*((pixelWidth + 31) DIV 32);
//4*Ceiling(pixelWidth/32)

  local pixelHeight  := byteCount DIV rawBytesPerRow;
  local outputLength := kBMPixelOffset + pixelHeight*rowBytes;

  //create bitmap with header initialized to zero
  local myBits :=
SetClass(SetLength(Clone(
  "\u00000000000000000000000000000000"),outputLength),'bits);

  StuffWord(myBits, kBMRowBytesOffset,  rowBytes);
  StuffWord(myBits, kBMBotOffset,       pixelHeight);
  StuffWord(myBits, kBMRightOffset,     pixelWidth);

  if rawBytesPerRow = rowBytes then
    BinaryMunger(myBits,kBMPixelOffset,nil,rawData,
byteOffset,byteCount)
  else
    begin
      local rawIndex := byteOffset;
      for myBitsIndex := kBMPixelOffset to
         outputLength by rowBytes do
        begin

BinaryMunger(myBits,myBitsIndex,rawBytesPerRow,
rawData,rawIndex,rawBytesPerRow);
          rawIndex := rawIndex + rawBytesPerRow;
        end;
    end;

  return {bits: myBits, bounds: {top: 0, left: 0, bottom: pixelHeight,
right: pixelWidth}};
end;


/*
Inspector sample:

//very simple example - data created in ResEdit using the SICN editor
and then copying the data from the general (hex) resource editor. Use
Copy and Paste to get the data to NTK.

s :=
"\u08005400AA00AA009500551C4BE2400387058A818002400C4030204020802080";
WriteBitMap(BitmapFromRawData(s, nil, nil, 16, nil),'draw);

. . . . * . . . . . . . . . . .
. * . * . * . . . . . . . . . .
* . * . * . * . . . . . . . . .
* . * . * . * . . . . . . . . .
* . . * . * . * . . . . . . . .
. * . * . * . * . . . * * * . .
. * . . * . * * * * * . . . * .
. * . . . . . . . . . . . . * *
* . . . . * * * . . . . . * . *
* . . . * . * . * . . . . . . *
* . . . . . . . . . . . . . * .
. * . . . . . . . . . . * * . .
. * . . . . . . . . * * . . . .
. . * . . . . . . * . . . . . .
. . * . . . . . * . . . . . . .
. . * . . . . . * . . . . . . .
#2        NIL

 */
```

TextBox function for drawing formatted text (6/9/94)

Q: Is there a way to draw text without using shapes or creating a view for the drawing?

A: Beginning with MessagePad 110, there is a new routine called TextBox that allows you to draw text on the screen without creating views. You will generally use this call during a viewDrawScript.

You can check for the existence of this function as follows:

```
if functions.TextBox exists then
   TextBox(...)
else
   MyOwnTextBoxTypeOfFunction(...)
```

TextBox(<text>, <funky-font-frame>, <bounds>)
<text> string to draw
<funky-font-frame> see description below
<bounds> bounds frame to draw into

<funky-font-frame> This frame contains both a font specification and, optionally, a justification. The format is:

{font: <font-frame>, justification: <justify-symbol>}
<font-frame>: must be a font frame as per NPG (cannot be an integer specification)
<justify-symbol>: ['left|'center|'right] if slot not present or slot is nil, defaults to 'left

wraps <text> so that it fits in <bounds> and blasts it onto the screen, use this in a viewDrawScript or within a DoDrawing callback only.

IMPORTANT NOTE: the bounds are in local coordinates of the view that makes the TextBox call.

See the sample code NewTextFuncs for an example.

No way to convert shapes to bitmaps (6/9/94)

Q: I have a custom font which I'd like to print to PostScript printers. If I make text shapes using the font with MakeText, they still get turned into some printer resident font. Is there any way to render a shape as a bitmap on the Newton, so I can print the bitmap?

A: No. There are currently no routines in the Newton API for drawing to off-screen bitmaps or for turning shapes or groups of shapes into bitmaps.

The best you can do is create a number of bitmaps for the glyphs in the font, and write code to turn the strings into the bitmaps and print those.

Storage space of pictures and bitmaps (6/15/94)

Q: I have some bitmaps which are mostly black. Run length encoding would work really well for compressing them. Does GetPictAsBits store bitmaps in a compressed form?

A: If you store your package compressed (using the "optimize for space" setting), then all items in your package are compressed in small (approximately 1 K) pages, rather than object-by-object. For the 'picture editor' in NTK and the compile-time routine GetPictAsBits, at this time no additional compression is used.

You can use the NTK compile-time function GetNamedResource(..., 'picture) to get a Macintosh PICT resource which can be drawn on the Newton in clPictureViews. PICT resources are generally smaller than bitmap frames because every 'bitmap copy' opcode within the PICT resource contains compressed bitmap data.

Note that the PICT2 (color) picture format is not currently supported by the Newton drawing system.

Difference between view:LockScreen(nil) and RefreshViews (6/17/94)

Q: In the NPG, it states that sending a view the `:LockScreen(nil)` message forces an "immediate update". How is this different from calling `RefreshViews`?

A: When you post drawing commands (for example, DrawShape) the system normally renders the shape on the screen immediately. `:LockScreen(true)` provides a way to "batch up" the screen updates for multiple drawing calls. Sending `:LockScreen(nil)` "unplugs" the temporary block that has been placed on the screen updater, causing all the batched drawing changes to be rendered on the LCD.

`RefreshViews` tells the system to execute the commands needed to draw every view that has a dirty region. You can think of it as working at a level "above" the screen lock routines. When you send the message `Dirty`, it does not immediately cause the system to redraw the dirtied view, instead it adds the view to the dirty area for later redrawing.

You could lock the screen, dirty a view with a `SetValue`, call `RefreshViews` (and not see an update) draw a few shapes, and then, when you unlock the screen, the refreshes to the dirty regions and your shapes will all appear all at once.

Shapes are Totally Self-Contained (6/28/94)

Q: Are shape objects self-contained or do they assume something about the view in which it was created?

A: Shapes are self-contained in the sense that you could use them in other views, store shapes in soups, beam shapes using OutputFrame or Routing, or transmit shape objects over other comms systems from Newton to Newton. Shape objects are Newton internal objects that cannot be represented on other systems.

Note that shapes may not necessarily be complete copies of input data. Shapes made from bitmaps with MakeShape() will keep the same reference to the bits as in the bitmap frame.

Drawing Performance (6/30/94)

Q: Are there any tricks to make complex QuickDraw routines execute faster?

A: If you have a fairly static background picture, you can try using a pre-defined PICT resource instead of sequences of QuickDraw calls. Create the PICT in your favorite drawing package, and use the PICT as the background (clIconView). The graphics system also has a picture drawing function that enables you to create pictures that you can draw over and over again.

Also, if possible, convert any shapes to Picture Shapes using the MakePict() function; the system draws bitmaps much faster than complex shapes.

ScaleShape loses data when scaling primitive shapes, however this does not happen when scaling pictures, it's also faster when scaling pictures.

____If you want to improve hit testing of objects, use a larger view in combination with viewDrawScripts or viewClickScripts rather than using smaller views with individual hit testing scripts. This is especially true of a view that consists of regular smaller views.

A RoadMap for BitMap, PICT 1, and PICT 2 objects (7/27/94)

Q: What's the difference between a BitMap and a Picture, and what does PICT Format 1 versus PICT Format 2 mean? Is a Macintosh Picture object the same as a Newton Picture object?

A: A BitMap is a frame with bits, bounds, and optionally mask slots. The bits and mask slots contain binary bitmap data which the Newton uses to render the image. The bounds slot contains the dimensions of the object.

A Picture, on the other hand, is a binary object of class picture which is a superset of the old-style

Macintosh QuickDraw Format 1 PICT. It consists of primitive drawing commands rather than binary bitmap data (although it can serve as a container for Macintosh bitmap objects as well). The dimensions of the object are contained within the object itself as binary data.

Which format you use depends on your needs and the type of source images you have. If the picture is a true shape-oriented image (composed of lines, circles, boxes, and other QuickDraw drawing commands) then a PICT object will probably be the most compact. However, if the PICT is merely a container for a BitMap, then the BitMap format may be preferable. Regardless, BitMaps will generally render faster on the Newton than PICTs, but will also generally take up more space.

clPictureViews know how to draw both of these types of objects, so you merely need to choose a format and assign the object to the icon slot of the clPictureView.

On the other hand, the Picture object returned by MakePict is a native Newton Picture and can NOT be assigned to the icon slot of a clPictureView. Instead, use the viewDrawScript (of a plain clView for example) to render these objects. This Picture format is NOT compatible with the Macintosh Picture format (due to Unicode string issues) and can NOT be rendered on a Macintosh as-is. Some conversion function would have to be written to translate a Newton format Picture into a Macintosh PICT.

An important point to remember is that the Newton can only render Format 1 PICTs from the Macintosh. Most graphics design applications use Format 2 PICTs when copying a image to the clipboard or when saving the image to a disk file. Be sure you use an application that can place a Format 1 PICT on the clipboard for pasting into your resource file if you plan to image true PICT objects on the Newton. The following example shows how to determine the PICT version:

```
OpenResFile(HOME & "Photos Of Ralph.rsrc");

// Here we convert a PICT 1 or PICT 2 into a BitMap.
// This is what NTK's picture slot editor does.
DefConst('kPictureAsBitMap,
         GetPictAsBits("Ralph", nil));

// Here the picture is assumed to be in PICT 1 format.
// If it is not, the picture will not draw and you may
// throw exceptions when attempting to draw the object.
DefConst('kPictureAsPict,
         GetNamedResource("PICT", "Ralph", 'picture));

// Verify this is a Format 1 PICT object!
if ExtractWord('kPictureAsPict, 10) <> 0x1101 then
   print("WARNING: Ralph is not a Format 1 PICT resource!");

// This is one way to get the picture's bounds information.
// You can also extract it from the picture's own bounds
// rectangle at either compile time or run-time, by using
// ExtractWord to construct each slot of a bounds frame.
DefConst('kPictureAsPictBounds,
         PictBounds("Ralph", 0, 0));

CloseResFile();
```

The picture bounds information is obtained differently for each format. Here is an example viewSetupFormScript which sets the clPictureView's viewBounds slot to that of the image's at run-time:

```
// Assume one of the following is true:
// self.icon := kPictureAsBitMap;
// self.icon := kPictureAsPict;

   if ClassOf(icon) = 'picture then
```

```
         self.viewBounds := SetBounds(
                               ExtractWord(icon, 4),
                               ExtractWord(icon, 2),
                               ExtractWord(icon, 8),
                               ExtractWord(icon, 6))
      else if ClassOf(icon) = 'frame then
         self.viewBounds := icon.bounds;
```

Format 1 PICTs are created by rendering into an original black-and-white GrafPort on the Macintosh. Format 2 PICTs are created by Color QuickDraw when rendering into a CGrafPort on the Macintosh. You can easily determine the format of a PICT resource in ResEdit by examining bytes 10 and 11 as shown above. Format 1 will contain 0x1101 (version opcode followed by version number) and format 2 will contain 0x0011 (version opcode) followed by the actual version number byte 0x02 at offset 12.

See the Color QuickDraw chapter of Inside Macintosh for further information.

PICT Swapping During Run-Time Operations (7/27/94)

Q: How can I change the PICT used within a view during run time? Do I have to build a reference to this PICT in the Project Data file and store it into a constant?

A: To set a default picture for a clPictureView, simply use NTK's picture slot editor to set the icon slot of the clPictureView. You may select a PICT resource from any resource file that has been added to your project. The picture will be converted on the Macintosh from a type 1 or 2 PICT into a BitMap, and stored in your package at compile time. To change this picture at run time, you will need to keep a reference to each alternate picture or bitmap. This is easily done using DefConst at compile time in Project Data as follows:

```
OpenResFile(HOME & "Photos Of Ralph.rsrc");

// Here we convert a PICT 1 or PICT 2 into a BitMap.
// This is what NTK's picture slot editor does.
DefConst('kPictureAsBitMap,
         GetPictAsBits("Ralph", nil));

// Here the picture is assumed to be in PICT 1 format.
// If it is not, the picture will not draw and you may
// throw exceptions when attempting to draw the object.
DefConst('kPictureAsPict,
         GetNamedResource("PICT", "Ralph", 'picture));

// Verify this is a Format 1 PICT object!
if ExtractWord('kPictureAsPict, 10) <> 0x1101 then
   print("WARNING: Ralph is not a Format 1 PICT resource!");

// This is one way to get the picture's bounds information.
// You can also extract it from the picture's own bounds
// rectangle at either compile time or run-time, by using
// ExtractWord to construct each slot of a bounds frame.
DefConst('kPictureAsPictBounds,
         PictBounds("Ralph", 0, 0));

CloseResFile();
```

Notice there are two types of pictures: BitMaps (a frame with bits, bounds, and mask slots) and Format 1 PICTs (binary objects of class picture). clPictureViews know how to draw both of these types of objects, so you merely need to choose a format and use SetValue on the icon slot, as follows:

```
// SetValue(myView, 'icon, kPictureAsBitMap);
```

```
        SetValue(myView, 'icon, kPictureAsPict);
```

See the Q/A "A RoadMap for BitMap, PICT 1, and PICT 2 objects", above, for more information.

MakeRegion arguments and text, also hit testing. (9/26/94)

Q: My text shapes do not show up in the region produced by `MakeRegion`. How can I do hit testing on text shapes?

A: Regions only work for the simple polygon-like objects. The returned shape is a new polygon-like shape that encapsulates all the shapes passed, but can also be drawn in a single style.

Polygon-like shapes are those created with `MakeRect`, `MakeLine`, `MakeRoundRect`, `MakeOval`, `MakeWedge`, and `MakePolygon`. For shapes made with `MakePict`, the elements of the shape that are polygon-like will be used, those that are not will be ignored.

For shapes made with `MakeShape`, if the input is a points array or a bounds frame, the result is included; if the argument is a bitmap, the result will be ignored. Shapes made with `MakeShape` and a picture (`PICT 1`) argument are treated like those made with `MakePict`: the polygon-like elements of the PICT are included, the others are ignored.

Shapes made with `MakeText` will never be included in the region.

You don't need to make a regions in order to do hit testing. The function `HitShape` will take an array of shapes, just like the rest of the drawing functions. However, for text shapes, `HitShape` will return `TRUE` if the point is within the bounds rectangle specified in `MakeText`, and `NIL` otherwise. This means it may return `TRUE` for points that are nowhere near the actual text, and will return `NIL` for points in the descender area of the text. (See the Font Scaling sample code for more details on text in shapes.)

End of DTSQA

ERROR CODES

TABLE OF CONTENTS:

ERROR CODES....................1
Introduction....................1
Newton System Errors....................1
Newton Hardware Errors....................3
Newton Communication Errors....................7
NewtonScript Environment Errors....................17

Introduction

This document addresses Newton Application Design 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.

Newton System Errors

(-7000) Common Errors

`0`
Reason: noErr

`-7000`
Reason: No Memory
Error Message: "Newton does not have enough memory right now to do what you asked."

(-8000) Newton Application Errors

`-8001`
Error Message: "The battery on the inserted card must be replaced. For cards that must remain powered, do it while the card information box is displayed."

`-8002`

Error Message: "The battery on the inserted card is running low. For cards that must remain powered, replace it while the card information box is displayed."

-8003
Error Message: "There is nothing to undo."

-8004
Error Message: "This routing slip is already open."

-8005
Error Message: "Tap on the close box to hang up the modem after you have picked up the handset."

-8006
Error Message: "There is nothing to print."

-8007
Reason: Generic Newton Error
Discussion:
This is a generic scripting error, indicating that you have a generic Newton application error. Most likely an exception was thrown that was not handled, and the Newton environment did not find any specific error codes inside this exception. Maybe the exception frame didn't conform to any of the standard exception formats. This error should occur rarely.
Workaround:
Try to use breakpoints in order to isolate the problem. You might also take a look at the lastX, lastExMessage and lastExError globals in order to figure out what happened (these globals should only be used for debugging, not in code!).

-8008
Error Message: "A length in the styles slot is too short and has been extended."

-8009
Error Message: "Could not display text. A length in the read-only styles slot is too short."

-8010
Error Message: "A Communications card has been inserted."

-8011
Error Message: "The note has too many items. Start a new note by drawing a horizontal line across the notepad."

-8012
Error Message: "The note is too big. Start a new note by drawing a horizontal line across the notepad."

-8013
Error Message: "This note is too long. Start a new note by drawing a horizontal line across the notepad."

-8100
Error Message: "A blank note could not be created."

-8101
Error Message: "The item could not be moved."

-8102
Error Message: "The changes could not be saved."

-8103
Error Message: "Sorry, a problem has occurred."

-8104
Error Message: "There is a problem with the card."

-8105
Error Message: "The note could not be changed."

(-10000) Newton OS Errors

-10021
Reason: Communication Problem related to OS
Error Message: "A communications problem has occurred."
Discussion:
You might get this error message if you do a Connect using an ADSP endpoint, and the so-called NBP Lookup has a time out. The NBP Lookup might do this either because it was unable to find the ADSP socket, or because there were gateways that slowed down the lookup time. At this point there's no way to define the timeout value when doing NBP Lookups.
Workaround:
Try to do another Connect after a while. The original Connect should return a non-nil value; then you know that the initial Connect failed. Also, check to see that your hard-coded address for the ADSP socket is valid.

(-10400) Package Errors

-10401
Reason: Bad Package

-10402
Reason: Package Already Exists
Error Message: "The same version of this application is already installed."

-10403
Reason: Bad Package Version

-10404
Reason: Unexpected End of Package

-10405
Reason: Unexpected End of Package Part

-10406
Reason: Part Type Already Registered

-10407
Reason: Part Type Not Registered

-10408
Reason: No Such Package

-10409
Reason: Newer Package Already Exists
Error Message: "A newer version of this application is already installed."

-10410

Reason:	Older Package Already Exists
Error Message:	"An older version of this application is already installed."

Newton Hardware Errors

(- 10500) PCMCIA Card Errors

-10501

Reason:	Unrecognized Card
Error Message:	"Unrecognized type of card"

-10502

Reason:	Card Not Ready

-10503

Reason:	Card, Bad Power

-10504

Reason:	Unexpected Card Error

-10505

Reason:	Card Reset

-10506

Reason:	Card Not Initialized

-10507

Reason:	Card Service Not Installed

-10508

Reason:	Card Service Not Suspended

-10509

Reason:	Card Service Not Resumed

-10510

Reason:	No Usable Configurations

-10511

Reason:	Format Failed
Error Message:	"This memory card could not be formatted."

-10512

Reason:	Unformatted and Write Protected Card
Error Message:	"This memory card could not be formatted. (This memory card is write protected.)"

-10520

Reason:	Parser Proc Pointer
-10521	
Reason:	Tuple is Unknown
-10522	
Reason	SubTuple is Unknown
-10523	
Reason:	Tuple Order
-10524	
Reason:	Tuple Size
-10525	
Reason:	Tuple No Link and Link
-10526	
Reason:	Tuple Link and No Link
-10527	
Reason:	Tuple Link Target
-10528	
Reason:	Tuple Version 1
-10529	
Reason:	Tuple Version 2
-10530	
Reason:	Tuple Jedec
-10531	
Reason:	CheckSum
-10532	
Reason:	No CIS (first tuple is 0xFF)
-10533	
Reason:	Blank CIS (all 0's for 16 bytes)
-10534	
Reason:	Bad CIS
-10535	
Reason:	Link Target C

(- 10550) Flash Card Errors

-10551	
Reason:	Flash Card Busy
-10552	
Reason:	Flash Card Not Erasing
-10553	
Reason:	Flash Card Not Erase Suspend

-10554	
Reason:	Flash Card Suspend Erase Error
-10555	
Reason:	Flash Card Erase Failed
-10556	
Reason:	Flash Card Write Failed
-10557	
Reason:	Flash Card Vpp Low
-10558	
Reason:	Flash Card Error in Sleep
-10559	
Reason:	Flash Card Error Not Enough Power

(-10600) Card Store Errors

-10600	
Reason:	Object Overrun, attempt to read or write outside object bounds
-10601	
Reason:	Bad Buffer Pointer
-10602	
Reason:	Bad Card Access
-10603	
Reason:	Bad Storage Type
-10604	
Reason:	Store Not Found
-10605	
Reason:	Write Protected, user has protected the store
Error Message:	"This memory card is write protected"
-10606	
Reason:	Object Not Found (non-existent object) FLASH store error when a low level OS object cannot be found on the card.
-10607	
Reason:	Block Full (flash card internal condition)
-10608	
Reason:	Not Virting (flash card internal condition)
-10609	
Reason:	Write Error (one or more bits failed to assert)
-10610	
Reason:	No More Objects
-10611	

Reason: Erase In Progress (flash card internal condition)

-10612
Reason: Card is Full (no more blocks left on the card)

-10613
Reason: No More Blocks (no more blocks left in search, flash card internal condition)

-10614
Reason: Log Full (flash log is full)

-10615
Reason: Needs Format (card needs to be formatted)

-10616
Reason: Bad PSSID (unknown PSSID)

-10617
Reason: Store Full
Error Message: "The internal memory or storage card is full"

-10618
Reason: Bad Battery (missing or low battery on SRAM card)
Error Message: "The battery on this memory card must be replaced."

-10619
Reason: Not In Transaction (attempt to modify store without a transaction in effect)

-10620
Reason: Transaction Aborted

-10621
Reason: WPBut Needs Repair (needs recovery, but card is write protected)

-10622
Reason: Object Too Big (too big for store)

Newton Communication Errors

(-12000) Generic AppleTalk Errors

-12001
Reason: Buffer Bad, buffer too small, other buffer reasons

-12002
Reason: Event Pending (state rather than error)

-12003
Reason: Cancelled

-12004
Reason: Cancel Failed (attempt to cancel and async message failed)

-12005
Reason: No Cancel Handler

-12006
Reason: Unknown Message Receiver

-12007
Reason: Can't Create AppleTalk Port

-12008
Reason: Can't Create AppleTalk Task

-12009
Reason: Not Implemented

-12010
Reason: Data Length

-12011
Reason: No Such Object To Open

-12012
Reason: Not Opened

-12013
Reason: AppleTalk Not Open

-12014
Reason: AppleTalk Already Open

-12015
Reason: Duration Too Small

-12016
Reason: Duration Too Large

(-12100) LAP Protocol

-12100
Reason: LAP Read Link Failed

-12101
Reason: LAP All Protocols In Use

-12102
Reason: No Protocol Handler

-12103
Reason: No Such Command

-12104
Reason: Bad Link

(-12200) DDP Protocol

`-12200`
Reason: No Such DDP Command

`-12201`
Reason: Invalid Socket

`-12202`
Reason: Not in Static Socket Range

`-12203`
Reason: Not in Dynamic Socket Range

`-12204`
Reason: Socket Already Open

`-12205`
Reason: Socket Not Open

`-12206`
Reason: Special Internal Socket

`-12207`
Reason: Socket In Use

`-12208`
Reason: Unknown LAP Type

`-12209`
Reason: DDP Back Check Sum

`-12210`
Reason: Bad Packet Size

`-12211`
Reason: No Socket Listener

`-12212`
Reason: No Such Known Protocol Type

`-12213`
Reason: External Client Timed Out

(- 12300) NBP Protocol

`-12300`
Reason: Bad Form

`-12301`
Reason: Name Already Registered

`-12302`
Reason: Too Many Names

`-12303`

Reason:	Name Not Registered
-12304	
Reason:	Too Many Names Requested
-12305	
Reason:	Too Many Lookups Pending
-12306	
Reason:	Not NBP Packet DDP Type
-12307	
Reason:	Unknown NBP Function
-12308	
Reason:	Unknown NBP Lookup Reply
-12309	
Reason:	Too Many Tuples in Lookup Request
-12310	
Reason:	NBP Lookup in Use
-12311	
Reason:	NBP Index Out of Range
-12312	
Reason:	NBP Lookup Aborted
-12313	
Reason:	No Such Command
-12314	
Reason:	No Names Found

(-12400) AEP Protocol

-12400	
Reason:	No Such Command
-12401	
Reason:	Not Echo Packet DDP Type
-12402	
Reason:	AEP Packet Size Zero
-12403	
Reason:	AEP Function Not Request

(-12500) RTMP Protocol

-12500	
Reason:	No Such Command

-12502	
Reason:	Packet Size Zero
-12503	
Reason:	RTMP Routed
-12504	
Reason:	RTMP Address Unresolved
-12505	
Reason:	RTMP No Router

(-12600) ATP Protocol

-12600	
Reason:	No Such Command
-12601	
Reason:	No ATP Packet DDP Type
-12602	
Reason:	Unknown ATP Function
-12603	
Reason:	ATP Request Data Zero Length
-12604	
Reason:	Expected Responses Out of Range
-12605	
Reason:	Response Buffer Too Small
-12606	
Reason:	ATP Retry Duration Too Small
-12607	
Reason:	ATP Transaction Timed Out
-12608	
Reason:	Responding Socket Already Open
-12609	
Reason:	Responding Socket Not Open
-12610	
Reason:	Response Packet Bad Length
-12611	
Reason:	Bad Number of Response Packets
-12612	
Reason:	Already Request On AutoRequest or Socket

(-12700) PAP Protocol

-12700	
Reason:	No Such Command
-12701	
Reason:	Unexpected Connection ID
-12702	
Reason:	Invalid Connection ID (trying to use an invalid ID)
-12703	
Reason:	Invalid Responder Socket (trying to use an invalid socket)
-12704	
Reason:	Unexpected Function
-12705	
Reason:	Printer Busy
-12706	
Reason:	Unexpected Conn Open Result
-12707	
Reason:	Bad Flow Quantum Requested
-12708	
Reason:	Connection Timed Out
-12709	
Reason:	EOF Sent (internal condition)
-12710	
Reason:	PAP Flushed (internal condition)
-12711	
Reason:	Printer Terminated Connection
-12712	
Reason:	Printer Not Found
-12713	
Reason:	No Status Available
-12714	
Reason:	No Data Available
-12715	
Reason:	Buffer Passed is Too Small
-12716	
Reason:	Put Data Timed Out

(-12800) ZIP Protocol

-12800
Reason: No Zones

(-12900) ADSP Protocol

-12900
Reason: Too Many ADSP Connections

-12901
Reason: ADSP Invalid Mode

-12902
Reason: ADSP Bad Packet Size

-12903
Reason: ADSP Bad Control Type

(-16000) Communication Tool Errors

-16005
Error Message: "Cancelled."

-16009
Reason: Phone Connection Died
Error Message: "The phone connection was cut off."
Discussion:
This error message was eventually issued from a modem endpoint connect call. Workaround: Try to catch these cases using exception handling; perhaps also try another connect.
This error message was issued from a modem endpoint connect call. Most likely the connection was terminated by an outside case.

-16013
Error Message: "Cancelled."

-16018
Reason: Connection end already bound

-16021
Reason: No Phone Number Provided
Error Message: "No phone number was provided."
Discussion:
The address frame to the modem endpoint connect call didn't contain a valid phone number.
Workaround:
Make sure that the address frame has a working phone number. See documentation about how a valid address frame looks.

(-18000) Serial Tools Errors

`-18003`
Error Message: "Bad connection."

Discussion:
This error is returned if the comms tool encounters any of the following problems with the comms hardware and buffers (async serial tool):
• Framing error (Link layer)
• Parity error (Link layer)
• Overrun error (overrun of comms buffers)

Workaround:
Consult the Communications chapter for more information about how to avoid such cases.

(-20000) MNP Errors

-20001
Reason: Connection parameter negotiation failure

-20002
Reason: Connect request (acceptor mode) timed out

-20003
Reason: Not connected
Discussion:
The MNP tool makes the conclusion that there's no MNP session happening. Most likely there has been a drop-out or a sudden disconnection between the MNP nodes. In some cases the MNP tool also makes the decision that bad frames indicate no connection.

Workaround:
Make sure you have a good phone connection. Analog cellular connections, for instance, tend to be very noisy. Check out any issues related to the MNP connection: are the two sides talking the same protocol? MNP10 against MNP5 will not work properly.

-20004:
Reason: Request aborted by disconnect request.

-20005
Reason: Link attention service is not enabled.

-20006
Reason: Connect (initiator mode) request retry limit reached.

-20007
Reason: Command already in progress.

-20008
Reason: Connection already established.

-20009
Reason: Connection failed due to incompatible protocol levels.

-20010
Reason: Connection handshake failed.

-20011

Reason: Memory for MNP not allocated. (Option was passed which indicated not to allocate MNP memory.)

(-22000) FAX Errors

-22001
Error Message: "The fax couldn't be sent."

-22002
Error Message: "The fax couldn't be sent."

-22003
Error Message: "The fax couldn't be sent."

-22004
Error Message: "The fax couldn't be sent."

-22005
Error Message: "The fax machine had a problem printing some pages."

-22006
Error Message: "The fax couldn't be sent."

-22007
Error Message: "The fax couldn't be sent."

(-24000) Modem Errors

-24000
Reason: No Modem Connected
Error Message: "No modem is connected."
Discussion:
A modem endpoint connect call returned with this error. Most likely the modem was not connected properly.
Workaround:
Check to make sure you have the right cable, that the connections are secure, that the modem is working, that it has batteries and so on.

-24001
Reason: No Dial Tone
Error Message: "There is no dial tone."
Discussion:
The Modem endpoint connect returned with this error . The modem is unable to dial. Problem with either the phone cable or the phone exchange.
Workaround:
Check the phone cable; try making normal phone calls using the same line.

-24002
Reason: No Answer
Error Message: "There is no answer."
Discussion:
The Modem endpoint connect returned with this error. Most likely the modem is unable to detect carrier (CD, signal 109). Some reasons for this might be: that the modem is missing on the other

side; that a person is answering instead of a modem; that the modem at the other end does not have power.

Workaround:

Wait a while and try again. If this is impossible, signal to the user that the modem can't connect.

`-24003`

Reason: Busy Tone at the Other End

Error Message: "The phone number is busy."

Discussion:

The modem endpoint connect call returned with this error. Most likely, either the phone at the other end is in use, or the phone number is wrong.

Workaround:

Provide a way to re-connect later. If the problem persists, check the validity of the assigned phone number.

`-24004`

Error Message: "There is no answer."

`-24005`

Error Message: "Your modem isn't responding properly."

`-24007`

Error Message: "Your modem isn't responding properly."

(-28000) Macintosh Connection Kit Errors

(-38000) Sharp IR Errors

`-38001`

Error Message: "No response"

`-38002`

Error Message: "Cancelled"

`-38006`

Error Message: "Bad connection"

`-38008`

Error Message: "Beaming error"

(-40100) Online Service Errors

`-40102`

Error Message: "Lost connection to host"

-40103
Error Message: "Lost connection to host"

-40104
Error Message: "The host is not responding."

-40105
Error Message: "There is a problem reading from the host."

-40106
Error Message: "Failed to connect to local access number."

(-44000) Printing Errors

-44000
Error Message: "Printer problem."

-44001
Error Message: "Newton is unable to print."

-44002
Error Message: "No printer is connected."

-44003
Error Message: "Printer busy."

-44004
Error Message: "Stopped"

-44005
Error Message: "Lost contact with the printer."

-44100
Error Message: "Insert the next sheet of paper."

-44101
Error Message: "Dial the phone number now."

-44102
Error Message: "There is no paper tray."

-44103
Error Message: "The wrong paper tray is attached."

-44104
Error Message: "The printer has no paper."

-44105
Error Message: "The printer has no ink."

-44106
Error Message: "The printer is jammed."

-44107
Error Message: "The printer door is open."

-44108

Error Message: "The printer is off-line."

(-46000) PC Connection Kit Errors

NewtonScript Environment Errors

(-48000) Store and Soup Errors

-48001
Error Message: "This is not a data storage card."

-48002
Reason: Store format is too old to understand
Error Message: "This card was formatted with an older version of the system and cannot be used."

-48003
Reason: Store format is too new to understand
Error Message: "This card was formatted with a newer version of the system and cannot be used."

-48004
Reason: Store is corrupted, can't recover

-48004
Error Message: "This card has been corrupted and cannot be used."

-48005
Reason: Single object is corrupted, can't recover

-48009
Reason: Not a soup entry

-48010
Reason: Tried to remove a store that wasn't registered

-48011
Reason: Soup index has an unknown type

-48012
Reason: Soup index has an unknown key structure

-48013

Reason:	Soup index does not exist

-48014

Reason:	A soup with this name already exists

-48015

Reason:	Tried to CopyEntries to a union soup

-48016

Reason:	Soup is invalid (probably from a removed store)

-48017

Reason:	Soup is invalid (probably from a removed store)

-48018

Reason:	Entry is invalid (probably from a removed store)

-48019

Reason:	Key does not have the type specified in the index

-48020

Reason:	Store is in ROM

-48020
Error Message: "This memory card cannot be changed"

-48021

Reason:	Soup already has an index with this path

-48022

Reason:	Internal error–something unexpected happened

-48023

Reason:	Tried to call RemoveIndex on the _uniqueID index

-48024

Reason:	Query type missing or unknown

-48025

Reason:	Discovered index inconsistency

(-48200) Object System Errors

-48200

Reason:	Expected a frame, array, or binary object

Discussion:

The easiest way to understand what is happening is to cause this error to occur by typing one of the following statements in the Inspector window:

```
   true.sillySlot
or
   false.sillySlot
```

These examples store values in slots that are not real references; specifically, the examples above try to dereference something that is not a pointer. The system expects either a frame, an array or a binary object and we pass an immediate value.

Another situation that causes -48200 errors is to define an immediate slot that is actually an underlying view/object method. For example, if you define a base view slot called Dirty, protoLabelInputLine and other clView-based protos start to behave strangely and you begin getting -48200 errors. This is because the system tries to send a message to an immediate slot, and, because of inheritance rules, it finds this slot before the clView slot.

Workaround:
Make sure that the values used are real references, not object methods, immediate values or any other types not documented.

`-48202`
Reason: Empty path

`-48203`
Reason: Invalid segment in path expression

`-48204`
Reason: Path failed

`-48205`
Reason: Index out of bounds (string or array)

`-48206`
Reason: Source and destination must be different objects

`-48207`
Reason: Long out of range

`-48210`
Reason: Bad arguments

`-48211`
Reason: String too big

`-48214`
Reason: Object is read-only

`-48215`
Reason: Functionality is unimplemented

`-48216`
Error Message: "The application does not have enough memory right now to do what you asked."

(-48400) Bad Type Errors

`-48400`
Reason: Expected a frame

`-48401`
Reason: Expected an array

`-48402`
Reason: Expected a string

-48403
Reason: Expected a pointer

-48404
Reason: Expected a number

-48405
Reason: Expected a real

-48406
Reason: Expected an integer
Discussion:
One case when this error is reported is when the underlying toolbox assumes that a slot has an integer value, or that a function/method parameter is an integer. A typical case, for instance, is when slots are nil instead of having values.

For instance if the viewJustify slot is nil, and you call the Hilite method on this function, you will get -48406 errors.

Workaround:
Make sure important slots have integer values, or that function/method parameters are indeed integers.

-48407
Reason: Expected a character

-48408
Reason: Expected a binary object

-48409
Reason: Expected a path expression (or a symbol or int)

-48410
Reason: Expected a symbol

-48411
Reason: Expected a symbol

-48412
Reason: Expected a frame or an array

-48413
Reason: Expected an array or Nil

-48414
Reason: Expected a string or Nil

-48415
Reason: Expected a binary object or Nil

-48416
Reason: Unexpected frame

-48417
Reason: Unexpected binary object

-48418
Reason: Unexpected immediate

(-48800) Interpreter Errors

`-48800`
Reason: Not in a break loop

`-48802`
Reason: Too many args for a CFunction

`-48803`
Reason: Wrong number of arguments

`-48804`
Reason: FOR loop BY expression has value zero

`-48806`
Reason: No current exception

`-48807`
Reason: Undefined variable

`-48808`
Reason: Undefined global function
Discussion:
This error is typical if you misspell a toolbox function — for instance StringReplace instead of StrReplace.

Workaround:
Make sure that the functions you call exist (consult documentation).

`-48809`
Reason: Undefined method

`-48810`
Reason: No _proto for inherited send

`-48811`
Reason: Tried to access slot in Nil context

`-48814`
Reason: Local variables and FOR/WITH loops not allowed at top level.

End of DTSQA

Hardware and Operating System

Hardware and Operating System..........1
Introduction..........1
Serial Port Hardware Specs (6/15/94)..........1
Serial Cable Specs (8/9/94)..........2
IR Hardware Info (9/6/94)..........3
NEW
How much power can a PCMCIA card draw? (3/31/95)..........3

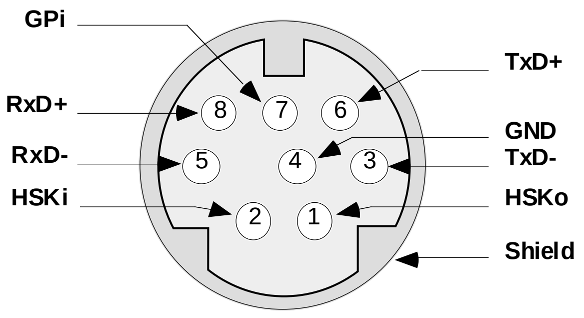
Introduction

This document addresses Newton Hardware and OS 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.

Serial Port Hardware Specs (6/15/94)

Q: What are the hardware specifications for the Newton serial port?

A: The Newton serial port is an EIA standard RS-422 port with the following pinout (as viewed looking at the female Mini-DIN-8 socket on the side of the Newton):



Pin **1**	HSKo	/DTR
Pin **2**	HSKi	/CTS
Pin **3**	TxD-	/TD
Pin **4**	GND	Signal ground connected to both logic and chassis ground.
Pin **5**	RxD-	/RD
Pin **6**	TxD+	(see below)
Pin **7**	GPi	General purpose input received at SCC's DCD pin.
Pin **8**	RxD+	(see below)

All inputs are:	Vih = 0.2V	Vil = -0.2V	Ri = 12k ohms
All outputs are:	Voh = 3.6V	Vol = -3.6V	Rl = 450 ohms

Pins 3 & 6 tri-state when SCC's /RTS is not asserted.

The EIA RS-422 standard modulates its data signal against an inverted (negative) copy of the same signal on another wire (twisted pairs 3/6 & 5/8 above). This differential signal is compatable with older RS-232 standards by converting to EIA standard RS-423, which involves grounding the positive side of the RS-422 receiver, and leaving the positive side of the RS-422 transmitter unconnected. Doing so, however, limits the usable cable distance to approximately 50 feet, and is somewhat less reliable.

Serial Cable Specs (8/9/94)

Q: I want to make my own serial cable. Which wires and which connector pins do I use?

A: To create a hardware flow control capable cable for Mac-to-Newton or Newton-to-Newton communications (also called a "null-modem" cable) all you need are two mini-din-8 connectors and seven wires connected as follows:

```
Ground (4) -> Ground (4)  (also connect to connectors' shrouds)
Transmit+ (6) -> Receive+ (8)
Transmit- (3) -> Receive- (5)
Receive+ (8) -> Transmit+ (6)
Receive- (5) -> Transmit- (3)
Data Term Ready (1) -> Clear To Send (2)
Clear To Send (2) -> Data Term Ready (1)
```

You should use twisted pairs for 6/3, 8/5, and 1/2, to improve signal quality and reduce attenuation, especially in long cables. You can use side-by-side pairs, as in telephone hookup cable, for short cable runs.

Remember that because RS-422 uses a differential signal for transmit and receive, you always need two transmit and two receive pairs, and a break of either wire will cause communications in that direction to fail. The advantage, however, is significantly longer and more reliable cable runs than RS-232.

If you don't use hardware flow control, you can eliminate the 1/2 pair, but that's not recommended unless you know this cable will be used only in software flow control situations.

Q: What's the pin mapping on the Newton-to-PC (DIN-to-DB9) cable?

A: Here it is:

Note that the pin numbers shown are as defined above.

PC (DB9)
Newton (DIN)
1
1
2
3
3
5
4
7,2

5
4,8
6
1
7
N/C
8
N/C
9
N/C

N/C=not connected.

IR Hardware Info (9/6/94)

Q: How does the Newton send "Remote Control" codes?

A: This information is hardware dependent, and is only valid for the Message Pad, Message Pad 100, and Message Pad 110 products.

The IR transmitter/reciever is a Sharp IR Data Communication Unit connected to the second channel of a built-in SCC. When in "Remote Control" mode, the SCC is not used. Instead, a carrier frequency of 38KHz is transmitted, and the CPU toggles a register to generate the data pattern.

NEW: How much power can a PCMCIA card draw? (3/31/95)

Q: How much power can I draw through the PCMCIA slot?

A: The current rating depends on which Newton you are using and the type of batteries in use. Alkaline batteries provide less current than NiCad due to higher internal resistance. There is also a 'semi' artifical limit in the ROM. Currently any card who's CIS indicates more than 200 mA current draw will be rejected by the CardHandler. Other than that, here's the run down by hardware:

MessagePad 100	50 mA
MessagePad 110	~160mA
MessagePad 120	~300mA

End of DTSQA

INTELLIGENT ASSIST

INTELLIGENT ASSIST 1
Introduction..1
Phone Dialer (11/11/93) 1
Conflict Resolution (12/13/93)...1
Overriding Built-In Words (12/13/93)...2
System-Supplied Target Templates (12/13/93)...2
Dynamically Changing Tasks (12/21/93)..3
Results of matching with who_obj (3/10/94)...3
who_obj inconsistencies, conflicts with company, etc. (6/9/94)........3

Introduction

This document addresses Newton Intelligent Assist 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.

Phone Dialer (11/11/93)

Q: Is there any way I can bring up the 'phone dialer' window (the one that appears when a user taps Assist and writes 'call 555-5555') without going through all the procedures necessary for registering templates with the assist button?

A: Try using the ParseUtter routine. The call 'ParseUtter("call 555-5555")' will bring up the slip you want. If you have the dialing options set to area code 408, calling the routine 'ParseUtter("call jonathan 4085551234")', should bring up the number '5551234' in the calling dialog, with 'jonathan' displayed as the first choice name, or the first match on 'jonathan' found in the Names soup.

Conflict Resolution (12/13/93)

Q: How can I register two actions that share the same verb? If I do it the straightforward way, nothing gets interpreted.

A: The problem is that the "shortcutting" that IA does when it completely matches an action is starting too soon. You can stop this from happening by registering a task template like this:
ds: changed "kicking in" to "starting", in the second line

```
tetrisTaskTemplate := {
      isa: 'task_template,
      primary_act: tetrisActionTemplate,
      preconditions: ['myTarget, 'myAction, 'generic_action],
      signature: [tetrisTemplate, tetrisActionTemplate,
'dyna_user_action],
      postparse: tetrisMsg,
      taskslip: nil,
};
```

where the tetrisActionTemplate has the lexicon "play" and the tetrisTemplate has the lexicon "tetris". Note that 'dyna_user_action has no lexicon, so it won't match any words, ever. This prevents the IA shortcutter from taking over, and normal conflict resolution can happen. That is, if another task registered "play" and "othello" along with 'dyna_user_action, the phrase "play tetris" would go to the tetris task, and "play othello" would go to the othello task.

See IA Tour sample code example for the full source code; this is also mentioned (but not explained) in the 1.0 Newton Programmer's Guide p11-29.

Overriding Built-In Words (12/13/93)

Q: I want to register an action that uses the word "lunch", as in "eat lunch". But I get errors when I try to add and use a lexicon with "lunch" in it.

A: Because "lunch" has already been added in the lexicon of one of the built-in IA templates, the IA word-matcher will always think "lunch" is one of those objects. All the system target templates inherit from 'user_obj, so if you add this to your task template, your action can get called for any built-in word. e.g.:

```
    eatTaskTemplate := {
        isa: 'task_template,
        primary_act: eatActionTemplate,
        preconditions: ['myTarget, 'myAction, 'generic_obj],
        signature: [eatTemplate, eatActionTemplate, 'user_obj],
        postparse: holdTheMsg,
        taskslip:  nil,
    };
```

The eatActionTemplate has the lexicon "eat", and the eatTemplate has the lexicon "breakfast", "brunch", "lunch", "dinner", "supper". The word "lunch" will normally match against both the built in target 'user_obj and the specific object in eatTemplate. However, the 'myTarget slot will not reliably be created in the IA result frame.

System-Supplied Target Templates (12/13/93)

Q: The section "Structure Of The Target Template" in the IA chapter of NPG mentions that the formats for who_obj, where_obj, what_obj, when_obj, phone_number and parsed_number are described in the *NewtonScript Programmer's Reference*, but I can't find this info anywhere.

A: After writing the IA chapter, we decided that this information really belonged in NPG with the rest of the IA material, but we neglected to update the reference in the IA chapter. The missing information is not really as consequential as the reference to it might make it seem --but here it is:

The `who_obj`, `where_obj`, `what_obj`, `when_obj`, and `parsed_number` target templates are frames having the value `user_obj` in the `ISA` slot. The `phone_number` template has been renamed as the `phone` template, and has the value `where_obj` in its `ISA` slot.

Dynamically Changing Tasks (12/21/93)

Q: How do I dynamically change my task template?

A: To "dynamically" change your template, unregister the old task template with UnregTaskTemplate, then register a new (modified) one with RegTaskTemplate. When you call RegTaskTemplate, it makes a complete copy of your task template, so you can actually just modify the lexicon array from the original target template and not worry about "re-setting" the task template. Just unregister the old one and register the new one.

Results of matching with who_obj (3/10/94)

Q: I use a who_obj in my signature array, with the symbol 'who in the matching slot in the preconditions array. When my action is matched, the 'who slot contains the symbol 'person. How do I get the name?

A: You must look through the results array until you find the slot that matched the who_obj, using the IsA() test. Once found, the corresponding slot in the phrases array will contain the string that was matched. Use this string as the key for a word query on the ROM_CardFileSoupName soup. This will give all matching names. (You may be tempted to use the global function SmartCFQuery. Do not. It is undocumented and may change on future platforms. Using a word query on the names soup is just as fast as using this function.)

'who_obj inconsistencies, conflicts with company, etc. (6/9/94)

Q: If I use 'who_obj for my target template, it works fine most of the time. But if "Bob" appears in both the person.firstName and company slots of a Names entry, then the slot from my preconditions for the name is not created, and "Bob"is not present in the noisewords array either. How can I find "Bob"?

A: This is another conflict resolution problem with IA. The system considers strings found in the company slot in names to be instances of places, which is a subclass of 'what_obj and not 'who_obj.

Both 'what_obj and 'who_obj are subclasses of 'user_obj, so you might think you could replace the 'who_obj in your signature with 'user_obj. Unfortunately, there are many other things that are subclasses of 'user_obj, and if you are just looking for names, you probably do not want to have companies, phone numbers, or other strings treated as names.

The most reliable solution is to parse the string yourself and do a words query on the card file soup to get the soup entry with the matching name.

End of DTSQA

LOCALIZATION

TABLE OF CONTENTS:

LOCALIZATION 1
Introduction...1
kincludeEverything should be kincludeAllElements (9/15/94)..........1

Introduction

This document addresses Newton Localization 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.

kincludeEverything should be kincludeAllElements (9/15/94)

Q: I tried to use kIncludeEverything in a call to LongDateStr, but I get an error.

A: The constant kIncludeEverything docmented on page 15-21 of version 1.0 of the Newton Programmers Guide is incorrect. It should be kIncludeAllElements.

End of DTSQA

NEWTON TOOLKIT

Table of Contents

NEWTON TOOLKIT 1
Serial Cable for NTK (9/15/93) (Obsoleted by 1.0 Newton Toolkit User's Guide p1-5) 1
Adding PICT Masks With NTK (9/15/93) (Obsoleted by 1.0 Newton Toolkit User's Guide p4-6, p4-32, pC-14)....................................1
Problems Changing Subview Links (9/15/93) (Obsoleted by NTK 1.0b7 or later.) 1
How do I produce packages for Windows/DOS from a Mac? (9/15/93) 2
Problems Downloading Packages (9/15/93)....................................2
How do I change the default size of the Inspector Window? (9/15/93) 3
Inspector Use (10/13/93)....................................3
Font Support? (11/10/93) 3
NTK Copy and viewFlags Problems (11/15/93) (Obsoleted with NTK 1.0 Final) 4
Linked Subviews and View Positioning (12/5/93)....................................4
Project Data File Problems, 1.0b7 (12/18/93) (Obsoleted by NTK 1.0 final) 4
NTK, Picture Slots and ROM PICTs (12/19/93)....................................4
AutoClose flag (1/4/94) (Obsoleted by 1.0 Newton Toolkit User's Guide p4-6) 5
Recognition Problems with the Inspector Window Open (3/8/94)....5
LocalTalk connections between Newton and NTK (3/8/94)..................5
Frames heaps, when, why, how to increase the Size (3/8/94) (Obsoleted by 1.0 Newton Toolkit User's Guide p4-11)....................................6
Locale Used at Compile Time (3/8/94) 6
Compression setting is per package , not per part (4/7/94).................7
Accessing Views Between Layout Windows (6/7/94)............................7
Dangers of StrCompare, StrEqual at Compile Time (6/9/94)................8
Dangers of Using _proto as an Unbound Variable (6/9/94)..................9
Package Part Information (6/20/94)....................................9
How to Create Auto Parts (6/20/94)....................................11
How to Use Layouts in Auto Parts (6/20/94)....................................12
How to Access Data in Auto Parts (6/20/94)....................................12
Auto Part Warnings (6/20/94)....................................14
How to Create Multi-Part Packages (6/20/94) 14
Copy Protection disables Connection Kit backup (8/25/94)..................15
Heap/Partition size strategy and Type1/Type3 errors (10/17/94)..15
The 'constant' keyword in evaluate slots (1/16/94)..............................16
CHANGED
If Statements & Constant Condition Errors (4/5/95)..................16

Serial Cable for NTK (9/15/93) (Obsoleted by 1.0 Newton Toolkit User's Guide p1-5)

Q: What is the pin configuration of the modem cable required to connect a tethered Newton to a desktop computer?

A: The serial cable you need is a called a **null modem cable**. That means pin 2 and 3 are crossed over. (That is, pin 2 on the Macintosh end goes to pin 3 on the Newton end, pin 3 on the Macintosh end goes to pin 2 on the Newton end).

The Apple-labelled cable model M0197 (part number 590-0552-A) is suitable for this purpose.

Adding PICT Masks With NTK (9/15/93) (Obsoleted by 1.0 Newton Toolkit User's Guide p4-6, p4-32, pC-14)

Q: How do I add a mask to a picture button using NTK? I want to use a mask so that the button highlights correctly when the user presses it.

A: Check the Use Mask box and add the mask as a resource having an exclamation point (!) appended to the name. For example, if your button's PICT is named "buttonPict", its corresponding mask would be stored in a resource file named "buttonPict!". The resource file must be added to the NTK Project for the picture slot editor to find the images.

Problems Changing Subview Links (9/15/93) (Obsoleted by NTK 1.0b7 or later.)

Q: How can I "unlink" linked subviews in separate NTK layout windows? To illustrate the problem, consider the following senario:
• A subview in layout#1 is linked to a subview in layout#2.
• The subview in layout#1 is deleted.
• Attempting to link a subview in layout #3 to the subview in layout#2 produces an error message claiming that the subview in layout #2 is already linked.

How can I break the old links in layout#2 in order to re-link its subview to the subview in layout #3?

A: NTK 1.0b4 occasionally did not properly unlink its views; later versions are much better about this and so it should rarely be a problem. However, if it does happen, you can "unlink" a layout by using ResEdit to open the offending layout file and remove the FMST resource.

NOTE: Removing other resources is downright dangerous!

How do I produce packages for Windows/DOS from a Mac? (9/15/93)

Q: How do I produce packages for Windows/DOS from a Mac?

A: The package format is identical on both Macintosh and Windows/DOS platforms. All you need to do is insert a Windows/DOS disk and copy your package to it. You can use something like Apple File Exchange™ to do this. This utility comes with System 7. Newton Connection Kit or Newton Package

Installer will download the package, whether using the Macintosh or Windows versions, and the result on the Newton will be identical.

Problems Downloading Packages (9/15/93)

Q: Why do I get -1040x errors when I download my package?

A: A -10401 error may indicate that your Macintosh is not running in 32-bit addressing mode. To correct this problem, open the Memory control panel on the Macintosh, set 32-bit addressing to On, and reboot the Macintosh. A -10401 error may also indicate that the old package is having problems cleaning up or leaving dangling pointers. Read Mike Engber's "The Newton Still Needs the Card You Removed" article.

A -10405 error usually indicates that the connection between NTK and the Newton is not valid. To correct this problem, you need to open a connection between the Newton and the Inspector window in NTK. For detailed instructions, see the section "Setting Up A Tethered Newton" in A Guide To The Newton Toolkit, supplied with NTK.

If you are using a Powerbook or Duo, there are other problems sources.

-You have an internal Modem and the Powerbook is set to use that instead of the serial port. (You can check this using the Powerbook control panel.)

-Appletalk Remote Access is on/installed. You may be able to use the Network Control Panel to set the network to Remote Only. This should free up the serial port (or printer/modem port on a Duo) for NTK use.

-Also, if you have an init or any other extension that keeps the serial (or printer port) constantly open, NTK can't open the port, and you will get -10405 errors. Beware of fax modem software and similar packages.

To test the viability of the connection, evaluate something in the Inspector, for example, "42". If the connection is not valid, NTK displays the message "NOT Connected to the Newt. Please connect!" in the Inspector window.

Insufficient User RAM
If you can't download successfully when the connection between Newton and NTK is valid and neither Newton nor NTK is displaying a dialog box,the Newton may not have sufficient user RAM available to store the package. Under these conditions, the package simply does not appear in the Extras drawer.

To correct the problem, check the package size and check the amount of available user memory on the Newton. The package size is shown in "Extras.Prefs.Memory.Remove Software". The size of a package in PCMCIA RAM is shown in "Extras.Card.Remove Software".

How do I change the default size of the Inspector Window? (9/15/93)

Q: How do I change the default size of the Inspector Window?

A: Save your Inspector, and next time the window is opened, the new size, position and monitor are respected.

Inspector Use (10/13/93)

Q: I tried out a one-line control flow statement in the Inspector as in:

```
 foreach i in [1,2,3] do print(i);
```

which gave no response at all (though single Print calls work fine). How do I test out control statements inside the Inspector window?

A: The Inspector's context is Newton's root frame, which is the frame in which your application's base view is installed. Loop constructs use local variables, which are not allowed at this "top level". You can work around this by defining a function (where locals are legal) on the fly and calling it, as in the following code fragment.

```
call func()
begin
      foreach i in [1,2,3] do print(i);
end with ()
```

Font Support? (11/10/93)

Q: Is there a way to include my own font in my project and access it from a package?

A: Currently, NTK does not support font resources. Stay tuned for future NTK versions that will provide this feature. See the sample code "Monaco Font" for one additional font you can download.

NTK Copy and viewFlags Problems (11/15/93) (Obsoleted with NTK 1.0 Final)

Q: When I copy views, the newly created views don't behave properly. What's wrong?

A: NTK versions 1.0b6 and1.0b7 have aproblem that happens sometimes when copying views that are Declared. The end result would be nil in a slot where a view was expected, yielding the error:

```
Exception |evt.ex.fr.intrp;type.ref.frame|: [-48811]
Tried to access slot "viewflags" in NIL context
Symbol: viewflags
```

This problem has been fixed in the latest NTK versions. If you encounter this problem, it can be repaired by opening the "Get Info" dialogs for the views that were moved and making sure they are declared to the proper parent in the new location.

Linked Subviews and View Positioning (12/5/93)

Q: I followed all the rules in the Newton Toolkit User's Guide when creating the floatNGo dialog box, but when this floater is opened it does not receive clicks. What is wrong?

A: Your floating window is larger than one of its parents, possibly your application's base view. User input (taps) are always clipped to parent regions, even though drawing may not be.

One solution is to make the floater a child of the root view, using BuildContext. See the Q&A Views, the Splash Screen sample, and Mike Engber's "Tales from the View System" article for details.

Project Data File Problems, 1.0b7 (12/18/93) (Obsoleted by NTK 1.0 final)

Q: My Project Data file isn't working.

A: If you have added your Project Data file into your project using an earlier version of NTK you will not be able to compile your own constants with 1.0b7.

The solution is to remove the Project Data file from the project, but to still keep it in the same folder as your other project and view template files.

With beta versions of 1.0 NTK, you must use the "Project Data" menu item when you want to create a Project Data file.

NTK, Picture Slots and ROM PICTs (12/19/93)

Q: How can I use a PICT in ROM from the Picture slot handler in NTK?

A: You have to use an AfterScript to set the appropriate slot in the view to point to the ROM based PICT (assuming that the constant for the PICT is defined in the NTK definitions file). Something like this in the AfterScript:

```
        thisView.icon := ROM_outboxbitmap;
```

AutoClose flag (1/4/94) (Obsoleted by 1.0 Newton Toolkit User's Guide p4-6)

Q: What does the "Auto Close" flag in NTK do, and should I use it?

A: The AutoClose flag causes all other AutoClose applications to be closed when your application is opened. The effect is that only one "auto close" application can be open at a time. You should *always* make your application auto-close, to help conserve memory and other resources, unless your application provides special functionality to other applications (like the built-in Calculator or Styles applications.)

Recognition Problems with the Inspector Window Open (3/8/94)

Q: When I have the Inspector window open and I debug the application, recognition does not work properly and the Newton complains about lack of memory. However, when I disconnect the Inspector, recognition works fine. What is going on?

A: The NTK inspector window uses system memory on the Newton side; the Toolkit App itself makes use of MNP in the Newton, which uses a 32K buffer taken from a space shared with the recognition working memory.

The 1.05 Sytem Update, and later releases of the Newton OS, give more space to this buffer, which makes the lack of memory error much less likely. However, if this happens you have several options:

- Disconnect the Inspector when testing the recognition side.
- Use the keyboard for text input while testing the code.
- Write shorter text items.

LocalTalk connections between Newton and NTK (3/8/94)

Q: I can't get the Inspector to work over AppleTalk.

A: The LocalTalk connection is unreliable for short bursts of information going both ways, as in Inspector connections. It works fine for package downloading and other large one-way transfers.

Developers should use only serial connections to the NTK Inspector in MessagePad platforms. Earlier versions of the NTK Toolkit application allowed AppleTalk connections to the Inspector, but this has been disabled in the NTK 1.0 and later versions of the Toolkit App. (Tap the words "Toolkit App" in the Toolkit Connection window to see the version number.)

Frames heaps, when, why, how to increase the Size (3/8/94) (Obsoleted by 1.0 Newton Toolkit User's Guide p4-11)

NTK has two special internal heaps, called the Main Frames heap and the Aux Frames heap. They can each run low on memory independent of each other. When this occurs the user will get an error dialog saying: "the XXX heap ran out of memory". The XXX will tell the user which heap ran out of memory, and thus needs it size increased. It is a waste to simply increase the size of both heaps.

In later versions of NTK, you can change the heap size in the NTK preferences dialog. In earlier versions, you must use the Macintosh resource editor ResEdit. Simply open the main Newton Toolkit file from ResEdit (or drag the NTK icon onto the ResEdit icon). You will be presented with the icons of numerous NTK resources in one large window entitled "Newton Toolkit". One of these icons will be named HEAP. The icon for HEAP looks like a collection of 0's and 1's. By double clicking on the HEAP icon, you will be presented with a small window containing two names: "Main Object Heap" and "Aux Object Heap". By double-clicking on either of these names you will be presented with the current value of that particular HEAP resource. You can change the value field by simply typing the new value.

We can't give a firm estimate of how large the Main Frames heap needs to be for any given project. However, the current implementation of NTK reads all of the layouts, user protos, etc. into memory so the main heap needs to be large enough to handle the accumulation of all that data. (Temporary data, for example local variables used in compile-time functions, will be reclaimed through garbage collection.) NTK 1.0 uses MultiFinder temporary memory for these heaps. In earlier versions, the Application partition size (from the Finder's GetInfo dialog) should be increased a corresponding amount.

The size of the Aux Frames heap can be quantified. If the application you are building is 200K then the Aux Frames heap needs to be about 200K. As the application grows to 300K then the size of this heaps needs to grow.

In addition, users might well run out of application memory if they have too many windows open or something like that. In this case they will get a generic message saying that "memory is too low to continue" or "out of memory". If this happens, simply increase the Multifinder application partition (in the Get Info dialog from the Finder), without changing the size of any of the two frames heaps.

Locale Used at Compile Time (3/8/94)

Q: `DefConst('myDate, StringToDate("30 June 1995"))` yields the wrong results.

A: When you do something like the above, you are asking NTK to evaluate `StringToDate("30 June 1995")` for you, in its own environment. The locale used inside NTK is always the English locale, and "30 June 1995" is not a valid date in that locale, so you get an undefined result.

If locale is important, you will have to make sure the expression is evaluated on a Newton set to the proper location. For constants, we suggest evaluating the expression once in the Inspector, and pasting the result into NTK as a constant definition.

Compression setting is per package , not per part (4/7/94)

Q: If I include a compressed auto part in my final package, and the final package is not compressed, the included auto part is also uncompressed, even though I built it compressed.

A: The compression setting is package wide. All parts in a single package are either compressed or uncompressed. You can not mix them.

The solution is to ship two packages, the auto part package compressed, and the application part package uncompressed.

Accessing Views Between Layout Windows (6/7/94)

Q: I have problems setting a protoStaticText text slot that is in one linked layout window from a button that is in another linked layout window. I tried to allow access to the base view from both linked layouts, but this didn't help. I even tried to allow access from the base view to both layouts, but this didn't help, either. What should I do?

A: There is no way to declare views across the (artifical) boundary imposed by the linked layouts. Until this feature of NTK is implemented, you must either create the link yourself at run time, or declare the button to the top level of the linked layout, and then declare the link.

For example, consider a view called `textThatChanges` which a child of `a` view called `changingContainer` and is declared to `changingContainer` with the name `textThatChanges`. `ChangingContainer` is the base view for a layout which is linked into the main layout, and the link (in the main layout) is declared as `changingContainerLink`. Code in the main layout can change the text of the `textThatChange` view like so:

```
   SetValue(changingContainerLink.textThatChanges, 'text, "Turn and face
the...")
```

See the DTS code sample DeclareDemo for a more detailed working example.

To do the equivalent of the declare yourself:

1) In the viewSetupFormScript script of the 'buttonThatChanges button, set the value of the base view's slot 'theTextView to self, as in the following code fragment:

```
func()
begin
      base.theTextView := self;
end
```

2) In the buttonClickScript script of the 'buttonThatSetsText button, use the global function SetValue to store new text in the text slot of the 'buttonThatChanges button, as in the following code fragment:

```
func()
begin
      SetValue(base.theTextView, 'text, "Now something happened!");
end
```

Note that this example assumes the existence of a view called `base`. In a commercial application, you must add error checking, as appropriate, to ensure that the base view does in fact exist.

Dangers of StrCompare, StrEqual at Compile Time (6/9/94)

Q: I've noticed that `StrCompare` can return different results at compile time than it does at run time. What gives?

A: While many functions, like `StrCompare`, are present in NTK at compile time, they should not be considered documented or supported unless explicitly defined in the Newton ToolKit User's Guide or other material from PIE.

In this case, the sort order for strings within the NTK NewtonScript environment is different from the ordering used on the Newton (and different from other commonly used desktop machine sort orders.) The differences are only apparent if you use characters outside the ASCII range, for instance, accented characters.

If it is necessary to pre-sort accented strings at compile time, you can write your own function that will return the same results as `StrCompare` on the Newton. Here is one such function (which assumes strings using only page 0 of the unicode table):

```
constant kNSortTable :=
'[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,2
6,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,
50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73
,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,6
5,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,
89,90,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,
114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131
,132,133,161,157,135,136,165,149,138,137,143,141,152,159,158,144,140,17
0,134,146,147,148,142,150,138,168,171,151,153,160,153,154,155,156,174,1
74,174,174,65,65,145,67,175,69,175,175,176,176,176,176,162,78,177,177,1
77,79,79,164,79,178,178,178,85,166,167,139,65,65,65,65,65,65,145,67,69,
69,69,69,73,73,73,73,169,78,79,79,79,79,79,163,79,85,85,85,85,172,173,8
9];

// function to compare strings (only page 0 characters)
// with the same order as Newton does.
defconst('kNewtonStrCompare, func(s1, s2)
begin
   local l1 := StrLen(s1);
   local l2 := StrLen(s2);
   local l := Min(l1, l2);
   local i := 0;
   while i < l and
      (r := kNSortTable[ord(s1[i])] - kNSortTable[ord(s2[i])]) = 0 do
         i := i + 1;
   if i = l then
      l1-l2
   else
      r;
end);
```

Dangers of Using _proto as an Unbound Variable (6/9/94)

Q: If I use `_proto` as an unbound variable (for example, "`func() return _proto`") my package gets about 80K larger than it used to be, and the function doesn't work. What's happening?

A: Because of the way constants are implemented by the NewtonScript compiler in NTK, there is an anomalous locally scoped constant defined within functions called `_proto`. Its value is a frame containing all constants scoped at the next highest enclosing function, with more `_proto` constants pointing up to a frame containing all the globally scoped constants.

If you use `self._proto` you won't have this problem. Because of the syntax of this statement, the `_proto` part cannot be interpreted as a constant, so it's kept as a symbol to be dynamically looked up at runtime.

This is, of course, a side-effect of the particular implementation of the current compiler, and not anything you should use, rely on, or otherwise exploit. (It's not even useful: you can get at everything in this constants frame using normal constants, and everything is defined in the NTK definitions file in a more readable format.)

Package Part Information (6/20/94)

A package is the basic unit of downloadable software. Both the Newton Connection and the Package installer download packages to the Newton. Packages contain parts. A package could contain multiple entries of various parts. Each part has a specific part code (four character type).

Here's a table of the currently supported parts:

Part Type	Part Code	Description
form	`form`	general purpose application created by NTK
book	`book`	books created by Newton Book Maker
auto	`auto`	faceless background apps
font	`font`	additional fonts
dictionary	`dict`	custom dictionaries
store	`soup`	store containing read-only soups

Form Parts

These are the basic parts in an NTK application. When a form part is installed, the following steps take place:

1: Its `partFrame` is `TotalCloned` (except for `theForm` slot).
2: The clone's `InstallScript` is called with one parameter, the `partFrame`.
3: A view is made using `partFrame.theForm` as the template.
4: This view is put in a slot of the root view using the app symbol as the slot name.
5: An icon is added to the extras drawer for that part.

When a form part is removed the `RemoveScript` is called. One parameter is passed, the same `partFrame` that was passed to `InstallScript`

Auto Parts

An auto part is a faceless application. Consider it to be the INIT of the Newton environment.

Auto parts are installed much like Form parts, except for the following differences:

1: The `partFrame` is not `TotalCloned`.
2: The `InstallScript` is called with two parameters, the `partFrame` and a `removeFrame`. The `InstallScript` has access to other data through the `partFrame.partData` slot. There is no `partFrame.theForm` slot.
3: No view is created.
4: No icon is added to the extras drawer.

When an auto part is removed its `RemoveScript` is called. It is passed one parameter, the `removeFrame` that was passed to the `InstallScript`.

A variation of the auto part is a so-called "dispatch auto part". These install like a regular auto part, but then are automatically removed. Removal works like a regular auto part, except it is triggered automatically. Dispatch-only auto parts load, their `InstallScript` runs, their `RemoveScript` runs, then they're gone.

Because Auto parts do not add an icon to the Extras drawer, it makes sense to use auto parts if, for instance, you want to add new panels to the Formulas roll without creating any icons.

You can also use auto parts to provide data for other applications. Break up the data into smaller auto part packages, and allow end users to install various combinations to their Newtons. The auto parts would each create or modify slots in the root view (using your application suffix!) to allow other applications access to data in their packages.

Dispatch auto parts could be used for making one-time changes in the system, such as creating or adding entries to soups or dictionaries. (Note that any changes are lost after a reset of the Newton.) RemoveScripts for auto-dispatch auto parts are not guaranteed to run, but this should not pose a problem as they are not useful and so should not be used.

Font Parts

Font parts install new fonts on the Newton when they are loaded. See the latest "Monaco Test" DTS sample code showing how a font part works.

Dictionary Parts

Dictionary parts provde a way to install custom dictionaries, and keep these in the package. There is no facility available yet to create dictionary parts. (Unlike dictionary parts, RAM-based custom dictionaries reside in the NewtonScript heap.)

Store Parts

Store parts allow packages to contain a read-only store. This store contains soups with the data.

Packages

Packages can contain more than one part. (If they do, they are called multi-part packages.) If a multi-part package contains two form parts, then the two parts' icons will show up in the Extras drawer. In other words, this is like installing the parts separately.

Multi-part packages could be used to facilitate multi-programmer projects; the application is broken into modules which can be compiled, downloaded, and tested separately, then integrated into a single package for the final product.

How to Create Auto Parts (6/20/94)

What follows is the current way to create auto parts in NTK 1.0.1.Expect changes in the user interface for this in the future.

Specify "`auto`" as your application name and symbol in the project settings dialog, or "`auto!`" if you want a dispatch auto part.

You need to create a dummy main layout so your project will compile, but this layout is not put into the package.

All the code needed for the auto part goes to the Project Data file. You create your `InstallScript` and `RemoveScript` in a similar fashion as with form parts. Here's a simple example:

```
InstallScript := func(partFrame, removeFrame)
begin
     ...
end

RemoveScript := func(removeFrame)
begin
     ...
end
```

If you want to have data in your auto part, define a `partData` global variable in the Project Data file, as in:

```
     partData := {s1: "moe", s2: "larry" , s3: "curly"};
```

The contents of this global `partData` variable will be stored away in your part frame in a slot with the same name. You should make `partData` a frame whose slots contain the various pieces used by your `InstallScript`. You could also embed data inside your `InstallScript` using literals or constants.

How to Use Layouts in Auto Parts (6/20/94)

You could define view templates for the auto part as pure NewtonScript frames (inside the Project Data file). It is often more convenient to generate view templates using the graphical NTK environment.

The layouts in your project window will not be automatically included in your package. (Auto part `partFrames` do not have the `theForm` slot.)

In order to include the NTK layouts in the package, you can:
0: Initialize `partData` in Project Data to a frame (probably to `{}`)
1: Create your templates in NTK normally.
2: Add these to your project.
3: Give each template top-level view an AfterScript that places the template in partData.

Here's an example of an AfterScript:

```
     partData.fooTemplate := thisView;
```

Now you are able to access this template from your `InstallScript` using `partFrame.partData.fooTemplate`.

How to Access Data in Auto Parts (6/20/94)

Auto parts do not have a base view, nor application names, nor app symbols. In other words, you can't access their application base view using the `GetRoot().(kAppSymbol)` mechanism.

If you want to access data from the auto part's `InstallScript` you have to provide code to place the data where other applications can access it. Say for instance that you want to provide data to a form part (named by `kTargetAppSymbol`) from an auto part. Here's an example `InstallScript` (and the `RemoveScript`) that place a reference to the data in that application's base view:

```
InstallScript := func(partFrame, removeFrame)
begin
   local targetView := GetRoot().(kTargetAppSymbol);
   if targetView then
      targetView.(EnsureInternal('theData)) :=
         partFrame.partData;
end;

RemoveScript := func(removeFrame)
begin
   local targetView := GetRoot().(kTargetAppSymbol);
   if targetView then
      RemoveSlot(targetView, 'theData);
end;
```

This assumes that the target app is installed before the auto part. If this is not guaranteed, you could instead use a global variable to share data. However, try to keep the number of active globals to a minimum in the system, and use a signature-specific global name. Here's an example how to use globals for data sharing:

```
InstallScript := func(partFrame, removeFrame)
begin
   local gData := GetGlobals().|app:Signature|;
   if not gData then
   begin
      gData := EnsureInternal({});
      GetGlobals().(EnsureInternal('|app:Signature|) := gData;
   end;

   gData.(EnsureInternal('theData)) := partFrame.partData;
end;


RemoveScript := func(removeFrame)
begin
   local gData := GetGlobals().|app:Signature|;
   RemoveSlot(gData, 'theData);

   if Length(gData) = 0 then
      RemoveSlot(GetGlobals(), '|app:Signature|);
end;
```

In this case we create a unique global using the signature as part of the name. The InstallScript creates this global variable if needed. The RemoveScript eliminates this global variable if we remove the last datum.

Auto Part Warnings (6/20/94)

The `partFrame` of an auto part is not copied into internal RAM before being executed. All the data and code in the `InstallScript` reside in the package. For instance, if the `InstallScript` creates a slot in the `removeFrame` for the RemoveScript, you need to do an `EnsureInternal` on the slot name and most likely with the contents of the slot as well:

```
removeFrame.(EnsureInternal('foo)) := EnsureInternal(...);
```

In some cases a simple `Clone` would be enough for the contents of the slot. This all depends what the `RemoveScript` will do with the slot contents.

Note however that the `RemoveScript` is copied into internal RAM. This is necessary because the `RemoveScript` must executes even if the package is no longer available (because of PCMCIA card removal.) Thus you don't need to use `EnsureInternal` in your `RemoveScript`.

How to Create Multi-Part Packages (6/20/94)

NTK 1.0.1 creates multi-part packages by merging packages together. You specify the packages by putting each one in the same folder, and renaming them in the following scheme:

```
include.pkg
include.pkg1
include.pkg2
include.pkg3
...
```

NTK will create a single package containing the part from the current folder followed by the parts in the various included packages in order of their file names. The order of the included packages and the parts within them determines the order in which the parts are installed on the Newton. The parts in include.pkg, then in include.pkg1, include.pkg2, and so on are installed in order, the part produced by the currently open project is installed last.

Certain aspects, such as package name, compression, copyright, version and copy-protection, apply to a package as a whole, and not to its individual parts. Only the information from the current NTK project is placed in the resulting package, the same information from the included packages is ignored.

Dispatch-only is also a package-wide setting, but can only be set (and is only useful) for auto parts. If the NTK project being built is a dispatch only auto part, then the entire package becomes auto dispatch, and will be removed (and the RemoveScripts for all appropriate parts called) after it is installed.

Note that you have to be careful with naming the parts; for example, two form parts can't have the same name in a multi-part package.

Copy Protection disables Connection Kit backup (8/25/94)

Q: What does the "Copy protected" check box in the NTK Project Settings dialog do?

A: It sets the copy-protected flag for the package. Various parts of the Newton OS that deal with packages should respect this flag and not copy flagged packages. In particular, the Newton Connection Kit will not "backup" copy-protected packages during a synchronize operation.

(PCMCIA backup does not currently respect the copy-protected flag.)

Heap/Partition size strategy and Type1/Type3 errors (10/17/94)

Q: I'm trying to build a really big book/application, and either I can't launch NTK, I run out of heap space during the build, or NTK crashes with Type 1 or Type 3 errors. How does it all work?

A: As described on p. 4-11 of the Newton Toolkit User's Guide, there are two heaps in NTK. The main heap is used for platform file material, global data, editor windows, and working space during compiles. NTK allocates this heap from the NTK partition when it is launched. This means that to launch, the "preferred size" setting in the Finder Get Info window for NTK must be large enough to hold the main heap, plus a couple of megabytes for NTK itself.

During the final stage of the compile, NTK allocates the build heap, grabs the output of the compile from the main heap, and copies it (and anything it references) into the build heap. (This guarantees no unnecessary data will make it into the resulting package.) NTK attempts to allocate the build heap from the current free memory in the Macintosh — that is, the Largest Unused Block as shown in the About this Macintosh dialog in the Finder. If there is not sufficient space there, NTK attempts the allocation from its own partition, swapping out resources if necessary. If there is still not enough space, NTK puts up an error dialog and the build fails.

Here is a good strategy for setting the various heap sizes, based on the expected size of the package being built:

For speed, the NTK main heap should be large: at least 1.5 times the size of the output package; it should also have additional space for editors, globals, and so on. If you open many editor windows at the same time, or have large Global Data, then increase space accordingly.

In NTK 1.0.1, there is an upper limit of 16M for the size of the main heap, because of a bug that occurs addressing more than 24 bits of heap space. (Using more than 16M can result in Type 1 or Type 3 errors when heap garbage-collection occurs.)

The build heap does not need to be any larger than the size of the resulting package.

The NTK partition (in the Finder Get Info box) should either be the main heap plus a couple of megabytes, or the main heap plus the build heap plus a couple of megabytes, depending on how many other applications you plan to run at the same time as NTK.

The ‘constant’ keyword in evaluate slots (1/16/94)

Q: I tried putting a constant in a beforescript for a prototype definition, for instance:

```
    constant aConstant:= 29;
```

When I built the package I received an error message stating that this constant was being redefined. I removed the declaration for the constant from the beforeScript, but NTK thinks that the constant is defined somewhere, even when I reload the project. How can I remove this hidden constant declaration so that I can change its value to something else? Why can't I define a constant in a beforeScript?

A: You have witnessed a bug in the way NTK currently deals with evaluate slots. In NTK, when you close or apply an evaluate slot (you can use command-E or the check box button to apply the contents), NTK tries to do a syntactical check of the contents. The way it accomplishes this is to actually compile the contents of the evaluate slot. In most cases, this is not an issue since NTK does not actually execute any code within the evaluate slot until you build your project. In the case of the "constant" statement, compiling code with a "constant" keyword has a side effect of storing the constant in NTK (not the project) for further compilations.

Here are some workarounds:

1)Move the constant definitions to the project data file, then quit and reopen NTK (not just the project file).
2) Change the constant definitions to DefConst('theName, theValue) in evaluate slots. Since DefConst is a global function (even if available only at compile time) and not part of the language itself, the syntax check of the code with a DefConst function is safe since the code which calls DefConst will be compiled, but not performed/executed during the syntax check.

Constants defined in the Project Data file (and in other parts of your project) normally exist in an a separate area for constants. This area is reconstructed every time you build your project. If the constants are accidentally created when 'checking' the evaluate slot, they are created in an area for global constants in NTK.

Note that since this only occurs during "syntax checking", if you quit NTK, open NTK, and then build your project, you should not see the "redeclaration" errors unless you edit the slot. However, we recommend that you choose one of the workarounds mentioned above so that you may edit your slots without problems.

CHANGED: If Statements & Constant Condition Errors (4/5/95)

Q: I have found a problem with functions containing nested `if`s. In some cases they seem to produce duplicate code. It seems to happen with any compile-time constant (even true or nil) in the `if` statement:

```
func()
   begin
      if true then
         debugStr := print("Blah");
      nil;
   end;
```

This causes "Blah" to be printed twice when run:

```
"Blah"
"Blah"
```

A: This is a bug in the current compiler's handling of if statements with constant conditions. It only happens when the `if` statement uses a constant conditional and the result of the `if` statement is unused. There are several ways to work around the problem, depending on your needs.

Example 1 - assign the result of the if statement to a local variable.

```
func()
   begin
      local trash := (if theConstant then
                           print("blah");)
      nil;
   end
```

Example 2 - use the result of the if statement in a boolean operation.

```
func()
   begin
      (if theConstant then
         print("blah")) or true;
      nil;
   end
```

Example 3 - trick the compiler into fixing it for you

```
DefConst('kNilFunc, func() nil);

func()
   begin
      if theConstant then
         begin
            myCount:= ticks();
            call kNilFunc with ();
         end;
  end
```

Example 1 shows the easiest way to avoid this bug: assign the result of the if statement to a local. This is especially handy if you needed the result of the statement anyway.

The drawback to this workaround is that an extra local variable is used in the function. The variable will still be there, but with a value of NIL, if the conditional is false.

Example 2 uses the result of the conditional in a boolean operation. The result of the boolean operation itself is unused. This workaround will leave a small amount of extra code in the resulting function, but no local variables are used.

Example 3 shows a way that your code can be structured to avoid the symptoms of this bug without leaving any extra runtime code if the constant is NIL. It turns out that if the last executed statement in the conditional block is a "call...with", a call to a global function, or a message send (self:dummy())), the problem does not surface. Placing a global function call or a "call...with" instruction at the end of the if statement clause (or the only statement within the clause) will supress the problem.

End of DTSQA

NEWTONSCRIPT PROGRAMMING LANGUAGE

TABLE OF CONTENTS:

NewtonScript Object Sizes (6/30/94)..........1
PLAY CATCH - NEWTONSCRIPT EXCEPTIONS..........3
Introduction..........8
Garbage Collection (10/15/93) (Obsoleted by NS Guide 1.0 Final Documentation)..........8
Order of Slots in Frames (9/15/93)..........9
Structured Literals (9/15/93)..........9
Passing Arguments (9/15/93)..........10
Slot (Variable) Lookup (9/15/93)..........11
Testing the Existence of a Slot (Variable) (9/15/93) (Obsoleted by NS 1.0 Final Documentation)...........12
Calling Methods Out Of Slot Context (9/15/93) (Obsoleted by NS Guide 1.0 Final Documentation)..........13
Inherited (10/13/93) (Obsoleted by NS Guide 1.0 Final Documentation) 13
Deeply (foreach deeply in...) (10/13/93)(Obsoleted by NS Guide 1.0 Final Documentation)..........14
Compile Function (10/9/93) (Obsoleted by NS Guide 1.0 Final Documentation)..........15
Function Definitions (10/9/93)..........16
Closures and Perform (10/9/93) (Obsoleted by NS Guide 1.0 Final Documentation)..........16
Symbol Hacking (11/11/93) (Obsoleted by NS Guide 1.0 Final Documentation)..........17
Literals and run time (11/11/93)..........18
Missing Semicolons cause Weird Errors (12/11/93) (Obsoleted by NS Guide 1.0 Final Documentation)..........19
Classical OOP programming withNewtonScript (12/17/93)..........19
Only 7-Bit ASCII In NewtonScript Symbols (1/26/94) (Obsoleted by NS Guide 1.0 Final Documentation)..........22
How to Avoid _parent Problems (6/28/94)..........22
Symbols vs Path Expressions and equality (7/11/94)..........22
Function Size and "Closed Over" Environment (7/18/94)..........23
CHANGED
If Statements & Constant Condition Errors (4/5/95)..........24

NEW

Assignment with path expressions as lValues (4/12/95)..........24

NewtonScript Object Sizes (6/30/94)

Generic

NewtonScript objects are objects that reside either in the read-write memory, in pseudo-ROM memory, inside the package or in ROM. In MessagePad platforms, these objects are aligned to 8-byte boundaries. Alignment causes a very small amount of memory to be wasted, usually less than 2%.

The Newton Object System has four built-in primitive classes that describe an object's basic type: immediates, binary objects, arrays, and frames. The NewtonScript function `PrimClassOf` will return an object's primitive type.

Immediates

Immediates (integers, characters, TRUE and NIL) are stored in a 4-byte structure containing up to 30 bits of data and 2 bits of primitive class identification.

Referenced Objects

Binaries, arrays and frames are stored as larger separate objects and managed through references. A reference is a four- byte object. The binary objects, frames, or arrays themselves are stored separately as objects containing a so-called Object Header.

Object Header

Every referenced object has a 12-byte header that contains information concerning size, flags, class, lock count and so on. This information is implementation-specific.

Symbols

A symbol is a binary object that contains a four-byte hash value and a name, which is a null-terminated ASCII string. Each symbol uses 12 (header) + 4 (hash value) + length of name + 1 (null terminator) bytes.

Binary Objects

A binary object contains a 12- byte header plus space for the actual data (allocated in 8 -byte chunks.)

Strings

String as binary objects of class `String`. A string object contains a 12-byte header plus the Unicode strings plus a null termination character. Note that Unicode characters are two-byte values. Here's an example:

```
    "Hello World!"
```

This string contains 12 characters, in other words it has 24 bytes. In addition we have a null termination character (24 + 2 bytes) and an object header (24 + 2 + 12 bytes), all in all the object is 38 bytes big. Note that we have not taken into account any possible savings if the string was compressed (using the NTK compression flags).

Array Objects

Array objects have an object header (12 bytes) and additional four bytes per element which hold either the immediate value or a reference to a referenced object. To calculate the total space used by an array, you need to take into account the memory used by any referenced objects in the array.

Here's an example:

```
[12, $a, "Hello World!", "foo"]
```

We have a header (12 bytes) plus four bytes per element (12 + (4 * 4) bytes). The integer and character are immediates, so no additional space is used, but we have 2 string objects that we refer to, so the total is (12 + (4*4) + 38 + 20 bytes) 86 bytes. We have not taken into account savings concerning compression. Note that the string objects could be referred by other arrays and frames as well, so the 38 and 20 byte structures are stored only once per package.

Frame Objects

We have two kinds of frames: frames that don't have a shared map object; and frames that do have a shared map object. We take the simple case first (no shared map object).

The frame is stored as two arrays. One array, called the frame map, contains the slot names, and the other contains the actual slot values. A frame map has one entry per symbol, plus one additional 4 -byte value.

The frame map uses a minimum of 16 bytes. If we add the frame's object header to this, the minimal size of a frame is 28 bytes. Each slot adds 8 bytes to the storage used by the frame (two array entries.) Here's an example:

```
{Slot1: 42, Slot2: "hello"}
```

We have a header of 28 bytes, and in addition we have two slots, for a total of (28 + (2 * 8)) 48 bytes. This does not take into account the space used for each of the slot name symbols or for the string object. (The integer is an immediate, and so is stored in the array.)

Multiple similar frames (having the same slots) could share a frame map. This will save space, reducing the space used per frame (for many frames all sharing the same map) to the same as used for an array with the same number of slots. (If just a few frames share the frame map, we need to take into account the amortized map size that the frames share. So the total space for N frames sharing a map is N*28 bytes of header per frame, plus the size of the frame map, plus the size of the values for the N frames.

Here's an example of a frame that could share a map with the previous example:

```
{Slot1: 56, Slot2: "world"}
```

We have a header of 12 bytes. In addition, we have two slots (2 * 4), and additional 16 bytes for the size of a map with no slots — all in all, 36 bytes. We should also take into account the shared map, which is 16 bytes, plus the space for the two symbols.

When do frames share maps?

1. When a frame is cloned, both the copy and the original frame will share the map of the original frame. A trick to make use of this is to create a common template frame, and clone this template when duplicate frames are needed.

2. Two frames created from the same frame constructor (that is, the same line of NewtonScript code) will share a frame map. This is a reason to use `RelBounds` to create the `viewBounds` frame, and it means there will be a single `viewBounds` frame map in the part produced.

Note again that these figures are for objects in their run-time state, ready for fast access. Objects in transit or in storage (packages) are compressed into smaller stream formats. Different formats are used (and different sizes apply) to objects stored in soups and to objects being streamed over a communications protocol.

PLAY CATCH - NEWTONSCRIPT EXCEPTIONS

(9/15/93) (Obsoleted by NS Guide 1.0 final docs)

Introduction

The NewtonScript exception system provides a structured way for a module to report a failure to another module. It communicates the type of the failure, and allows (optionally) transmission of data that may explain the reason for the failure. The NewtonScript implemention uses the exception system to report errors (for example, errors in an application, failures due to lack of resources, or internal errors). Developers can use the exception system to communicate errors between different modules in an application.

Basic Syntax

The Exception handling is based on a `try block`, and if an exception is thrown from somewhere, the code should catch this with a special `onException` block.

Try Statements

The general format of the try statement is:

```
try
      <statement list>
onexception <exception symbol> do
      <statement>
onexception <exception symbol> do
      <statement>
...
```

An example of this might be:

```
try
begin
      // create entries and store them to soups
end
onexception |ext.ex.store.err| do
begin
      // problems with the store, most likely not enough memory
      // do something
end
```

The <exception symbol>s must be single part exception symbols. Details about exception symbols — that is, what multiple parts mean — are discussed below.

One or more onexception clauses are allowed. The try statement executes <statement list>. If no exceptions are thrown in the process, then the value of the try statement is the value of the last statement in <statement list> and the onexception clauses are never executed.

If during execution of <statement list> an exception is thrown, then the execution of <statement list> stops and control is transferred to one of the onexception clauses. When an exception, X, is raised, the onexception clauses of the try statement are examined in order. The first clause whose <exception symbol> is a prefix of any of X's parts is executed and its <statement> value becomes the value of the try statement.

If no onexception clause matches the exception, then the exception is passed to the next enclosing try statement for processing. In a Newton application, the exception will ultimately be handled by the view system (putting up an error dialog box) if your application doesn't catch it.

NOTE: It is important that the search for a matching onexception clause uses dynamic scoping, not lexical scoping.

Since all exceptions have an evt.ex part, an exception clause with evt.ex as its symbol will catch any exception. (See the following section for a description of "parts".) If your try statement has an evt.ex clause, it should be last, since onexception clauses occuring after it will never be executed. In general, you should order your onexception clauses from most specific to least specific.

Exception Symbols

Exceptions have names which are symbols. These symbols have a particular format which must be adhered to. An exception name consists of one or more parts separated by semi-colons. Each part is a structured name beginning with evt.ex.

A few facts about exception symbols:

- They can have as many as 127 characters.
- They can contain periods, so the symbols must be enclosed in vertical bars (|'s)
- They can have multiple parts, separated by semi-colons.
- They must have a part starting with evt.ex.

The simplest possible exception symbol is |evt.ex|. An example of an exception symbol with two parts would be '|evt.ex;type.ref|. Some more examples: '|evt.ex.div0|, '|evt.ex.fr.intrp;type.ref.frame|.

Exception Frames

Associated with every exception is an exception frame. When handling an exception you can get this frame using the global function, CurrentException(). An exception frame always has a name slot which contains the exception symbol. It will contain one other slot whose name and contents depend on the type of exception as follows: (this info is summarized on the NS Quick Ref Card)

• if type.ref is a prefix in the exception symbol, then the other slot will be called data and can contain anything.

```
|evt.ex;type.ref|: {name: <string>, data: <frame>}

Ex: {name: "the llama exception", data : {type: 'inka, size: 42, weight: 177}}
```

• if evt.ex.msg is a prefix in the exception symbol, then the other slot will be called message and contain a string

```
|evt.ex.msg|: {name: <string>, message: <string>}

Ex: {name: "Ho ho exception", message: "You have a serious problem, mate"}
```

• otherwise, the other slot will be called error and will contain an integer (error code)

```
|evt.ex|: {name: <string>, error: <integer>}

Ex: {:name: "Hi Ho exception", error: -48666}
```

Here are examples of exception frames from real life:

• Example A, division by zero:

```
{name: |evt.ex.div0|, error: 1764744}
```

If you want to catch division by zero errors, here's the trick:

```
try
      5/0
onexception |evt.ex| do
      CurrentException();
```

• Example B, undefined variable:

```
{name: |evt.ex.fr.interp; type.ref.frame|, data: {error:-48807, symbol: foo}}
```

How to Raise Gentle Exceptions

You can throw an exception using the global function, Throw(<exception symbol>, <exception data>). The value you pass for <exception data> is put into the "other" slot of the exception frame. Make sure it is the correct type (as per the above rules) or your call to Throw will raise another exception.

If the code block has more than one try statement in effect at one time, you can pass control to the next enclosing try statement using the Rethrow function.

Here are examples of the three different ways to create and throw exceptions. Note that you need to send an exception symbol ('):

```
Throw('|evt.ex.foo|, 99);

Throw('|evt.ex.msg|, "string");

Throw('|evt.ex;type.ref.something|, ["a", "b", "c"])
```

Default Exception Handling

Unfortunately, there is currently no general way for your application to specify a default exception handler, for example, viewExceptionScript. This means you need to use try statements wherever you want to catch exceptions.

Examples

Here's some code to test out the exception system. This tries various cases and is designed to print no output, assuming the exception system works as advertised. However, the NTK Inspector prints out every exception that is thrown, whether it's handled or not. Expect to see output; however, none of thePrint("*error?*) statements should execute.

```
  //simple cases & a nested block
  try
     begin

        try
           Throw('|evt.ex;type.ref;foo|,"dsafsda");
           Print("*error*1");
        onexception |foo| do
           begin
              //Exception handled quietly
           end;

        try
           Throw('|evt.ex;type.ref;foo|,"dsafsda");
           Print("*error*2");
        onexception |type.ref| do
           begin
              //Exception handled quietly
```

```
         end;

      //outer block should catch this one
       Throw('|evt.ex.foobar|,42);

      Print("*error*3");

   end
onexception |evt.ex| do
   if CurrentException().error <> 42 then
      begin
         Print("*error*4");
         Print(CurrentException())
      end;



//try out rethrow()
try
   begin

      try
         Throw('|evt.ex|,42);
         Print("*error*5");
      onexception |evt.ex| do
         begin
        //outer block should catch this one
            Rethrow();
         end;

   Print("*error*6");

   end
onexception |evt.ex| do
   if CurrentException().error <> 42 then
      begin
         Print("*error*7");
         Print(CurrentException())
      end;


//try it out with fn calls
try
   begin

      call func()
         begin
            try
               call func(esym) Throw(esym,42) with ('|evt.ex|);
               Print("*error*8");
            onexception |type.ref| do
               begin
              //outer block should catch this one
                  Rethrow();
               end;
         end
      with ();

      Print("*error*9");

   end
onexception |evt.ex| do
```

```
      if CurrentException().error <> 42 then
         begin
            Print("*error*10");
            Print(CurrentException())
         end;


   Print("no news, is good news");
```

Introduction

This document addresses NewtonScript Programming Language 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.

Garbage Collection (10/15/93) (Obsoleted by NS Guide 1.0 Final Documentation)

In NewtonScript, the run-time module, not the programmer, is responsible for allocating storage for objects and for reclaiming the storage of objects that are no longer used. Garbage collection is carried out by the basic object system, so it's not technically NewtonScript that does it. The storage taken up by an object will be freed sometime after the last reference to that object goes away.

One of the many benefits of garbage collection is that the programmer has to think about freeing objects only in a small number of important places, as opposed to dealing with the constant background worry that objects must be freed. Wth automatic garbage collection you only need to consider disposing objects in rare cases.

One place where you *should* think about it is when you close an application; here you want to free as much storage as possible. You do this by either removing slots or by setting slots to `nil` in the application's context frame. Setting the value of all slots and variables referring to a particular object to `nil` allows the Garbage Collector to destroy the object.

Within the Newton operating system, automatic garbage collection is triggered every time the system runs out of memory. There's not really any reason to invoke garbage collection manually. However, if you must do so, you can call the global NewtonScript function `GC`. Consult the *Newton Programmer's Guide* to find out more about the `GC` function.

Order of Slots in Frames (9/15/93)

Q: The description of the foreach function in the manual might be taken as a hint that slots in frames are ordered. Does slot or array order imply anything about layout of data in memory (as with C structs?)

A: Slots are not ordered, nor have we seen any performance gains from ordering slots for particular sequences or search criteria. Don't count on their being in any particular order, nor on data structures in memory being ordered with any regard to the placement of slots (as in the ANSI Common LISP object extension CLOS).

Q: Where is a newly-defined slot placed in a frame? at the end?

A: Because slots are unordered, the location of a new slot cannot be predetermined.

Structured Literals (9/15/93)

Q: Is it really dangerous to assign a string literal to a variable, as in the following example?

```
s := "abc";
```

A: That depends on what you're going to do with s. If you're going to modify it, then yes, it's dangerous. In that case, you should use

```
s := Clone("abc");
```

In general you should treat string literals in NewtonScript as read-only (just as you should in ANSI C). String literals might reside in read-only memory or share storage with other literals.

The same warning applies to quoted array and frame literals, as in the following example:

```
 '[1,2,3] or '{one: 1, two: 2, three: 3}
```

Here are some rules regarding string literals and quoted array/frame literals:

- Treat them as read only. Use clones when necessary.

- Realize that they always represent the same object.   In the example below,

```
func foo() begin "abc"
end;
```

- always returns exactly the same string - not different, equivalent strings.

- In your code, an unquoted array or frame literal will generate a new array or frame each time it is evaluated. So it’s acceptable for a function to return [1,2,3] or {one: 1,two: 2,three: 3} as long as they're not quoted.

Passing Arguments (9/15/93)

Q: Is NewtonScript call-by-reference or call-by-value?

A: A seemingly simple question. The short answer is “call-by-value,” the litmus test being that

```
func foo(x) x := 1;
```

does not affect the argument passed to it. However, this is not meant to imply that NewtonScript functions can never make changes to their arguments. If the argument has structure—for example, if it's an array, frame, or string—then changes to the internal structure are persistent. Consider the following code fragments:

```
func foo(x) x[0] := 1;
// changes the 1st element of array arguments

func foo(x) x.slot1 := 1;
// changes the slot1 slot of frame arguments
```

Depending on what you're used to, this example might be confusing because the concepts of call-by-reference, call-by-value are mixed, and copying arguments are intertwined (and often misunderstood.) The following discussion of this terminology with respect to other languages may help clarify the differences.

Pascal - Supports both call-by-reference (VAR parameters) or call-by-value (the default). When array or record arguments use call-by-value, copies of the arrays/records are made and passed in. To Pascal programmers, NewtonScript may seem to pass arrays and frames as VAR parameters.

C - Only supports call-by-value. Struct (not struct*) arguments are
passed as values; that is, a copy of the struct is passed. However, array arguments pass in the address of the first element of the array. Thus, C programmers will find the array arguments to be treated normally.

LISP - Only supports call-by-value. Copies are not made of structured
arguments (lists, structures, arrays, objects,...) Argument passing in Newton Script works just like it does in LISP.

Slot (Variable) Lookup (9/15/93)

Q: What differences are there among the various ways to access a slot?

A: The means by which you access a slot specifies the search path and inheritance mechanisms used to locate it. Each of the five ways to access a slot is described below.

•Specifying the slot name only

If you specify only the name of the slot, as in the example

```
theSlot
```

the search begins in the current function frame with lexically-visible variables, followed by slot names. In other words, if the currently-executing function has a local variable named theSlot, it is found before a local variable named theSlot in the enclosing function, and so on. Similarly, any of those variables are found (in scope order) before an actual slot named theSlot is found. If a variable named theSlot is not found, global variables are searched next. If a global variable is not found, the current receiver is searched for a slot named theSlot.

If the slot is not found in the receiver of the message, the remaining frames are searched in order of prototype and parent inheritance. An exception is thrown if you try to access a slot that can't be found. If you do an assignment operation using an undeclared slot, the slot is created if you place the self. prefix in front of the slot name; if you do the operation without the self. prefix you will get a local variable.

•Using the dot (.) operator

If you use the dot operator to specify the frame and the slot, as in the example

```
myframe.myslot
```

the search begins in the specified frame. If the slot is not found in the specified frame, the remaining frames are searched in order of prototype inheritance. Parent inheritance paths are not searched. If the slot does not exist, the system returns NIL.

•Using the GetSlot function

If you call

```
GetSlot(frame, slot)
```

only the specified frame is searched for the specified slot. Inheritance paths are not searched.

•Using the GetVariable function

If you call

```
GetVariable(frame, slot)
```

the search begins in the specified frame. If the slot is not found in the current frame, the remaining frames are searched in order of prototype and parent inheritance.

•GetVar will be removed from the language

You can usually simulate GetVar using GetVariable(self,slot). The difference will be that local variables will not be found - only slots.

If you use GetVar you may find that is not available in future products, in which case your code will throw an exception; or GetVar may work differently not finding local variables anymore, in which case your code may subtly break. DTS advises against using GetVar.

Testing the Existence of a Slot (Variable) (9/15/93) (Obsoleted by NS 1.0 Final Documentation).

Q: What differences are there among the various ways to determine whether a slot exists in a frame?

A: The means by which you test for a slot's existence specifies the search path and kinds of inheritance used to discover it. The following examples describe three ways to discover the existence of a slot in a frame.

• Using the exists operator

The exists operator follows the same rules as the expression to which it is applied. For example, if you use this operator to test a slot access expression, as in the example

```
myframe.myslot exists
```

the search begins in the specified frame. If the slot is not found in the current frame, the remaining frames are searched in order of prototype inheritance only. Parent inheritance paths are not searched.

However, when using the exists operator to check whether an identifier exists, as in the example

```
myvar exists
```

both prototype and parent inheritance paths are searched.

• Using the HasSlot function

If you call

```
HasSlot(frame, slot)
```

only the specified frame is searched for the specified slot. Inheritance paths are not searched.

• Using the HasVariable function

If you call

```
HasVariable(frame, slot)
```

the search begins in the current frame. If the slot is not found in the current frame, the remaining frames are searched in order of prototype and parent inheritance.

Calling Methods Out Of Slot Context (9/15/93) (Obsoleted by NS Guide 1.0 Final Documentation)

Q: I'm trying to pull a method out of a slot and call it. Unfortunately, the functions apply and call both seem to set self to {}. Is there another way?

A: The global function Perform(frame, message, paramaterArray) does what you want, except that the method needs to be in a slot in the current frame if you really want to preserve the value of self.

Inherited (10/13/93) (Obsoleted by NS Guide 1.0 Final Documentation)

Q: What is the function of "inherited:" and "inherited:?"

A: In NewtonScript, the inherited:X() message send specifies a NewtonScript function call on the function X, where X is found UP the inheritance chain starting from the caller's _proto. In other words, lookup for that method ONLY starts up the caller's _proto chain, NOT in "self" (the currently executing frame). The conditional message send (`:?`), with `inherited` specified, operates almost exactly the same except that no error results if the given method (function) is NOT found. The conditional message send (`:?`), with `inherited` specified, is helpful in complex heirarchies because you can use it without worrying about whether any such method was ever implemented higher up in the inheritance chain.

Deeply (foreach deeply in...) (10/13/93)(Obsoleted by NS Guide 1.0 Final Documentation)

Q: What is the difference between "foreach" and "foreach deeply in"?

A: 'Deeply' specifies that only the slots/elements of the current frame/array are listed, except that if passed a frame, slots of ancestors up the _proto chain are considered elements of the current frame.

Here is some Inspector code to illustrate the difference between the two:

```
//oo DEFINITIONS
normallist := func (param)
begin
      local tempItem;
      foreach tempItem in param
            collect tempItem;
end;
deeplylist := func (param)
begin
      local tempItem;
      foreach tempItem deeply in param
            collect tempItem;
end;



//oo SAMPLE DATA
x := {one: 1, two: 2, three: 3};

y:= {four: 4, five: 5, combo: x};

z:= {six: 6, _proto: y};

complex := {a: "first string", b: ["array1", "array2",
      {_proto: x}], c: 3.1415926};



//oo TESTS
:normallist(x)          // same
#4413441  [1, 2, 3]
:deeplylist(x)
#44137D9  [1, 2, 3]


:normallist(y)          // same
#4413A11  [4,
     5,
     {One: 1,
      two: 2,
      three: 3}]
:deeplylist(y)
#4413C49  [4,
     5,
     {One: 1,
      two: 2,
      three: 3}]


                  // For z, the answer is different.
                  // the deeply version travels proto chains!
:normallist(z)
#4416E29  [6,
     {four: 4,
      five: 5,
      combo: {#4415D79}}]


:deeplylist(z)
#4416FE1  [6,
     4,
     5,
     {One: 1,
      two: 2,
      three: 3}]


           // for this complex frame, the answers are the same
           // because there is no _proto slot in the *MAIN* frame
           // hence there is nowhere to go deeply into.
:normallist(complex)
#4418741  ["first string",
     ["array1", "array2", {#44183B9}],
     3.14159]

:deeplylist(complex)
#4418AC1  ["first string",
     ["array1", "array2", {#44183B9}],
     3.14159]
```

Compile Function (10/9/93) (Obsoleted by NS Guide 1.0 Final Documentation)

Q: I need the ability to download the validtest and endtest functions to the Newton for searches. I do not see how I can convert the incoming textual representation of the function to code to execute. How could I do this?

A: You could use the special compile function in NewtonScript. Here's an example of how to use compile:

```
begin
    local x;

    // Set x to be a function that returns the new function.
    x := Compile("Print(\"Hello\")");

   // Now call the actual compiled function.
   call x with ();

   // At this point, "Hello" will have been printed to the inspector...
end;
```

Note that compile returns a function taking no arguments whose body is the string passed to compile. If you need to create a function taking arguments, you can create a function with Compile whose return value is a function that takes arguments, like this:

```
f := call Compile("func(x) x*x") with ();
call f with (10);  // prints 100.
```

Function Definitions (10/9/93)

Q: Is it "safer" to use a return statement at the end of a function or is it just a style issue?

A: All functions return the last value in the function body. This means that you have a choice of explicitly writing a return statement, or assuming that the code reader knows that the last statement is automatically returned.

One might argue that a return statement should always be used. However there are cases where the code looks prettier with no return statement, and the return statement does generate code. Compare the following two examples:

```
begin
   ...
   if expr then "" else text
end


begin
   if expr then
      return ""
   else
      return text
end
```

Additionally, in the current NewtonScript compiler, an extra byte or two is used for the return statement, but in most cases it will make no apparent difference in speed.

Closures and Perform (10/9/93) (Obsoleted by NS Guide 1.0 Final Documentation)

Q: The NewtonScript manual indicates that the receiver during a call is the receiver that made the call. For example, the code

```
a := {
id: "a",
foo: func() Print(id),
};

b := {
id: "b",
bar: func() call a.foo with (),
};

b:bar();
```

should print "b".

When I run this in an Inspector, however, it appears that the receiver during the execution of foo() is the top level frame in which a (and b) is defined. How is the receiver actually determined? Is it always the top level of the Inspector?

A: What you really create is a closure, which is the executable code bound up with the dynamic and lexical environments at the time the function was created. That is, whatever self evaluated to when the function was compiled is what it will evaluate to when it's called.

The Perform call, or message-passing does change the closure's dynamic environment (self) to the frame that was specified by the message pass or Perform call.

To do what you seem to be attempting, you could try this:

```
a := {
id: "a",
foo: func() Print(id)};

b := {
id: "b",
bar: func() begin self.temp := a.foo; :temp() end};

b:bar();
```

Instead of :temp() above, you could have used Perform(self, 'temp, []); Perform is handy when you don't know at compile time which message you need to pass. For example, you can write Perform(self, jumpTable[i], []); where jumpTable contains an array of symbols representing methods. Perform takes a frame, a message and an array of message parameters.

Symbol Hacking (11/11/93) (Obsoleted by NS Guide 1.0 Final Documentation)

Q: I would like to be able to build frames dynamically and have my application create the name of the slot in the frame dynamically as well. For instance, something like this:

```
MyFrame:= {}; tSlotName := "Slot_1";
```

At this point is there a way to then create this ?:

```
MyFrame.Slot_1
```

A: There is a function called Intern, that takes a string and makes a symbol. There is also a mechanism called path expressions (see the NewtonScript manual), that allows you to specify an expression or variable to evaluate, in order to get the slot name. You can use these things to access the slots you want:

```
MyFrame := {x: 4} ;           // MyFrame -->{x: 4}

theXSlotString := "x" ;
MyFrame.(Intern(theXSlotString)) := 6
                              // MyFrame --> {x: 6}

tSlotName := "Slot_1" ;       // the following code creats a
                              // slot called Slot_1 in MyFrame and
                              // assigns it the number 7.
MyFrame.(Intern(tSlotName)) := 7 ;
                              // MyFrame --? {x: 6, Slot_1: 7}
```

Q: Is there a way to look for a slot programmatically starting with a slot name string?

A: Here's the code which can do this:

```
if MyFrame.(Intern(slotToFind)) exists then
```

```
            x := MyFrame.(Intern(slotToFind)) ;  // use the slot
```

Literals and run time (11/11/93)

Q: Is the literals slot really needed during run time? I noticed that I could figure out quite a lot about how my functions work from the literals used (examining the literals slot). I would rather not make it easy for others to reverse engineer certain routines.

A: The literals slot is indeed needed during run time (in the current implementation of NewtonScript) The interpreter is using it at run time. Here's an example:

```
f := func() "abc";
#4409CB9  <CodeBlock, 0 args #4409CB9>
f.literals
#4409B61  [literals: "abc"]
f.literals[0] := {foo: 'bar}
#440A1D1  {foo: bar}
call f with ()
#440A1D1  {foo: bar}
```

Don't assume anything about the implementation of the literals slot in future Newton system software.

Missing Semicolons cause Weird Errors (12/11/93) (Obsoleted by NS Guide 1.0 Final Documentation)

As with Pascal, semicolons are statement separators, not terminators. If you forget to add a semicolon at the end of the Newtonscript statement, the interpreter will try to interpret a larger statement than intended, causing strange error reports.

The following is an example of such a case.

You are getting an "expected array, frame, or binary but got 8" exception because you wrote the following code:

```
:Emit(dest, 'subprim, "Send", [t2, t1, Length(node.r3)])
 :PopTemp('ref);
```

Note the missing semicolon on the first line. This is parsed as

```
(:Emit(blah blah)):PopTemp('ref);
```

The :Emit call resulted in an 8, which was used as the receiver for the PopTemp message. This was NOT what the programmer intended.

There are two habits that would have prevented this error. You might want to consider them as part of your personal coding style, though neither of them is extremely important since this error is pretty rare.
- Always use "self:foo()" instead of ":foo()".
- Always put a semicolon at the end of a statement, even when you don't need to.

Classical OOP programming withNewtonScript (12/17/93)

(Obsoleted by NS Guide 1.0 Final Documentation)

Q: I would like to see the ability to use more of a C++ approach, where I could inherit code — not necessarily user-interface (view) oriented code — with functionality like templates.

A: You can easily simulate most aspects of traditional class-instance OO programming (as in Smalltalk or C++) using NewtonScript. There aren't any specific features in NTK 1.0 to support this, but you can do it without too much trouble, using the Project Data window.

The basic idea is to use _parent inheritance to provide the link between "instances" and "classes". You define a frame in the Project Data to hold the methods (this is the "class"), and you construct "instances" whose slots are the "instance variables" (or "members" if you prefer C++). Quotes will continue to be used around these words to emphasize that there is no built-in concept of a class in NewtonScript; we are using the prototype-based inheritance system to simulate class-based inheritance.

Suppose we want to create a "class" to represent a stack. We will represent the stack as an array. Each "instance" needs to store the array and the index of the top element of the stack, so an "instance" will look like this:

```
{ _parent: ...,                  // pointer to the "class"
  items: [...],                  // array of items
  topIndex: 2 }                  // index of top item
```

The "class" contains the methods, like so:

```
Stack :=
      { Push:     func (item) begin
                          topIndex := topIndex + 1;
                          items[topIndex] := item;
                          self
                      end,

              Pop:      func () begin
                          if :IsEmpty() then
                              NIL
                          else begin
                              local item := items[topIndex];
                              items[topIndex] := NIL;
                              topIndex := topIndex - 1;
                              item
                          end
                      end,

              IsEmpty:  func ()
                          topIndex = -1,

              Clone:    func () begin
                          local new := Clone(self);
                          new.items := Clone(new.items);
                          new
                      end,

              New:      func ()
                          {_parent: self,
                           items: Array(10, NIL),
                           topIndex: -1}
             }
```

Obviously, this ignores such niceties as stretching the array and signalling error conditions, for the sake of clarity.

To get a new Stack "instance", you send the New message to the "class":

```
        s := Stack:New();
```

Now you can send messages to s to get work done:

```
        s:Push('a)
        x := s:Pop()
        if s:IsEmpty() then ...
        s2 := s:Clone()
        // etc.
```

You can put "class variables" and "class methods" in the "class" if you want; they're all the same thing in this model, just slots of the "class". The New method is an example. Note that you can get a new instance from an old instance (i.e., s:New()) because of this unity.

Those who have been paying attention to the Listener output may already have noticed this method of programming, because it is used by soups, stores, and cursors. (But you should assume that anything undocumented you notice in the Inspector may be changed in the next ROM.)

To use this technique in NTK, you can put the "class" in the Project Data window. To access it from your code, you'll need to put a slot that refers to it in your main view (or lower down if you don't need the class throughout your application). You can just add a slot called "Stack" whose value is "Stack", or if you prefer, you can use distinct names (for example, call it "StackClassFrame" in Project Data).

Here's an example, in NTK text-export form. This does not include the project data section, because it looks just like the "Stack :=" section above. This is just an application with a button that prints "30" in a complicated way using a stack object.

```
   main :=
      {title: "xxx",
       viewBounds: {left: -1, top: 0, right: 236, bottom: 333},
       Stack: Stack,
       _proto: protoapp,
      };

   _view000 := /* child of main */
      {text: "Button",
       buttonClickScript:
         func()
         begin
            local s := Stack:New();
            s:Push(10);
            s:Push(20);
            Print(s:Pop() + s:Pop())
         end,
       viewBounds: {left: 74, top: 74, right: 154, bottom: 106},
       _proto: prototextbutton
      };
```

You can use an extended form of this technique to put an object-oriented interface on soup entries. "Wrap" the soup entry in an "instance" frame like the one above by pointing a slot of the "instance" to the soup entry. You can have "class methods" to do queries and return these "instances". You can also have "instance methods" to change, validate, undo, flush, and so on. You can also choose _proto as the slot that points to the soup entry, so you inherit the entry's slots--but remember that slot assignments will go into the "instance", not the entry.

If you're intrigued by all this, see the various papers on the language "Self", by Ungar, Chambers, and others at Stanford (now at SunSoft). Many aspects of NewtonScript were inspired by Self. The postscript files are available via ftp on Internet from self.stanford.edu.

Only 7-Bit ASCII In NewtonScript Symbols (1/26/94) (Obsoleted by NS Guide 1.0 Final Documentation)

Q: My frame has some slot symbols that contain accent and special letters. For example:

```
f := {|à|:"help me please", |é|:"Thanks in advance", ...,...}
```

My program received a letter from ProtoInputLine and turned it into a symbol. I have tried many different ways to do this but I could not convert it into a symbol.

A: The basic problem is you can only use 7-bit ascii in symbols.

How to Avoid _parent Problems (6/28/94)

Q: I read somewhere that developers should never access a view _parent slot directly? Are there any cases when it is safe to access the _parent slot?

A: In most cases, you should use the :Parent() view method to determine a parent view. However, there are some cases when you want to access the _parent slot of a view directly. The situation which is bad is to use _parent as a simple variable because results will vary depending on implementation of the current function. Some guidelines:

`_parent`	is bad
`view:Parent()`	is OK
`self._parent`	is OK (and should be the same as :Parent())
`view._parent`	is OK (and should be the same as view:Parent())

Symbols vs Path Expressions and equality (7/11/94)

Q: While trying to write code that tests for the existance of an index, I tried the following, which did not work. How can I compare path expressions?

```
if value.path = '|name.first| then ...
```

A: There are several concerns. `'|name.first|` is not a path expression, it is a symbol with an escaped period. A proper path expression is either `'name.first` or `[pathExpr: 'name, 'first]`. The vertical bars escape everything between them to be a single NewtonScript symbol.

The test `value.path = 'name.first` will always fail, because path expressions are deep objects (essentially arrays) the equal comparison will compare references rather than contents. You will have to write your own code to deeply compare path expressions.

This code is further complicated by the fact that symbols are allowed in place of path expressions that contain only one element, but the two syntaxes produce different NewtonScript objects with different meanings. That is, `'name = [pathExpr: 'name]` will always fail, as the objects are different.

A general test is probably unnecessary in most circumstances, since you will be able to make assumptions about what you are looking for. For example, here is some code that will check if a given path value from a soup index is equivalent to `'name.first`.

```
   if classof(value.path) = 'pathExpr and length(value.path) = 2
      and value.path[0] = 'name and value.path[1] = 'first then ...
```

Function Size and "Closed Over" Environment (7/18/94)

Q: I want to create several frames (for soup entries) that all share a single function, but when I try to store one of these frames to a soup, I run out of memory. Can several frames share a function and still be written to a soup? My code looks like this:

```
   ...
   local myFunc := func(...) ...;
   local futureSoupEntries := Array(10, nil);
   for i := 0 to 9 do
      futureSoupEntries[i] := {
         someSlots: ...,
         aFunction: myFunc,
      };
   ...
```

A: When a function is defined within another function, the lexically enclosing scope (locals and paramaters) and message context (self) are "closed over" into the function body. When NewtonScript searches for a variable to match a symbol in a function, it first searches the local scope, then any lexically enclosing scopes, then the message context (self), then the _proto and _parent chains from the message context, then finally the global variables.

Functions constructed within another function, as in your example, will have this enclosing lexical scope, which is the locals and parameters of the function currently being executed, plus the message context (self) when the function is executed. Depending on the size of this function and how it's constructed, this could be very large. (Self might be the application's base view, for example.)

A `TotalClone` is made during the process of adding an entry to a soup, and this includes the function body, lexical scopes, and message context bound up within any functions in the frame. All this can take up a lot of space.

If you create the function at compile time (perhaps with `DefConst('kMyFunc, func(...) ...))` it will not have the lexically enclosing scope, and the message context at compile time is defined to be an empty frame, and so cloning such a function will take less space. You can use the constant `kMyFunc` within the initializer for the frame, and each frame will still reference the same function body. (Additionally, the symbol `kMyFunc` will not be included in the package, since it is only needed at compile time.)

If the soup entries are only useful when your package is installed, you might consider instead replacing the function body with a symbol when you write the entry to the soup. When the entry is read from the soup, replace the symbol with the function itself, or use a `_proto` based scheme instead. Each soup entry will necessarily contain a complete copy of the function, but if you can guarantee that the function body will always be available within your application's package, it might be unnecessarily redundant to store a copy with each soup entry.

CHANGED: If Statements & Constant Condition Errors (4/5/95)

Please refer to the NTK section, entry "If Statements & Constant Condition Errors"

NEW: Assignment with path expressions as lValues (4/12/95)

Q: Sometimes it seems like I can write to ROM. For example, `f := {_proto: {a: {b: 0}}` where the _proto slot `(a: {b:12}}` is actually a frame in my package, I can do `f.a.b := 12`. I expect -48214 "object is read only" but it works!?

A: You're not really writing to protected memory. There is a bug in the way NewtonScript handles assignment when the lValue is a path expression.

As defined, assignment with a path expression on the left will create intermediate frames if necessary:

```
   f := {};
   f.a.b := 12;
   f
   #4411509  {a: {b: 12}}
```

However, the _proto or _parent chains in the lefthand expression are not taken into account, and may be overridden. This may cause other slots that would normally be found via _proto or _parent inheritance to be masked.

```
   f := {_proto: {a: {b: 0, c: 1}}};
   f.a.b := 12;
   f
   #4415E99  {_proto: {a: {b: 0, c: 1}},
              a: {b: 12}}

   f.a.c;   // expected result is 1
   #2        NIL
```

You can work around this by avoiding long path expressions as lValues in assignment statements. To do this, use temporary variables to force the intermediate path to be evaluated prior to assignment:

```
   f := {_proto: {a: {b: 0, c: 1}}};
   x := f.a;   // force evaluation of f.a
   x.b := 12;  // would throw -48214 if f.a was in protected memory
   f
   #4411899  {_proto: {a: {b: 12, c: 1}}}
```

End of DTSQA

WORKING WITH PROTO TEMPLATES

TABLE OF CONTENTS:

WORKING WITH PROTO TEMPLATES...1
Introduction...1
Creating a viewFormat Slot in a ProtoLabelInputLine (10/15/93)..1
Introduction...2
protoTable versus protoTextList (10/8/93)...2
Deselecting in protoTable and protoTextList (10/8/93)...3
Expando your horizons... (10/8/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 3-10 to 3-29)...3
protoTable Ghost Drawings (11/11/93)...4
protoClock, another Clock (12/10/93)...4
User Protos (NTK) and System Protos (12/19/93)...4
ProtoLabelInputLine Lining (12/19/93)...5
protoPictRadioButton, Highlighting of Radio Button (1/26/94) (Obsoleted by Newton Programmers Guide 1.0 Documentation 3-48,9)...6
protoExpandoShell
ExpandNone (4/28/94)...7
protoExpandoShell
ExpandLine (4/28/94)...7
Programmatically Updating protoPictIndexer (6/9/94)...7
Programmatically Highlighting a protoTable Entry (6/15/94)...8
Stopping protoFolderTab highlight remains (6/17/94)...8
Fixing overview for protoRoll with large roll items (8/11/94)...8
Resetting RadioCluster to nothing chosen (8/18/94)...9
valueChanged called for other reasons (9/13/94)...9
fix for protoPhoneExpando does not expand keyboard (10/18/94) .9
protoRoll needs Visible viewFlag set (12/16/94)...9
protoRoll printing problems (1/16/94) 10

Introduction

This document addresses Newton Proto Template 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.

Creating a viewFormat Slot in a ProtoLabelInputLine (10/15/93)

Q: When I create a viewFormat slot for my LabelInputLine, the lines disappear. What's happening here?

A: When you add the viewFormat slot, make certain the "Lines" value is set to "None". Any other setting (including, unfortunately, the default setting of "Grey") causes the lines to disappear.

How to Stop autorepeat on Keys (10/8/93)

Q: Every key on a protoKeypad autorepeats. How do I stop this from happening?

A: Here is a piece of code you can use for your keyPressScript that will 'disable' autorepeat

```
keyPressScript: func (key)
begin
  if key == lastKey and (Ticks() - lastKeyTime) < 15 then begin
      // ignore repeated keys
  end
  else begin
      // handle the keypress
  end;
  lastKey = key;
  lastKeyTime = Ticks();
end;
```

NOTE: lastKey and lastKeyTime need to be slots in the keyview. Initialize them as follows:

```
lastKey:          EvaluateSlot   nil
lastKeyTime:      NumberSlot     0
```

protoTable versus protoTextList (10/8/93)

Q: What is the difference between protoTable and protoTextList?

A: I am not sure of all the differences, but basically a protoTextList only has one child for all the items. It uses one big clParagraph (well, clText that is, a read-only paragraph) that has the items of the text list separated by newlines. A protoTable has one view per entry in the table.

This means that in a protoTable each entry can have a click script that does something based on the entry (though each entry must be of the same prototype). protoTable uses LayoutTable to layout the items. You can have multiple column tables. The table definition that you pass can have the same type of slot values that LayoutTable can (see 2-96 in the NPG). protoTable also keeps track of the currently selected text in a slot (currentSelection).

You can define a buttonClickScript in your protoTextList. This script can determine what was clicked on and take the appropriate action.

Basically, use protoTable if you want to have a number of "buttons", or a multi-column table, or if you want the current selection tracked. Otherwise use protoTextList. Note that protoTextList may be drawn faster than protoTable since it only has one child.

Deselecting in protoTable and protoTextList (10/8/93)

Q: Question: In a "ProtoTable" or "ProtoTextList" is it possible to have a tap toggle the selected state of a line?

A: See the tables and lists sample.

Expando your horizons... (10/8/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 3-10 to 3-29)

Here are some general notes on protoExpandoShell and the expando items you can put in it. You should also look at the following sample codes:

numberExpando - shows how to create an expando item that inputs and displays numbers
dateExpando - shows how to use the dateExpando item
phoneExpando - shows how to use the phoneExpando item
shellGame - 2 expandoShells side by side, items expand and shrink together
twoLineExpando - shows how to do an expando that has a 2-line label

A protoExpandoShell has a slot called lines that is an array of items that are editable. It has a related slot called numLines that specifies how many entries from the lines array to use. Each of those items must be based on an expando item prototype. There are three of these, protoTextExpando, protoDateExpando and protoPhoneExpando. Each item has two important common messages:

setup1(target) - returns a string to use for the label in the collapsed state
setup2(target) - returns a string to use for the value in the collapsed state

target is either the target specified in the protoExpandoShell or nil.

Items must have a path slot. This is used in combination with the target slot of the protoExpandoShell to get and set the value of the expando item. The path slot can contain a symbol or a path expression. The slot edited is found like this:

```
item := target.(path);
```

Items can also have a specialClass slot. This is a symbol that is used to set the class of target.(path) (It is used in a setClass statement.)

When an expando closes, a script called CloseEdit(itemIndex) is called. You could use this to detect situations where the user finishes a field by clicking in the close box.

The protoTextExpando is the most generic expando item. It is based on a protoLabelInputLine, so if you need to access the text, it is in the entryLine.text slot. In general, you will probably create your own user prototype expando items using protoTextExpando (as the sample code numberExpando does).

protoDateExpando is used for dates.

protoPhoneExpando expects the target.(path) to be an array of strings (or an empty array). Each string may have a special class that corresponds to the type of phone number (home, work, ...).

protoTable Ghost Drawings (11/11/93)

Q: I have noticed in protoTable that when selecting a cell (in columns 3 or 4) in a table with more than one column, a selection rectangle appears as a ghost off the right edge of the screen.

A: We have seen this problem before and we have reported it as a bug. The workaround is to set vClipping on for your protoTable view. That should eliminate any highlighting outside the table.

protoClock, another Clock (12/10/93)

Q: If I attempt to scale the protoSetClock, the hands scale but the clock face does not. What should I do?

A: You could insert another, different clock picture into the icon slot; this works fine with regard to scaling. The other things you need to do are:
• add justification to FullH/FullV
• keep the viewBounds square

Unfortunately the clock hands will always draw with the same pen size, so large clocks will look weird with the originally sized hands.

User Protos (NTK) and System Protos (12/19/93)

Q: I'm trying to add dynamic behavior at run time using PT_protoFileNames (NTK User Protos). I would like to store references to different behaviors in the slot of a view, and at run time change the value of the view's _proto slot so it points to these other behavior protos. For example:

```
doSortBehavior := func()
begin
     self._proto := PT_protoSortBehavior;
end;
```

The problem is that NTK tells me that PT_protoSortBehavior is an unrecognized variable. If I create a view on the fly and set it's _proto slot to PT_protoSortBehavior it works fine, however. It seems that PT_ syntax is only valid during compile time.

A: If the literal is the name of a system proto, the Newton run time uses the proto that is stored in ROM. If the proto is not a sysem proto, the proto is found by name from the current context.

NTK creates a slot in your application base view with the name:
PT_ <name of user proto file>

Because of the relationship between your subview and your base view, the pt_ constant works because the system is doing a slot lookup in your application base view.

The easiest way to disturb this relatioship is to violate the _parent chain between the current context and the base view. For instance, if you dynamically create a frame like {_proto: PT_myuserProto} it will work if it's run from a real subview. But if you call its viewSetupFormScript and that uses a pt_user proto, then this will not be able to access the proto because it can't follow the parent chain to reach the base view.

One possibility is to specify a particular proto, and override one method in order to change the behavior.

ProtoLabelInputLine Lining (12/19/93)

Q: We have problems with protoLabelInputLine concerning lining up the label and value. We are using simple 9 bold as the label and 12 fancy as the input text. If we lock the field it shifts the input text to high; if unlocked, it's too low.

A: The protoLabelInputLine is actually just a clView with two children, labelLine and entryLine. labelLine and entryLine are both clParagraphViews, but the clParagraphView view class has a "feature" whereby much of the class complexity is turned off if the view has no input (for example, if it's readOnly and has no recognizers on.) This "lightweight" clParagraphView does NOT respect the viewLineSpacing setting, and so the text is placed at the (default) top left edge of the view.

A standard clParagraphView that does not accept input but does respect viewLineSpacing can be created by simply turning off vClickable flag, but leaving the recognition flags set. We recommend using this combination instead of just vVisible + vReadOnly for your no-input entry lines.

To get the labelLine lined up with the entryLine, it's necessary to change the viewBounds for the labelLine. The best place to do this is probably in the viewSetupChildrenScript. Here's an example viewSetupChildrenScript that lines up the labelLine with the baseline of the entryLine. (Note that the viewLineSpacing field must also be set in the protoLabelInputLine.)

```
func()
begin
      inherited:viewSetupChildrenScript();
      labelLine.viewBounds := RelBounds(0, viewLineSpacing-
                        FontHeight(labelFont),
                              indent, viewLineSpacing);

      entryLine.viewFlags :=  vVisible + vCharsAllowed +
                        vGesturesAllowed;
      // for editable entryLines, add the vClickable flag.
end
```

It is better (more appropriate) to set the entryLine viewFlags in the viewSetupChildrenScript for the protoLabelInputLine *after* the inherited viewSetupChildrenScript is called.

You should probably turn this all into a user proto and use that instead of protoLabelInputLine to save memory.

protoPictRadioButton, Highlighting of Radio Button (1/26/94) (Obsoleted by Newton Programmers Guide 1.0 Documentation 3-48,9)

Q: When I select a button, protoPictRadioButton displays a box in the button's view. I would like to change the box size to accommodate it to the view size, but I do not know how.

I tried to draw a box of the size I need using inherited:viewClickScript(), according to the manual. However, the default box was unexpectedly displayed, too.

Then I used viewClickScript without inherited keywords and was able to draw the box I needed without a default box being displayed. This was not what was said in the manual, though. If I use this procedure, will I cause a problem?

A: Because of a bug in this particular prototype, you must provide the code to highlight the radio button when it is selected. You could do this by drawing an inner black border. Your code should begin like this:

if viewValue then begin //do drawing here...

Here's an example:

```
pictRadio := {...
  _proto: protoPictRadioButton,
  viewFormat: vfFillWhite+vfFrameBlack+vfPen(1), //override frame
  icon: myPict,
  buttonValue: 3,
  viewDrawScript: func()
          begin
          if viewValue then  // if button is selected then
                      // highlight it
                 :DrawShape(MakeRect(0,0,15,15), nil);
          end,
...}
```

In this example, you would change the size of the rectangle MakeRect produces in order to change the size of the area highlighted.

protoExpandoShell:ExpandNone (4/28/94)

NPG 1.0 documentation does not include this protoExpandoShell message. You can send this message to close the currently expanded item in a protoExpandoShell.

protoExpandoShell:ExpandLine (4/28/94)

NPG 1.0 documentation does not include this protoExpandoShell message. The syntax is:

```
protoExpandoShell:ExpandLine(<lines-index>)
```

<lines-index> an integer index into the lines array of the protoExpandoShell

Close any currently expanded edit item and expand the item corresponding to lines[<lines-index>]
Closes any currently expanded edit item and expands the item corresponding to lines[<lines-index>]

Programmatically Updating protoPictIndexer (6/9/94)

Q: How do I update the display of the indexed item in a protoPictIndexer?

A: You can use the hiliter and unhilite messages of the protoPictIndexer:

hiliter(index)
- toggles the highlight of the index'th element. For example, if you were using an A to Z indexer, hilite(0) toggles the highlight on A. This MUST be called in a :DoDrawing.
 NOTE: this does NOT change the curIndex setting.

unhilite()
- unhighlights the current selection and sets curIndex to -1.

A possible function to set the highlighted item of a protoPictIndexer is:

```
protoPictIndexerView.Seta2zHilite := func(newIndex)
begin
   if newIndex <> curIndex then
```

```
         begin
              :DoDrawing('hiliter, [curIndex]) ;
              curIndex := newIndex ;
              :DoDrawing('hiliter, [curIndex]) ;
          end;
      end
```

Programmatically Highlighting a protoTable Entry (6/15/94)

Q: How do I highlight an item or items in a protoTable?

A: You can use the selectedCells slot in combination with RedoChildren and the UpdateSelection message to highlight the cell or cells you need.

As an example, if you wanted to highlight the item in your list that corresponded to the third element in the array (array index 2), you would use these 3 lines of code:

```
theTable.selectedCells := [2] ; // setup the cells you want selected
theTable:RedoChildren(); // clear the current selection
theTable:UpdateSelection(): // hilite the selected cells
```

Stopping protoFolderTab highlight remains (6/17/94)

Q: Using protoFolderTab, when the user changes from a long folder name to a short name, some pixels from the highlighting of the longer name are left black when the popup closes. How can I make this look nice?

A: Add a viewFlags slot to the template based on protoFolderTab, and turn on vClipping (and vVisible). This will prevent the stray pixels from appearing.

Fixing overview for protoRoll with large roll items (8/11/94)

Q: I have a protoRoll that has roll items larger than the roll itself. If the user clicks in the overview button when they are in the middle of a large roll item, the roll appears to be blank, or worse, weird stuff is drawn.

A: protoRoll currently uses SyncScroll to scroll its children. If a roll item is larger than the protoRoll, and the user scrolls into the middle of the item, this will set the viewOriginY slot of the protoRoll to a non-zero value. If the user now taps the overview button, the viewOriginY value is not reset to 0.

The solution is to reset the viewOriginY to 0 when an overview occurs. You can do this by giving your protoRoll a viewOverviewScript:

```
viewOverviewScript.func()
begin
   if not allCollapsed then
      viewOriginY := 0 ;
   inherited:?viewOverviewScript() ;
end;
```

Resetting RadioCluster to nothing chosen (8/18/94)

Q: I have a view containing radio button clusters in which the values are set according to information in a soup. If I try to set all the radio buttons as unselected with `SetClusterValue`, I get no errors, but also the radio buttons are not reset to have none selected when passed a value of `nil`.

How can I programmatically make a radio button cluster have none of the buttons set?

A: You should use another (previously undocumented) method on `protoRadioCluster` called `InitClusterValue`, which takes a single argument like `SetClusterValue`, but will also accept `nil`. It is intended to be used to initialize a cluster, and so the `ClusterChanged` method will not be called when `InitClusterValue` is used.

valueChanged called for other reasons (9/13/94)

Q: I have defined a valueChanged script for my checkbox, but it appears to be called even though the the viewValue has not changed.

A: Correct. The valueChanged script can be called for any change that would cause viewChangedScript to be called. That is, using a SetValue to change the value of a slot of your checkbox (like changing the value of the text).

fix for protoPhoneExpando does not expand keyboard (10/18/94)

Q: Even though I assign a value of 'phoneKeyboard to the keyboard slot of a protoPhoneExpando, the phone keyboard will not automatically open. What am I doing wrong?

A: The protoPhoneExpando is not what you think. The automatic keyboard opening is actually a feature of protoTextExpando. But protoPhoneExpando is not based on a protoTextExpando (though protoDateExpando is). You need to add a textSetup method to your phone expando:

```
func()
begin
   keys := nil;
   if self.keyboard then
   begin
      keys := GetVar(self.keyboard);
      AddDeferredAction(ExpandKeys, [self.entryLine]);
   end;
   inherited:?TextSetup();  // this method is defined internally
end
```

protoRoll needs Visible viewFlag set (12/16/94)

Q: I dragged out a protoRoll inside a protoApplication, but the protoRoll does not show up. What am I doing wrong?

A: You are not doing anything wrong. Unfortunately, the Newton Programmers Guide (1.0) does not mention that the Visible viewFlag is not set by default. You need to add a viewFlags slot to the protoRoll and check Visible. Then it will show up by default.

protoRoll printing problems (1/16/94)

Q: I have been having a problem with printing, in which only the top half of the page is being printed. My print format consists of a protoPrintFormat containing one child, which is a descendent of protoRoll.

After the completion of the protoRoll's viewSetupFormScript, and before the call to viewSetupChildrenScript, viewBounds is being clipped to a height of 310, apparently as a result of the vApplication flag. When the flag is turned off, the viewBounds is left at the proper 720 pixels. What is wrong?

A: Because protoRoll usually scrolls when on the screen, it has the vApplication viewFlag set. However, on some ROMs, specifically on the MessagePad 110, code exists that resizes applications which are too large for the MessagePad 110 screen. (This screen was slightly smaller than its predecessors.) The code which checks for the vApplication flag exists for both on-screen and printing style imaging, although with that flag set, the code is only necessary for on-screen views .

For descendents of protoRoll that are going to be used in printing, clear the vApplication flag in NTK, or change the flag at run time if the proto will be used in both places, using the functions BAND and BOR in the viewSetupFormScript.

End of DTSQA

ROUTING

TABLE OF CONTENTS:

ROUTING..1
Introduction..2
Were Routing for You (Routing for the Newton).......................................2
Sending Frames, What Application is Opened at the Other End? (11/11/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-13) 12
Landscape Printing, Currently Not Supported (12/18/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-7)..........................13
Item Titles, Beaming (12/19/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-7) 13
Printing and the Application Not Present (12/19/93)...........................14
Changing Print Formats on the Fly (12/19/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-21)..14
How do I get the triangle in the filing button for card items (12/20/93) 14
How do I get a default name in the faxSlip (or elsewhere) (12/22/93) 15
Printing Resolution 72DPI/300DPI (2/8/94)..16
Not all Drawing Modes Work with a Postscript Printer (3/8/94).......16
Printing Does Not Have Access to the App Frame or Globals (6/21/93) 16
Title in the In/Outbox Does Not Wrap (6/3/94)...16
Out of Memory When Adding Items to the Outbox (6/7/94)...............17
PICT Printing Limitations (6/9/94) 17
Fax Timeout & Banding (6/21/94)...18
How to Improve Print & Fax Speed when Banding (6/21/94) 18
How to Determine the Drawing/Banding Region (6/21/94)..................18
Do Not Use the SetupTarget Mechanism (6/29/94).................................19
TargetView Does Not Get Routing Messages (6/29/94)..........................19
Checking for Fax Sent (7/6/94) 19
Printing Special System Fonts with a PostScript Printer (7/26/94)..20
NewtonMail APIs (9/14/94) 20
Updated
Checking the ROM Language Environment for MungePhone (2/16/95) 21

Introduction

This document addresses Newton Application Design 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.

Were Routing for You (Routing for the Newton)

(Obsoleted by Newton Programmers Guide 1.0 Documentation Section 13)

version 0.3 Jun 8, 1994

Welcome to the wonderful world of routing. This document describes how your application can support the action button (protoRoutingSlip) for duplicating, deleting, moving soup data to and from cards, printing, faxing, beaming and mailing.

The amount of work you need to do depends on what type of routing you want to support. Duplicating, deleting and moving soup data to and from cards are on the simplest level to support. We suggest you implement these first, and then do the rest, a bit at a time. Beaming and mailing are more difficult; printing and faxing are the hardest, though not nearly as difficult as supporting Mac printing.

This interim doc has 7 sections: 1) What every application needs, 2) DDC (Deleting, Duplicating, Carding), 3) Beaming, 4) the other routing categories, 5) Routing Formats, 6) Banding Issues, 7) Faxing and timeout

1. What Every Application Needs

To support routing, every application needs to have the following things

`appSymbol`

(slot) A symbol that is the same as the one assigned by NTK (the Symbol part of the project preferences dialog). for example,

```
appSymbol: wiggyRoutingApp
```

`target [or :setupTarget(slip) -- see correction below]`

(slot or message) The target should be the data to be routed. In general this will be a soup entry (that is, a frame that comes from a cursor:Entry() call). The target should point to the soup entry for the current thing your application is editing. It is the item to be routed (deleted, duplicated, printed...). This slot is also required for filing (see relevant documentation).

Note that for certain routing categories (for example, printing, faxing) you manually set up the target and your app will be the only thing which uses it. That way you can set up an array of things to print.

The setupTarget message is sent one argument — the information slip used by the type of routing. The function should set the target and targetView slots of the slip:

```
SetupTarget: func(slip)
begin
      slip.target := :GetMyCurrentTarget() ;
      slip.targetView := self ;
end
```

In general, you will only use setupTarget if you have to do something special to find the target. Otherwise, use the target slot.

[**SetupTarget CORRECTION - 6/29/94**: Although the current documentation specifies that your base view's SetupTarget method can be implemented to do custom setup of a target, this is incorrect. The setupTarget mechanism is not fully supported in the current Newton ROMs and we recommend that you avoid currently implementing SetupTarget. SetupTarget is not consistently called and the only way to force its use with a protoAction button (that is, have NO target slot exist) will throw exceptions from the current protoActionButton. Note: the "Do Not Use the SetupTarget Mechanism" Q&A also includes this information.]

`targetView`

[**targetView CORRECTION - 6/29/94**: The targetView is used by several parts of the Newton system, including routing and filing. When your application registers your routing frame your base view will actually get routing messages, which in some cases include the target and the targetView as parameters. For instance, if you want specific views to be sent messages for delete, your base view's deleteActionScript could send a message to the view represented by targetView, which will be the second argument to deleteActionScript. Note: the "TargetView Does Not Get Routing Messages" Q&A also includes this information.]

`setupRoutingSlip(fields)`

This message is used to setup a text description in the OutBox entry of your routed item. The `fields` argument contains a slot called `title`, that you set to the string you want to appear in the OutBox as a description of your item (for example, Notepad, Fri 1/1/ 11:00 am).

This is also a place to put some data in the fields variable that your print and fax layouts can use to find the information used in printing/faxing. This is easy to do because layouts have access to the fields variable. You can put a slot in fields that lets your layouts get at the data they need. Keep this information small. The best is something like a key that can be used for a soup query.

An example function would be:

```
func(fields)
begin
   // put reference to my data in fields so my
   // print/fax formats can get at it
   fields.myKey:= target.keySlot ;
   fields.title := "WiggyApp," && DateNTime(Time());
end
```

Warning: The fields variable may be TotalCloned after this message is done. References to your application or ROM will cause the Newton to hang, at best.

A routing frame

This is a frame that contains the types of routing your application can do. Your application must add this frame to the global routing frame in a slot with the same symbol as your appSymbol. The following code appears in the InstallScript text document that is in the same directory as your pacakge:

```
//a simple routing frame
// NOTE in NTK, this would be a slot in your application
wiggyRouteFrame := {
   // something for printing
   print: {
      title: "Print Wiggy",
      routeForm: printSlip,
      formats: [wigBusinessFormat, wigMemoFormat]
   },

   // put a spacer in the popup menu
   spacer: nil,

   // something for deleting
   delete: {
      title: "Delete",
      routeScript: deleteActionScript
   }
};

// script run when package is installed
// NOTE: you can also do this from
// your viewSetupFormScript

const kAppSymbol := '|wiggyApp:WIGGY| ;
```

```
InstallScript := func(partFrame)
begin
   // put application routing frame into the
   // global routing frame
   Routing.(kAppSymbol):=
      partFrame.theForm.wiggyRouteFrame;
end;

// script run when package is removed from Newt
// you might also remove formats (see below)
DeleteScript := func(partFrame)
begin
   // remove application routing frame from
   // global routing frame, 'wiggyApp is the
   // application symbol
   RemoveSlot(Routing, kAppSymbol);
end;
```

Note: When you click on the action button (protoActionButton), you get a selection menu that is constructed from this frame. Each item is taken in order to construct the menu. A `nil` item is a spacer bar in the menu.

Note that routing will read the global routing frame and that kAppSymbol is the appSymbol for your application. Notice how the application removes its routing frame when it gets removed. This ensures your application will clean up after yourself and avoid the "the Newton still needs the card" message when you eject a card with your application on it.

Also Note: Your routing frame must use the following symbol/service pairs:

Routing Service	routing frame symbol
Printing	print
Faxing	fax
Mailing	mail
Beaming	zap

2. DDC (Deleting, Duplicating, Carding)

To support DDC you need to have all the materials from section1 above, as well as support scripts for deleting and/or duplicating. Carding requires only a special entry in the application routing frame.

Deleting

To support deleting you need to add an entry to the routing frame for deleting, a delete script to your application to receive the delete route and, optionally, a real data delete script .

`Routing Frame Entry`

The entry for deleting has this form:

```
delete: {
   title: "Delete",
   routeScript: deleteActionScript},
```

The user interface convention is for the title to just say Delete since this is not likely to be the first item in the routing frame. The routeScript is a symbol for a script that you define in your application. You can call the script whatever you want, deleteActionScript is just a convention.

`deleteActionScript(target, targetView)`

The name of the delete script must correspond to the name you specified in the routeScript slot of the routing frame entry for delete. The two arguments are:

target: the target frame, as specified by your application targetView: the target view as specified by your application

In most cases your delete script will call send the Delete message to a view. This causes the target view to do the crumple animation and sends a message to the view to delete the real data.

If you are not using your deleteActionScript to send a Delete message to a view, then the deleteActionScript should do any required data deletion.

An example delete script might be:

```
deleteActionScript: func(target, targetView)
begin
   // send a :Delete message to the correct view
   // so that it crumples.
   targetView:Delete(doDeleteAction,
                         [target,targetView]);
end
```

Note: The Delete view message syntax is: Delete(message, args);

message: a symbol for the message to call

args: an array of arguments to pass to the message

Real Data Delete Script

You only need to define this script if your *deleteActionScript* calls it with the Delete view message. The arguments are whatever you define them to be (see the Note above). The script will probably delete your real data. An example would be:

```
doDeleteAction: func(entry, entryView)
begin
   EntryRemoveFromSoup(entry);
   entryView:drawFromSoup();
end
```

Duplicating

To support duplicating you need to add an entry to the *routing frame* for duplicating, a *duplicate script* to your application to receive the duplicate route.

Routing Frame Entry

The entry for duplicating has this form:

```
duplicate: {
   title: "Duplicate",
   routeScript: duplicateActionScript},
```

The user interface convention is for the title to just say Duplicate since this is not likely to be the first item in the routing frame. You can call the script whatever you want, duplicateActionScript is just a convention.

duplicateActionScript(target, targetView)

The name of the duplicate script must correspond to the name you specified in the routeScript slot of the routing frame entry for duplicate. The two arguments are:

target: the target frame, as specified by your application

targetView: the target view as specified by your application

In general, your duplicate script will put a new entry into the soup that is a duplicate of the current entry. You may also want to add a new view for the new data item and set it as the current view.

An example duplicate script might be:

```
duplicateActionScript: func(target, targetView)
begin
   // make a copy of the target
   local newItem := copyWiggy(target);

   // update the view
   targetView:drawFromSoup();
end
```

Carding

To support carding you need to add an entry to the *routing frame* for carding, a *card script* to your application to receive the card route.

`Routing Frame Entry`

The entry for duplicating has this form:

```
card: {
   GetTitle: func(item)
   begin
      if item and length(GetStores()) > 1 then
      begin
         if entrystore(item) = GetStores()[0] then
         begin
            "Move to card";
         end
         else begin
            if EntryStore(item):IsReadOnly() then
            begin
               "Copy from card";
            else
               "Move from card";
         end;
      end;
      else
         nil;
   end,

   routScript: cardActionScript},
```

The GetTitle function is used to assign a title to the item in the list of possible routing actions. The nil means that no entry will be present.

The routeScript is a symbol for a script that you define in your application. You can call the script whatever you want, cardActionScript is just a convention.

However, the root view contains a cardActionScript that will move your data and then do a BroadcastSoupChange message. This means that if you do not define a cardActionScript for your application, the default one will be used — less work for you.

Better still, you do not need to type most of the code in. The card frame exists in ROM and can be accessed by the constant `ROM_cardaction`. That means the card entry in your routing frame will look like this:

```
card: ROM_cardaction
```

`CardScript(target, targetView)`

If you really want to define this script, it should move/copy the target from its current soup to or from the card. The GetTitle function above shows you how to determine where the current entry is. You could use that function and insert the appropriate code where the strings are returned.

target: the target frame, as specified by your application

targetView: the target view as specified by your application

3. Beaming

Support for sending beams is trivial; more work is required to support receiving incoming data.

`Routing Frame Entry`

The entry for beaming (or zapping) has this form:

```
zap: {
      title: "Beam",
      routeForm: 'zapSlip},
```

The `routeForm` slot contains a symbol for the slip that will be shown to the user when they pick that type of routing from the action button. This slip is used to get the information used to route the target. `zapSlip` is the standard system beaming slip. There is should be no need to create a custom slip.

Note that beaming will use the target slot of the application to get the data it will send. If there is no target slot, it will look for a setupTarget function that will be called (see section 1). If for some reason (like multipart beaming) you are setting up your own outbox entries, you must set the body slot of fields to the data that will be sent (do this in the setupRoutingSlip method).

`PutAway(item)`

When someone beams or emails an object to you and you choose "auto put away" (for beaming only) or "put away" (for email and beaming), this message is sent to your application to tell it to store the new item in the receiving Newton soup. PutAway is also used by mail (and could be used by other services).

The message has one argument, that is a frame. The only slot you need is the one called `body`. This is the frame that you will add to your soup. An example script is:

```
func(item)
begin
// NOTE: may be called when the app is closed

   local newEntry;
   // look at body slot to get the new frame
   newEntry := item.body;

   // make sure that filing folder exists
   // on this Newton, or nil it out if not
   CheckThatFolderExists(newEntry);

   // get the soup to store it in
   local soup := GetUnionSoup("WiggySoup");

   if soup then
   begin
      // add the frame to the soup
      soup:AddToDefaultStore(newEntry);
      // inform apps that the soup has changed
      BroadcastSoupChange("WiggySoup");
   end;
end;
```

Warning: PutAway can get called when your application is closed. This means you can not rely on anything that you set up in places like the viewSetupFormScript existing. This is true for messages that are sent from your PutAway script.

A good example is the BroadCastSoupChange call made above. It will cause the SoupChanged script in your application to be called, and your application may not be open. So if your SoupChanged script attempts to update your display based on the new data, you will crash. Make sure you check if your application is open with something like:

```
if Visible(GetRoot().(appSymbol)) then ...
```

4. ATR (All the Rest - Printing, Faxing, Mailing)

All the other types of routing require two things: a routing frame entry and one or more formats. The formats are what take time.

Routing Frame Entry

The routing frame entries for printing, faxing and mailing have a slightly different format from the others. The general form is:

```
routeType: {
   title: <some title>
   routeForm: <a slip type>
   formats: <format array>}
```

routeType: can be anything, but is usually print, fax or mail

title: a title that makes sense like Print Wiggy or Fax

routeForm: a symbol for a slip used to get information for the route. This is either `printSlip, faxSlip,or mailSlip`

formats: an array of formats for the routed thing (see formats below)

An example routing frame entry for printing would be:

```
print: {
   title: "Print Wiggy",
   routeForm: printSlip,
   formats: [wigBusinessFormat, wigMemoFormat]}
```

Note: The user interface convention is that the first entry in the routing frame (for example, the first entry in the action button), has the action name (Print, for instance) and the object you are printing (Wiggy). All other titles in the routing frame have just the action, but not the object (Carding seems to be a minor exception, but is not really, since you have to say where you are going to Move or Copy the item to /from).

Formats

In the routing frame entry is a formats slot that contains an array of symbols. Each of these symbols refers to a format that the user can select. A format is a way of formatting the page for printing, faxing or mailing.

Each of the formats must be attached to the root view. The slot name of the format must be the same as the symbol in the formats array in your routing frame. Using the print entry above, you would need to add two slots to the root view: one would be wigBusinessFormat; the other wigMemoFormat. Each slot contains a routing format frame.

You can use the GetRoot function to add these slots. In general, you will attach your formats in the installScript of your application, and remove them in the removeScript. An example of this is:

```
const kWigBusinessFormatSymbol := '|wigBusFormat:WIGGY|;
const kWigMemoFormatSymbol := '|wigSymFormat:WIGGY| ;

installScript := func(partFrame)
```

```
begin
   ...
   // attach my formats to the root
   // wigBusinessFormat is a user template
   GetRoot().(kWigBusinessFormatSymbol) :=
      partFrame.theForm.wigBusinessFormat;
   // wigMemorFormat is a user template
   GetRoot().(kWigMemoFormatSymbol):=
      partFrame.theForm.wigMemoFormat;
end

removeScript := func(partFrame)
begin
   ...
   // remove my formats from the root
   RemoveSlot(GetRoot(), kWigBusinessFormatSymbol );
   RemoveSlot(GetRoot(), kWigMemoFormatSymbol );
end
```

5. Routing Formats

Routing formats lay out data from the Newton in a way appropriate for the routing medium. This gives you a chance to take information written on a tall thin screen and reformat it for the printed or faxed page, or perhaps for electronic mail as well.

Format Frame

The format frame is a container frame that specifies some general information about the format. It also contains a layout/template that is used to generate the pages for printing and faxing.
The format frame looks like this:

```
{
   title: "Business",
   auxForm: nil,
   textScript: 'wiggyTextScript,
   attachment: true,
   _proto: ROM_coverPageFormat,
   mainFormat: printFormat_wiggyBusinessFormatLayout
}
```

title: title that shows up in the Format popup in any routing slips
auxForm: defines a layout for a slip to gather extra routing information.
See the auxForm sample code for an example of how to use this slot

textScript: symbol of a script in your application that returns a text only representation of the information to route

attachment: true if frame data can be attached to electronic mail

_proto: ROM_coverPageFormat

mainFormat: template that generates the pages to print/fax

To find where the title is used, go to the notepad, click on the action button and pick print. You will get the print slip with two pickers. The second picker is the Format picker. The entries in this picker are generated from the title slot.

It is highly unlikely that you will ever need to define an auxForm for routing. If you want an example of an auxForm in action, pick the Memo format. You get an auxiliary slip that gets more information. This is an auxForm.

You should use the ROM_coverPageFormat. It represents the printing behavior and does not necessarily require an actual cover page, despite the name of the constant. ROM_coverPageFormat will automatically generates the standard Fax coverpage if the user requests one, and ensures that the "fax header" is printed at the top of all your pages.

The mainFormat is a layout and is explained below later in this document.

`textScript(fields, target)`

This function must return a text representation of the information. It will be used as the body of an electronic mail message.

fields: the special "fields frame" (the actual outbox/inbox soup entry)

target: the target entry of the route (as set by your application).

This script will just concatenate a string of the right things. Note that you can not send graphics or other Newton data using this script. If the attachment slot of the format frame is true, the Newton will take care of attaching the target frame to the e-mail message.

An example text script...

```
func(fields, target)
begin
   local text := "";
   text := "Check #" & NumberStr(target.ckNumber);
   text := text & "\n" "To:" && target.ckPayee;
   text := text & "\n" & "For:";
   text := text & :localizedMoney(target.ckAmount);
end
```

The example script returns a string something like this...

Check #4To: Ned the Wonder LlamaFor: $1,000,000.00

mainFormat

The mainFormat is a view template that specifies how to format your information. There are three important things about this format:

- The top level view should be a protoPrintFormat
- The top level will be the correct size for the output paper
- The top level *must* define a printNextPageScript

Use the protoPrintFormat, it saves a lot of trouble!

The protoPrintFormat will automatically adjust itself to the size of the output paper. That means you can use the viewJustifySlot in the children you add to the slot to make sure they use the full page. You can also get the size (in pixels) of the output page by sending a LocalBox message to the view that protos from the protoPrintFormat.

`printNextPageScript()`

This function determines if there is another page to print. If so, it sets its view children for the new page (perhaps using RedoChildren), and returns true. If there is no next page, it returns nil.

The key part of this script is setting up the next page. In order to do this you need access to the target of the route. However, the format is attached to the root view, that is, it does not have the context of your application. Instead, it has access to the `fields` frame. This frame is set up by the routing system and contains a slot called `target`, which is the target entry that your application set up. In other words, use `fields.target` to get at the data being routed.

Note: You can use the New Print Format menu item in the File menu in NTK to do a layout for the mainFormat. You can access this layout as printFormat_<file-name>. You will have to manually specify the formatFrame, and then give the mainFormat slot a value of printFormat_<file-name>.

IMPORTANT: Your **`setupRoutingSlip`** message in your base view must set fields.target to a valid target, otherwise this message will never get sent!

Sending Frames, What Application is Opened at the Other End? (11/11/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-13)

Q: I made an application with routing, and tried to transfer the soup with IR beam between two Newton Message Pads. The Newton which sent the soup has my application on it, and the other Newton, the one which receives the soup, doesn't have my application.

I can transfer the soup, but when I tapped the item in the inbox, the Inspector displayed an error message "Undefined Variable (xxxxsym)" ("xxxxsym" is the value of the appSymbol slot and defined as a quoted symbol.) Of course, I can delete this item with the Inspector, but I cannot delete the item in the end user position. Could you tell me how to do it?

A: The answer is that the information sent to the other Newton contains the application symbol of the application that sent it. When you tap Put Away in the In Box, the system looks at the application symbol and tries to run the PutAway script of the corresponding application

There are two possible solutions to this problem:

1. On the receiving Newton, have a copy of the application including a PutAway script that can deal with the information beamed over.

2. If you are sending over information from the built-in applications (like cards from the cardFile), you can change the application symbol of the transmitted data and redirect it to the correct system application.

You can change the application symbol in your setupRoutingSlip message. As explained in the other Q&As on Routing, this message is called just before the routed item (in this case, the beam) is placed in the Out Box. This message takes one argument called fields.

Fields is a frame that contains the both the data to be sent to the other Newton (in the target slot) and information about where the data should go. The fields frame contains an appSymbol slot that is set to your application symbol by default. You can change this in your setupRoutingSlip message.

Why This Works:
When you route something, the system creates the fields frame to contain your data and extra information. This fields frame is totally cloned and sent to the other Newton. One slot in the fields frame is called appSymbol, this slot contains the application symbol of the application on the receiving Newton that can operate on the received information.

When the user taps the Put Away button in the In Box, the system uses the appSymbol to call the appropriate PutAway message, the call is something like this:

```
GetRoot().(fields.appSymbol):PutAway(fields);
```

If you change the appSymbol slot to another application, that application will be called.

Landscape Printing, Currently Not Supported (12/18/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-7)

Q: How can I print from the Newton in Landscape orientation?

A: Unfortunately, printing in landscape mode is currently not supported by Newton. It will be supported in future Newton family products.

Item Titles, Beaming (12/19/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-7)

Q: I don't know how to set my beam item title so it shows up in the Inbox or Outbox. How do I set the displayed title?

A: Use the fields.title field in your setupRoutingSlip method, as in:

```
myApp.setupRoutingSlip(fields): func
begin
     fields.title := "My Stuff" && ShortDateStr(Time(), 0);
end;
```

Consult The Little Prints sample code for more details on how to use the fields slots.

Printing and the Application Not Present (12/19/93)

Q: When I remove the card with my application on it, I cannot print. What is wrong?

A: Printing of outbox entries will work only if the application is present in the system. If the application is gone (card is not inserted, package removed), print entries queued using the specific application are not printed. Typical error codes when this happens are related to unknown variables, methods or something else missing.

Changing Print Formats on the Fly (12/19/93) (Obsoleted by Newton Programmers Guide 1.0 Documentation 13-21)

Q: In my application I have two main print formats, but only one is really valid at any time. Can I change the default print format during the application run time without getting into trouble?

A: Every time the user opens the printSlip (or faxSlip) the system will check a slot in your application to see what format is the default one. You set the `lastFormat` slot to be the symbol that corresponds to the format you want to be the default choice.

Note: If the user chooses a different slip, the `lastFormat` slot will be set accordingly. The `lastFormat` slot exists in your application base view; in other words, the one attached to the root View.

Note also that changing the default format in your `setupRoutingSlip` method will not have the desired effect, because the print (or fax) slip is already shown at this point of time. You have to set this up before `setupRoutingSlip` time, perhaps in reaction to some event in your application.

Also note that Mail uses a slot called `lastMailFormat`, not `lastFormat`.

How do I get the triangle in the filing button for card items (12/20/93)

Q: How do I get the triangle in my filing button? Do I have to draw it in the view myself?

A: To get the triangle you need to do 4 things:

1. Support soup notification by adding yourself to the soupNotify array.
2. Declare your protoFilingButton to your base view.
3. Keep your base view target slot up to date (that is, pointing to the current cursor entry).
4. In your soupChanged method, add this code:

```
// protoFilingButton declared as filingButton
if filingButton then
      filingButton:Update();
```

The update message will tell the button to check the current target of your baseview and update the triangle based on that entry.

How do I get a default name in the faxSlip (or elsewhere) (12/22/93)

Q: How do I set up a name in the faxSlip automatically?

A: To get a default name in the faxSlip you need to tell the slip what cardfile entry or entries to use. You specify an array of cardfile entries to be used for the name pick list in the faxSlip. The first element in this array will be chosen by default. The phone number will be set to the first phone of type Fax from the entry in the cardfile.

However, you do not always want to add the information for the faxSlip since that would add confusion to a beam (for example). So here is a way to do things:

```
app.setupRoutingSlip(fields)
begin

// may want to limit the number of items shown
// in the list, i.e., MapCursor is not necessarily
// the best way to get the hits...
local hits := MapCursor(
      Query(GetUnionSoup(ROM_cardfilesoupname),
            {type: 'words, words: ["Apple"]}),
      func(entry) entry) ;

if fields.category = 'faxSlip then
      fields.alternatives := hits;

      // do other stuff like setting the outbox title
end;
```

Note that `MapCursor` returns an array of cardfile entries that contain the string specified in the argument. To get a list of people in California, set theName to the string "CA". If you know the name of the individual, you could use that too (for example, "Reg Llama").

You can use this technique in other places that require a name (the mail slip, the addresseeSlip auxiliary information slip, and so on). Just make sure that you do not add an alternatives slot unless needed, since in a beam (or mail enclosure) the alternatives slot will be duplicated and sent.

Note that you could also use ParseUtter to parse a string such as "fax joe" or "fax bill at 555-1212".

Printing Resolution 72DPI/300DPI (2/8/94)

Q: I've tried to print PICT resources; the picture was designed in Illustrator and copied to the clipboard as a PICT. The picture printed correctly but at a very low resolution. Is there any way of printing PICTS with a higher resolution?

A: Currently the only supported screen resolution for PICT printing is indeed 72dpi. This may change in future platforms, so stay tuned for more information.

Not all Drawing Modes Work with a Postscript Printer (3/8/94)

Q: It seems that not all drawing modes work with printing. Is that true?

A: Yes. Not all drawing modes will work when printing to a Postscript printer. The reason is that Postscript behaves like layers of paint; you can not go back and change something. Anything that uses an invert mode (like XOR, and possibly ModeNot* (to be tested)), will not work.

Note: If you want to get the effect of white text on a black/filled background, use bit clear more for drawing the text.

Printing Does Not Have Access to the App Frame or Globals (6/21/93)

Q: Why can't I find my application slots from my print format?

A: One important thing to remember is that the print format does not have direct access to your application context because it is a child of the root view, and not in the parent inheritance chain.

It does have access to the fields variable. The fields frame is TotalCloned, so do not put a view reference or a reference to ROM in the fields frame or else you will waste much space and possibly run out of memory while adding the entry the outbox soup! Try to put *all* of the necessary information to print the item within the 'body slot of the fields frame.

If for some reason, you need to access slots from your application (which may not be open during printing!), you can access them using getroot().(yourAppSymbol).theSlot.

Title in the In/Outbox Does Not Wrap (6/3/94)

Q: The message title in the outbox seems to overwrite other lines if it is too long. What is the maximum number of characters?

A: The title that you setup for the inbox or outbox in your SetupRoutingSlip method will not be wrapped by the in/outbox if it is too long. You should currently limit this title to 44 characters. If you ommit characters, place an ellipses at the end of the string (use the unicode ellipses character for this).

Out of Memory When Adding Items to the Outbox (6/7/94)

Q: My print format works fine in Print Preview, but when I try to print to a real printer, I always get an out-of-memory error adding it to the Outbox. Why?

A: In your SetupRoutingSlip method, you are saving something into fields which references something very large, or points into the ROM. Views, soups, and cursors are examples of things that can't be saved into fields. Instead, save the view's symbol, the soup's name, or the cursor's querySpec and do a lookup on the item in your print format. Do not save anything which might indirectly reference something large and unnecessary for the routing item, like the root view itself. Alternatively, your SetupRoutingSlip can retrieve the data from views, soups, and cursors and save the data in the body slot of the fields frame.

PICT Printing Limitations (6/9/94)

Q: My large pictures cannot print on my LaserWriter. Is there a maximum size Newton picture?

A: The current PostScript printing system in the Newton ROMs is unable to print extremely large individual bitmap frames, the kind of pictures created using the NTK Picture editor or the GetPictAsBits routine. This is because in order to print these, the Newton must copy the bitmaps into an internal buffer. Thus the GetPictAsBits case fails (current limitation is a 168K buffer, but do not rely on a specific number for other Newton devices).

Using the GetNamedResource(..., 'picture) routine, you can use PICT resources to be drawn in clPictureViews. Macintosh PICT resources often contain multiple opcodes (instructions). For single-opcode PICTs, compression is done for the whole picture. You can check *Inside Macintosh* documentation for specifications of the PICT format. If you are using very large bitmaps which you will print, you should use PICT resources composed of many smaller 'bitmap copy' opcodes because they will print much faster and more reliably on PostScript printers. This is because very large PICT opcodes printed to LaserWriters must be decompressed on the printer. The printer's decompression buffer is sometimes too small if the opcodes represent large bitmaps. Check your Macintosh graphics programs for more information on segmenting your large PICTs into smaller pieces. For some applications, you might have two versions of the PICTs, one for displaying (using GetPictAsBits for faster screen drawing), and a large tiled PICT for printing.

Note that the PICT2 (color) picture format is not currently supported by the Newton drawing system.

Fax Timeout & Banding (6/21/94)

Q: Printing seems to work fine for my application but faxing does not seem to work. It looks like it hangs during connection while my application is drawing a lot of shapes. Are some drawing commands not allowed during faxing?

A: Faxing is time-sensitive so you might be "timing out" the other faxing machine. Since the imaging system needs to know the resolution of the other fax device before imaging the page, the connection has to be established before your printFormat is opened.

You can run into problems here because the fax at the other end of the line will disconnect if there is no activity for a certain period of time (around 5 seconds). This means that your printFormat must initialize itself quickly. Do things like caching your data in your setupRoutingSlip method instead of making lots of soup queries in your printFormat.

Splitting up your printFormat into multiple views (see the other Q&As about Banding) will also save time during faxing.

How to Improve Print & Fax Speed when Banding (6/21/94)

Q: Is there a speed difference between printing one big, complicated view and printing many smaller simple views?

A: Printing and faxing use a banding method to render the page to the output device. This means that your viewDrawScript for the mainFormat can be called many times for a single page. If you need to do initialization, do it in the viewSetupFormScript of the top level print format, but this will be called after the Fax connection is established so this can also cause a timeout if taken too long.

If your viewDrawScript changes a cursor to point to other entries, it should Clone (using cursor:Clone) before changing it. If it uses variables to keep track of things like width, height, or other counters, it should reset those variables on each call.

Only the viewDrawScript will be called multiple times. View scripts like viewSetupFormScript and viewSetupDoneScript are called only once. Of course, your printNextPageScript could call RedoChildren which would call the viewSetupFormScript again.

If you use multiple views to implement your print format, the system will only render the views in the current band. In other words, do not draw your entire page; break up your page into multiple views if you want to speed up drawing.

You do not necessarily have access to the coordinates of the current band, so if you are drawing your entire page, things will be very slow. This can be nasty for faxing, since long delays between bands can cause a timeout. See the "How to Determine the Drawing/Banding Region" Q&A for more information about this.

How to Determine the Drawing/Banding Region (6/21/94)

Q: Is there any way to determine which printing "band" is printed? If I cannot tell, my application must do a lot of drawing and long delays between bands can cause a timeout with the other fax machine.

A: You do not necessarily have access to the coordinates of the current band, so if you are drawing your entire page, things will be very slow. If you use multiple views to implement your print format, the system will only render the views in the current band. In other words, do not draw your entire page; break up your page into multiple views.

There is a routine to determine the current "band", but it is available only in ROM versions 1.3 and greater, so you cannot rely on it existing on all Newton ROM versions. You can call the *view*:GetDrawBox method to get a rectangle in global coordinates representing the current drawing area. This function is defined in the root view so you will use the method through inheritance. Note that you should not rely on a standard "height" for bands; it will differ among different printer and fax drivers.

Do Not Use the SetupTarget Mechanism (6/29/94)

Q: I cannot get my SetupTarget method to work. When is it supposed to be called?

A: Although the current documentation specifies that your base view's SetupTarget method can be implemented to do custom setup of a target, this is incorrect. The setupTarget mechanism is not fully supported in the current Newton ROMs and we recommend that you avoid currently implementing

SetupTarget. SetupTarget is not consistently called and the only way to force its use with a protoAction button (that is, have NO target slot exist) will throw exceptions from the current protoActionButton.

TargetView Does Not Get Routing Messages (6/29/94)

Q: My targetView does not get my deleteActionScript or deleteScript and I am sure I set targetView correctly. What could be wrong?

A: The current documentation incorrectly specifies the use of the targetView slot. The targetView is used by several parts of the Newton system, including routing and filing. When your application registers your routing frame, your base view will actually get routing messages, which in some cases include the target and the targetView as parameters. For instance, if you want specific views to be sent messages for delete, your base view's deleteActionScript could send a message to the view represented by targetView, which will be the second argument to deleteActionScript.

Checking for Fax Sent (7/6/94)

Q: How can we make sure a fax has been sent? We would like to know when this has happened so we could do persistant changes to a soup or to some other entity after the fax has completed.

A: Usually when an item has been sent, it will be removed from the outbox soup. You should not modify an entry as long as it is referred in the outbox. If the entry is no longer in the outbox soup, then it is has been sent.

Printing Special System Fonts with a PostScript Printer (7/26/94)

Q: When printing from my application on the Newton to a PostScript Laser printer, I noticethat the fonts are being substituted. One problem is that the diamonds indicating a popup list don't show up. Also, the text is unpredictable since the substituted font is bigger, and thus certain words are cut off. Is there a way to turn off font substitution?   Is there a PostScript version for the LaserWriter that can be downloaded? Printing always looks fine on a QuickDraw printer like the StyleWriter.

A: Yes, this is true.The additional System font (Espy Sans) is not printed on a LaserWriter, because the fonts are missing in the PostScript font specifications. Actually, just printing Espy Sans (Newton system fonts) is currently not possible on the LaserWriter, but is possible on faxes and bitmap printer drivers, since the rendering for those is done inside the Newton.

The troublesome characters are the Apple-specific ones, starting with Hex FC. The filled diamond is one of these characters, the specific tick box arrow is another.

You might try another PostScript font and use a character that resembles the filled diamond. If you want to create a label button, check the Button Mania sample.

NewtonMail APIs (9/14/94)

Q: What APIs are available for NewtonMail?

A: There are several different types of APIs:
1) Standard send/receive mail routing API. You can send frames between Newtons using the standard routing interface. You put the data in the item's 'body slot, in the same way you set up data for beaming. If your application has a PutAway script, it will allow the application to receive data. Check the CheckBook sample for more info.
2) The Send API. You can programatically send electronic mail from NewtonScript if you use the NTK Platforms File version 1.1. For more information, see the AutoRoute sample code project and the NTK Platforms File documentation for Send API details.
3) Install package via NewtonMail. When you send a file from the Macintosh eWorld client to a NewtonMail account, if the package type is 'pkg ', then the recipient will be prompted to install the package. Warning for developers: package size is limited to about 15K, and the recipient must have appropriate free space in their inbox soup (the internal store). Note that if users have the NewtonMail Extension 1.1 installed (or a ROM version greater than 1.3) , it will increase reliability. This is because of patches that address problems in the original inbox/outbox design and help with receiving mail in low memory conditions.

Updated: Checking the ROM Language Environment for :MungePhone (2/16/95)

Q: I did a small telephone interface in my application. On my US Newton everything runs fine. But on a German Newton there are two situations when a error occurs
(-48409):
1. when I call :MungePhone(theNumber, "Germany") and;
2. after opening the callOptionsSlip, when the user closes the Optionsslip.
Maybe on localized Newtons the ROM_countrylist has different values? Can I use the GetGlobals().userConfiguration.country slot?

A: If you print out the values of the the ROM_Countries global on a German system, the name slot of Germany is " Deutchland", as in:

```
Germany: {name: "Deutschland",
          outgoing: "00",
          countryCode: 49,
          inPrefix: "0",
          postalCodeNumeric: TRUE},
```

The same test with ROM_Countries on a US 1.0 ROM system returned:

```
Germany: {name: "Germany",
          outgoing: "00",
          countryCode: 49,
          postalCodeNumeric: TRUE},
```

The names are indeed translated to the language environment itself; the name slot is Finland in the US system, and Finnland in the German one .

This means that you can't make any conclusions about the country ROM environment , either using the ROM_Countries global, or using the country slot of userConfiguration.

NTK 1.0.1 has a new function that will return the country code of the particular environment. It's called kGetLanguageEnvironmentFunc, and an example of using this is:

```
call kGetLanguageEnvironmentFunc with ()
```

This function will return the Apple-specific country code: US is 00, Germany is 03, France 01.

End of DTSQA

SOUND

TABLE OF CONTENTS:

SOUND 1
Introduction..1
ROM Dial Tone Sounds (3/8/94)..1
Sampling Rates (3/8/94) 2
PlaySoundSync and PlaySound Synchronization Problems (11/11/93) 2
Looping of Sounds (3/8/94)..2
Overriding Button Sounds (3/8/94)..3
Dynamic Sounds (3/8/94)..3
Stopping a Synchronous Sound (3/29/94)..4
Dialing Synchronously (8/22/94) 4

Introduction

This document addresses Newton Sound 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.

ROM Dial Tone Sounds (3/8/94)

Q: Are the dial tones stored in ROM in a place I can use?

A: There are a series of phone dialing tones stored in ROM. These are stored in ROM_dialtones. This array contains 12 tones, all of which are also sound frames, so you can play them using:0

```
    PlaySound(ROM_Dialtones[5]);
```

(See also the Sound Q&A "Dialing Synchronously")

Sampling Rates (3/8/94)

Q: What sound sampling rates are allowed?

A: There are two standard rates, and these are defined in the NTK definitions file:

```
kFloat11kRate := 11013.21586; //In kilohertz
kFloat22kRate := 22026.43172; //In kilohertz
```

Looking at the built-in sounds from the Inspector shows them:

```
rom_funbeep
#19B  {sndFrameType: simpleSound,
      samples: <samples, length 5236>,
      samplingRate: 11013.2,
      dataType: 1,
      compressionType: 0}

rom_hilitesound
#1BB  {sndFrameType: simpleSound,
      samples: <samples, length 3244>,
      samplingRate: 22026.4,
      dataType: 1,
      compressionType: 0}
```

If you manipulate the samplingRate slot of a sound frame, you can set the samplingRate to any float value less than the kFloat22kRate. However, this usually results in poor sound quality. What usually works best is to take an 11kHz sound and play it at some higher rate. Of course, 22kHz sound resources cannot be played at any higher sampling rate.

PlaySoundSync and PlaySound Synchronization Problems (11/11/93)

Q: Why does PlaySoundSync(ROM_click) produce a sound and PlaySound(ROM_click) not?

A: If you are calling PlaySound just after a PlaySoundSync, there is a known problem with the sound channel not getting reset properly for a short time. We recommend that you one of those two routines, but not both in succession. For a complex or quick music program, PlaySoundSync would minimize unexpected effects. For instance, if you call PlaySound twenty times in a for loop, you are likely to hear the sound only 4 or 5 times, and sometimes it will be audibly interrupted.

Looping of Sounds (3/8/94)

Q: Can the Newton support continuous repeated playback of a single sound?

A: Currently Newton does not support this. You could implement something approximating this from scripting using a viewIdleScript and a fairly long sound. So, for instance, in your viewIdleScript you might call PlaySound with a sound that was 2 seconds long.

We are aware that this functionality would be very useful, especially for games, and we are looking into adding it to future Newton products.

Overriding Button Sounds (3/8/94)

Q: How can I override the click sound my button makes?

A: One way to do it is to call PlaySound in your buttonPressedScript (not the buttonClickScript) with the sound you want to hear. The buttonClickScript plays the click sound before your buttonClickScript is called, so calling PlaySound there will result in the user hearing both the click sound and your sound.

Another way to do this is to override your viewClickScript, and call PlaySound from there. The problem with that method is that you will need to take care of all the button tracking to ensure that the button highlighting is done properly.

Dynamic Sounds (3/8/94)

Q: How can I create my own sound frames during run time on the Newton?

A: Creating your own sounds during run time on the Newton is currently slow but not terribly difficult. The SoundTricks sample code shows how to do this.

First of all, look at the built-in ROM sounds to see what a sound frame looks like:

```
rom_funbeep
#19B  {sndFrameType: simpleSound,
       samples: <samples, length 5236>,
       samplingRate: 11013.2, /
       dataType: 1,
       compressionType: 0}
```

Currently all sounds use the same sndFrameType, dataType, and compressionType. These slots are discussed in the NPG chapter "Working With Sound". The dataType of k8bit (which evalutes to 1) means that the sound data is stored in the samples slot as a series of one-byte samples. Future Newton products will probably support other dataTypes and sndFrameTypes.

When you create your own sounds, note these important things:
1. Your samples slot must currently have class 'samples. You can obtain this by using SetClass.
2. Your samples slot must have a length which is a multiple of 4. If it doesn't, you might observe memory heap problems.
3. Use the sound-related constants rather than actual numbers for things like dataType.

For example, here's how the GetSound function sets up a sound frame:

```
{
       sndFrameType:     'simpleSound,
       samples:          soundSample,
       samplingRate:     kFloat22kRate,
       dataType:         k8Bit,
       compressionType:  kNone
};
```

Stopping a Synchronous Sound (3/29/94)

Q: How do I stop a synchronous sound from playing?

A: Currently in the 1.0 ROM there's no way to stop a sound that is played synchronously. This will change in future system releases.

One workaround is to play the long sound, and when you want the sound to suddenly stop, play another very short and inaudible sound. The second PlaySound will terminate the first sound.

Dialing Synchronously (8/22/94)

Q: The view method `:Dial(number, where)` returns immediately, and the dialing begins a moment later. I want to post some status messages while various parts of the number are dialed. How can I do this?

A: There is a global function called `RawDial(number, where)` that you can use to accomplish this. In the current system the view method `Dial` adds a deferred action that calls `RawDial`, but you can call `RawDial` directly. The parameters are the same as for the view method: the first is a string representing the number to dial; and the second is a symbol — either `'speaker` or `'modem` — that specifies how dialing is done. The function will return immediately after dialing the last digit.

Normally you would use the view method `Dial` or the global function `RawDial` in preference to generating the dial tones and `PlaySound` or `PlaySoundSync` using the sounds in `ROM_dialtones`.

End of DTSQA

SYSTEM DATA

TABLE OF CONTENTS:

SYSTEM DATA....1
Introduction....1
Drawing Persistent Objects into PaperRoll (11/11/93)....1
Finding the Notes Information (11/11/93)....1
Finding CardFile Information (11/11/93)....3
Finding Calendar Information (11/11/93)....3
Doing a Pick List with Names (6/15/94)....3
Creating exceptions for repeating meetings (6/9/94)....4
Getting/Manipulating UserConfiguration Values (7/7/94)....4

Introduction

This document addresses Newton System Data 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.

Drawing Persistent Objects into PaperRoll (11/11/93)

Q: I tried adding a drawfunction to GetRoot().paperroll. I was able to draw a little rectangle on the notepad. Unfortunately it was mangled after a scroll. I guess I need a function call to my code somewhere around the time the view draws those dashed lines. Is this possible?

A: Unfortunately, modifying the built-in apps is not supported by Apple and it is probably more trouble than it is worth. The reason you are having trouble removing the dashed lines is that they are not drawn by the "paperroll" but are part of the behavior of standard clEditViews which are created dynamically by the Notes application. In order to modify those individual views, you would have to navigate through children of the paperroll and intercept any conceivable event which could cause the Notes application to create, draw, or modify children views.

Unfortunately, the kind of single desktop deskHook with which you might be familiar on the Macintosh is not present on the Newton API because the Newton metaphor includes a paper notepad in the backmost layer rather than an abstract desktop. Replacing the entire Newton notepad application with your own is not a simple project either and we recommend against it.

Finding the Notes Information (11/11/93)

The Notes application has a slot called datacursor. When this is non-NIL (you should check this), it is a cursor that points to the Note the user is currently viewing. You can access the note with this call:

```
currentNote :=
      GetRoot().paperroll.datacursor:Entry();
```

This returns the frame entry for the current note. The frame has a slot called data that is an array of all the data in the current note. Each entry in this array is either a clParagraphView or a clPolygonView. You probably need only the text, so you will want entries that are clParagraphViews. It turns out that each entry also has a viewStationery slot that has a value of 'para for text entries.

This function will return a list of all the text views in the current note. If there is no current note or the current note has no text items, it will return an empty array:

```
currentNoteText: func()
begin
      local currentNoteCursor :=
      GetView('viewFrontMost).datacursor ; local currentNote;
      local textEntryArray := [];

      if currentNoteCursor and
      (currentNote := currentNoteCursor:Entry()) then
      foreach item in currentNote.data
      do
            if item.viewStationery = 'para then
      AddArraySlot(textEntryArray,item);
      textEntryArray;
end
```

Once you have a list of the text views in the current note, you can find out what the text is by looking at the text slot. You can also change this text slot to some other value. If you do this, you need to make sure you update the soup. You also need to tell the Newton to redraw the changed children.

The following function will number each of the text items in the current note; it will also update the Notes soup and the display:

```
numberTextNotes: func()
begin
      local currentNoteCursor :=
      GetView('viewFrontMost).datacursor ;
      local currentNote ;
      local num := 1 ; // check that there is a current note

      entry if currentNoteCursor and (currentNote :=
      currentNoteCursor:Entry()) then
      begin
            foreach textNote in :currentNoteText() do
            begin
                  textNote.text := num & "." && textNote.text;
                  num := num + 1 ;
            end ; // update the Notes Soup
            EntryChange(currentNote) ;
            // update the Newton views
            GetView(currentNote.data[0])._parent:RedoChildren();
      end;
end
```

NOTE: If this function is called repeatedly, it will continue to append numbers to the text items in the note. That is, it does not check to see if a text item has already been numbered. For your application, this means you need to check to see if some piece of text has already been
converted.

Finding CardFile Information (11/11/93)

The cardfile also contains a datacursor slot. However, the data format for the cardfile is a bit different. This format is documented in the Accessing System Data chapter of the Newton Programmer's Guide. This chapter containes the soup data formats for the Notes, CardFile and Calendar soups.

Finding Calendar Information (11/11/93)

To access information in the calendar is difficult. The easiest solution is to determine which dates are displayed, and use these as the basis of a query into the correct soup. This requires knowing which soup to check (Calendar, Calendar Notes, ToDo).

The Calendar and Calendar Notes soups would be checked if the calendar is active. The ToDo soup is checked when the todo screen is active.

To find which dates are displayed, check the selectedDates slot. This is an array of the selected dates (or date if just one is selected). This bit of code gets the value of the slot:

```
local theDates := GetView('viewFrontMost).selectedDates;
```

You also need to find out if the calendar is in Month view or ToDo view to figure out which soup to check. The following function returns true if the calendar is in Month view, nil otherwise (including if the calendar is NOT the current application):

```
IsMonthView: func()
      visible(GetView('viewFrontMost).monthView) ;
```

Doing a Pick List with Names (6/15/94)

There are a few useful functions defined in the cardfile for doing pick lists and other things:

If you want to implement a pick list of names you could use a word query into the names soup to get the names, then addding some number hits into an array.

```
BCFullName(<entry-name-slot) - RootView message
```

<entry-name-slot> the name slot from a cardfile entry (i.e., the frame with slots first, ...)

This function returns a string that is the full name of the person specified by the name frame. Does not include any honorifics (Mrs., Dr., ...) or titles.

The two functions are extremely useful in constructing pick lists for names. If you had the string a user was entering in a protoLabelInputLine, you could use that string to set the popup portion to a list of possible names...

```
    // may want to limit the number of items shown
    // in the list, i.e., MapCursor is not necessarily
    // the best way to get the hits...
    local hits := MapCursor(
          Query(GetUnionSoup(ROM_cardfilesoupname),
                {type: 'words, words: ["Apple"]}),
          func(entry) entry) ;
    if hits then
          :SetLabelCommands(ListFullCFNames(hits)) ;
```

Note that you could do even more, like setting a check mark on the current name.

Creating exceptions for repeating meetings (6/9/94)

Q: On p. 8-9, the Newton Programmer's Guide says "The internal format of [exceptions] is subject to change, so you should treat this array as read-only, and not attempt to add to it." How can I create an exception meeting?

A: Modify the array. We will support "deleting" an instance of a repeating meeting by adding the exception time and NIL as two elements to the exceptions array in a repeating meeting entry. Once you have "deleted" the instance of the meeting, you can create a new entry in the regular meetings soup (ROM_CalendarSoupName) to represent the rescheduled or otherwise modified meeting.

Getting/Manipulating UserConfiguration Values (7/7/94)

Q: If I want to change preferences stored in the userConfiguration frame, how do the Newton applications know that there is a change?

A: You should not directly manipulate or retrieve information from the userConfiguration frame.

You can use the new NTK Platforms File functions `kGetUserConfigFunc, kSetUserConfig, and kFlushUserConfig` to manipulate the userConfiguration values. Use the `kFlushUserConfig` after you have modified various entries in the user confifguration frame with `kSetUserConfig`. Flushing the user configuration frame ensures that previous changes to the user configuration frame are saved to the system soup. For instance, to get the name of the current printer, you can use the following code:

```
// do NOT modify or read any slots in the userConfiguration frame
// or the printer frame...treat them as a "black box"
myPrintFrame := call kGetUserConfigFunc with ('currentPrinter);
thePrinterName := call kGetPrinterName with (myPrintFrame);

// if you want to change userConfiguration, don't do it directly!
call kSetUserConfig with ('name, "Elvis");
call kFlushUserConfig with ();
```

End of DTSQA

HANDLING SYSTEM MESSAGES

TABLE OF CONTENTS:

HANDLING SYSTEM MESSAGES...............1
Introduction...1
Clarification
ViewHideScript and Disposing of Views (9/15/93) (Obsoleted by NPG 1.0 5-7)...1
ViewSetup Scripts (9/15/93) (Obsoleted by NPG final Doc 2-47, 5-2 to 18) 1
viewWordScript Problems (9/15/93) (Obsoleted, Documented in the NGP 1.0 Final Documentation 5-12,18)..2
Application Restart (6/7/94)..3
TieViews and Untying Them (6/9/94)..4

Introduction

This document addresses Newton System Messages 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.

Clarification: ViewHideScript and Disposing of Views (9/15/93) (Obsoleted by NPG 1.0 5-7)

Q: Why isn’t my viewHideScript script called properly when my view is disposed?

A: ViewHideScript is actually never called when a view is disposed. If you want to trigger an action when the view is closed, design your application to use the viewQuitScript script instead.

ViewSetup Scripts (9/15/93) (Obsoleted by NPG final Doc 2-47, 5-2 to 18)

Q: What's the real story with all these scripts named viewSetup*XXX*Script?

A: Here's a simplified list of the steps in the creation of a view:

1. The parent view is created.
2. The script viewSetupFormScript executes.
3. The script viewSetupChildrenScript executes (using viewChildren array).
4. The children views are created with the same sequence of events.
5. The script viewSetupDoneScript executes.
6. The parent view and its children are displayed.

Here are some helpful things to know about these scripts:

viewSetupFormScript
This is your chance to change your bounds, flags, font, and so on, in case you're calculating them dynamically.

viewSetupChildrenScript
This is your chance to change your viewChildren array before your child views are created. Depending on how your view works, you might do things with the elements of the viewChildren array that affect how the children will be created. You might even be creating the viewChildren array from scratch.

Note that this method, unlike viewSetupFormScript, is also called by RedoChildren. RedoChildren informs the view that something has changed; viewSetupChildrenScript is your chance to respond to these changes.

viewSetupDoneScript
After your child views are instantiated you can do something with them in viewSetupDoneScript. Because you can use viewAddChildScript to do something to each child as it's added, you usually only need to use viewSetupDoneScript if what you're doing to the view depends on instantiations of the other children, such as creating direct references to other siblings.

viewWordScript Problems (9/15/93) (Obsoleted, Documented in the NGP 1.0 Final Documentation 5-12,18)

Q: It doesn't appear that viewWordScript gets called AT ALL in my code. Is this possible?

A: Like viewGestureScript, viewWordScript is only called for views that don't handle the adding of words.

If taps are on for a view, the viewClickScript will get called as soon as the pen goes down within that view. The viewClickScript gets called before the view system does any view processing. If the viewClickScript returns TRUE, that pen interaction is DONE and no other piece of code will ever see it. If it returns FALSE, the pen interaction falls through to underlying views, the view system, etc.

If the tap is not handled in a viewClickScript, and strokes are on, the viewStrokeScript will get called as soon as the pen is lifted. Processing proceeds like the viewClickScript with regard to TRUE/FALSE return values. Note that the system sometimes regards multiple pen down/pen up events as a single stroke (for writing letters, words, and so on.) The viewStrokeScript will only get called the FIRST time the pen goes up, later pen events that are part of the same stroke can not be handled by the viewStrokeScript.

If gesture recognition is turned on, the viewGestureScript may be called. However, unlike taps and strokes, the viewGestureScript is called AFTER the view system processes the gesture, and will not be called if a default action was taken. That is, clParagraphView handles the scrub gesture, so a viewGestureScript on a clParagraphView will never receive the scrub gesture.

If word recognition is turned on, the viewWordScript may be called. Like the viewGestureScript, the viewWordScript is called AFTER the view system has processed the word. More complex views that handle written input, like the clParagraphView, will not call the viewWordScript, because the word has been handled by the view system.

Q: We are using the viewGestureScript to get hold of gestures for a box. We can get the tap and line gestures but the scrub and hilite never make it (I don't know about the others). How do we get the scrub and hilite gestures (as well as the rest) to be routed through the viewGestureScript?

A: See the answer above, in addtion:

In your case, you are probably using a view that already handles the gestures for you. If you use a base clView you will be able to get all the gestures, but you won't have any of the benefits from other views.

Application Restart (6/7/94)

Q: What exactly happens when an application is re-opened after having been closed? Specifically, which scripts get called, what is the application supposed to do when this occurs—start over, continue exactly where it left off, etc…?

A: When an application closes, it behaves like any other declared view: its visual representation (viewCObject) is destroyed; however, its base view is not disposed because it is declared to the system root view. If you store large objects in the application base view, you should clear them in your application's viewQuitScript method (note you could either set them to nil or use RemoveSlot). If you want to save state information (so your application can pick up exactly where it left off, for example) you must store it in a soup (such as a preferences soup - see the Preefer Madness sample) and retrieve it in your application's viewSetupFormScript method.

To reopen the base application view, send it the Open message. When your application (re)opens, the normal viewSetupFormScript is executed.

Also note that the base view frame in the root view is rebuilt when the Newton restarts.

TieViews and Untying Them (6/9/94)

Q: What triggers the pass of a message to a tied v iew? If I want to "untie" two views that have been tied with TieViews, do I simply remove the appropriate slots from the viewTie array?

A: The tied view's method will be executed as a result of the same actions that cause the main view's viewChangedScript to be called. This can happen without calling SetValue, for example, when the user writes into a view that has recognition enabled, the viewChangedScript will get called.

There is no current API for untying tied views. It may be wise to first check for the existance of an UntieViews function, and call it if it exists, but if it does not, removing the pair of elements from the tied view's viewTie array is fine.

End of DTSQA

ADDITIONAL SYSTEM SERVICES

TABLE OF CONTENTS:

ADDITIONAL....1
Changing Icons/Text in the Extras Drawer....1
Introduction....3
Notify Problems (9/15/93) (Obsoleted by the 1.0 Newton Programmer's Guide documentation 12-53)....3
Find and the Title Slot (11/10/93)....4
Confirm Notification (11/10/93)....4
Removing Filing Button, How? (11/10/93)....5
EnsureInternal Notification Information (12/18/93)....5
Notify Race Condition, Low on Memory (12/18/93)....5
That Triangle in the Filing Button (01/4/93)....5
Checking a Folder Exists (01/19/94)....6
Finding Folders and Folder Warnings (01/19/94)....6
AddUndoAction and PCMCIA (avoiding -10401) (2/25/94)....6
Exception |evt.ex|, -8003? (4/20/94)....7
DefaultFolderChanged does not exist (5/25/94)....7
UpdateFilter method of protoFolderTab (7/13/94)....7

Changing Icons/Text in the Extras Drawer

(11/10/93)

Obsoleted by the NTK 1.0 Platforms File and Release Notes. Do NOT use this code if you have NTK 1.0.1 and/or the correct platforms file.

```
/*

SetExtrasInfo.f v1

Snippet for an application dynamically changing its icon in the Extras Drawer:
SetExtrasInfo(appSymbol,newInfo)

  return value - the application's current Extras Drawer info frame (or nil if
    the application isn't found). Useful for restoring state.
  appSymbol - appSymbol of the application with the Extras Drawer information you
    want to change
  newInfo - new Extras Drawer information to use. A nil or missing slot means
    leave that particular attribute unchanged.

This function changes the specified application's Extras Drawer information and
returns its current state. The Extras drawer information is specified as a frame.
Currently,the only slots supported are icon and text:

{
  icon: //BitMap or PICT that shows in the extras drawer
  text: //text diplayed underneath the icon
}

*/

//define this method in your base view

SetExtrasInfo: func(appSymbol,newInfo)
begin
  local xDrawer := GetRoot().extrasDrawer;
  if xDrawer:SetExtrasInfo exists then
    return xDrawer:SetExtrasInfo(appSymbol,newInfo);

  local newIcon := EnsureInternal(newInfo.icon);
  local newText := EnsureInternal(newInfo.text);

  //hack around bug in EnsureInternal
  if NOT newIcon then newIcon := newInfo.icon;
  if NOT newText then newText := newInfo.text;

  local iconSlotSym := EnsureInternal('icon);
  local textSlotSym := EnsureInternal('text);

  local TestFn := func(searchVal,targetElt) searchVal = targetElt.app;

  //find icon in extras global array
  local xArrayPos := ArrayPos(extras,appSymbol,0,TestFn);
  if NOT xArrayPos then return;

  //save original icon for return value
  local xFrame  := extras[xArrayPos];
  local oldIcon := xFrame.icon;
  local oldText := xFrame.text;

  //extras element may require cloning - into RAM
  if IsReadOnly(xFrame) then
    extras[xArrayPos] := xFrame := Clone(xFrame);

  //update icon in extras global array
  if newIcon then xFrame.(iconSlotSym) := newIcon;
  if newText then xFrame.(textSlotSym) := newText;
  IconBox(xFrame,xArrayPos);

  //update extras drawer item iff extras drawer is open
  if xDrawer.viewCObject then
    begin
      local xDrawerViews := xDrawer:ChildViewFrames();
      local xDrawerPos := ArrayPos(xDrawerViews,appSymbol,0,TestFn);
      if xDrawerPos then
        begin
          local xDrawerView := xDrawerViews[xDrawerPos];
          if newIcon then SetValue(xDrawerView,iconSlotSym,newIcon);
          if newText then SetValue(xDrawerView,textSlotSym,newText);
          SetValue(xDrawerView,EnsureInternal('viewBounds),xFrame.viewBounds);
        end;
    end;

  local result := Clone('{icon: nil, text: nil});
  result.icon := oldIcon;
  result.text := oldText;
  return result;
end;
```

Introduction

This document addresses Newton Miscellaneous System Service 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.

An entry that has changed since the last distribution of this document is distinguished by the prefix NEW at the beginning of the subject heading , or if the entry has changed, marked with CHANGED.

Notify Problems (9/15/93) (Obsoleted by the 1.0 Newton Programmer's Guide documentation 12-53)

Q: I can't ever seem to get Notify to work (the Inspector window claims it doesn't exist, among other things.) Am I missing something?

A: Notify is a message to a view, so you have to either call it in a view context or send the message to the root view from the Inspector.

The syntax is...

```
:Notify(notifyType, <title>, <message>)
```

From the Inspector you can type this:

```
GetRoot():Notify(3, "Llama", "Your Llama is whooping")
```

Find and the Title Slot (11/10/93)
(Obsolete, Documented in the 1.0 Newton Programmer's Guide, Additional System Services Chapter)

Note that in order for Find to work properly, the base view needs a slot called title, and the title should have a valid name. One example is if you don't use protoApplication as the base view, and you put together your own base views using clView with other views and prototypes.

Confirm Notification (11/10/93)
(Obsolete, Documented in the 1.0 Newton Programmer's Guide, Views

Chapter, Modal Dialogs 2-66)

Q: How to I implement the OK-Cancel notification I've seen in the delete software user interface?

A: The function you want to use is called Confirm. Just like Notify, it is a message you send to a view. Its parameters are a bit more complicated, however:

```
view:Confirm(title, message, context, callback fn)
```

title is any string
message is the string you want to appear in the dialog box
context is the context in which the callback fn is to be called

callback fn is the function called when the user has selected OK or cancel. The callback function is passed a parameter which is either TRUE or NIL (in the case of OK or cancel, respectively) after the modal dialog box has closed.

So, for instance, your callback function in the Inspector might look like:

```
MyCallbackFn := func(userChoice)
begin
      If userChoice PlaySound(rom_funbeep)
      else PlaySound(rom_poof);
end;
```

Then you can use Confirm like this:

```
Getroot():Confirm(EnsureInternal("Nonsense"),EnsureInternal("Play
happy sound?"),self,'MyCallbackFn);
```

Always do an EnsureInternal around any string information, because these error messages are stored in the notifications global. If an end user pulls the card where the original strings are stored, you will get the "Insert Card" message.

Removing Filing Button, How? (11/10/93)

Q: Is there a way to get rid of the filing button in the Find Overview? My application does not support filing, so having the button there is a little bit confusing.

A: Currently there is no way to eliminate the filing button. We have filed a request for change concerning this.

EnsureInternal Notification Information (12/18/93)

You need to wrap any information passed to Notify with an EnsureInternal statement. This is because the information is stored in the notifications global, and if the end user pulls the card the system will be unable to find information if notifications, the global, is used somewhere.

This is an example of the right way to create a notification:

```
GetRoot():Notify(kNotifyAlert, EnsureInternal("LlamaCalc"),
EnsureInternal("You run out of Llamas, sorry mate!"));
```

Notify Race Condition, Low on Memory (12/18/93)

Q: Has anyone seen a bug in which the Newton goes into a fit of random "beep" sounds? I call Notify and usually it works fine. Every so often though (maybe 1% of the time) the Notify view doesn't show up; instead my Newton plunges into a sporadic beep state. Resetting the Newton is the only option at this point.

A: What is happening is that occasionally when you call Notify there isn't enough memory for Notify to complete its task. The system gets caught in what is essentially an infinite loop-- it tries to throw up an error box declaring that it doesn't have enough memory, but of course this is essentially another call to Notify.

Right now there's no real workaround for this (you *could* try forcing GC() just before the call to Notify), but I hope this helps your understanding of the problem a bit.

That Triangle in the Filing Button (01/4/93)

Q: When a data item is on the card, the built-in applications put a little triangle in the filing button. How does my application update the filing button to show (or not show) this triangle?

A: You need to keep the filing button in synch with changes made to the data. The way to do this is to declare the filing button to your application and then send it an Update message. This will cause the filing button to look at the target slot in your application and set its state (triangle or not) accordingly.

Checking a Folder Exists (01/19/94)

Q: How do I make sure that an item from another Newton is filed in a folder that exists on this one?

A: There is a global function called `CheckThatFolderExists`. It takes one argument, the frame with the suspect labels slot. It will check that the folder exists. If it does not, the labels slot is changed to nil (that is, unfiled.)

Finding Folders and Folder Warnings (01/19/94)

Q: I see a number of folder names, but where are the folders? How do I modify them?

A: Modifying/Adding/Deleting existing folders is user unfriendly at best. There should be a very good reason for doing something like that, but for this discussion we'll assume you have a very good reason.

The folders are stored in the `userconfiguration` global in the `userFolders` slot. The slot holds a frame where the slot name corresponds to the symbol found in a labels slot in an soup entry. The value of each slot is a string which is the name of the folder, as seen in the user interface.

Warning: The maximum number of folders is 12.

AddUndoAction and PCMCIA (avoiding -10401) (2/25/94)

Q: I get -10401 when I tap Undo after pulling a card with my application on it. This happens even if I use EnsureInternal() on all the paramaters to AddUndoAction. What's wrong?

A: AddUndoAction is implemented in a slightly odd way. In addition to saving the message and arguments, it also saves the current context ("self") so that later it can send the message passed (the first paramater) to the appropriate place.

Because of this, references to your application (on the card) can stay in the system inside the Undo implementation's data structures.

To avoid this, it is necessary to flush the undo actions from the system. You can do this with the hitherto undocumented global function ClearUndoStacks(), called in your application's viewQuitScript.

ClearUndoStacks() [no paramaters] clears all undo actions from the system, even those which may be destined for other applications, so it should be used sparingly. The proper time to call it is *not* in your RemoveScript. Perhaps the best time to call it would be in your application's viewQuitScript; ideally it would only be called if your application had used AddUndoAction.

Exception |evt.ex|, -8003? (4/20/94)

Q: What does the exception -8003 mean? I'm trying to get undo to work and I'm also getting "Sorry there is nothing to undo" messages when tapping on Undo. This is after my application has called AddUndoAction without any complaints.

A: The -8003 error really means that there is nothing to undo. Keep in mind that you can not call AddUndoAction from within an UndoAction. Perhaps this is the problem.

AddUndoAction saves the current message context ("self") as part of the undo event, and that context is sent the action paramater when undo is tapped. If the context was a view, and the view was deleted, the system removes the undo action and there may be nothing to undo.

Check the Undo-Redo sample code to see how Undo is done.

DefaultFolderChanged does not exist (5/25/94)

The Newton Programmer's Guide erroneously describes the DefaultFolderChanged function as being available from NewtonScript. You cannot call DefaultFolderChanged from NewtonScript; your application must supply its own folderChanged method, as described on the pages 12-47 and 12-48.

UpdateFilter method of protoFolderTab (7/13/94)

The UpdateFilter message documented in the Newton Programmers Guide on page 12-50 is a method of protoFolderTab. The message must be sent to a view based on a protoFolderTab as in the following example:

```
// folderTabView is a view whose _proto is a protoFolderTab
folderTabView:UpdateFilter();
```

End of DTSQA

NEWTON USER INTERFACE

TABLE OF CONTENTS:

NEWTON USER INTERFACE........................1
Introduction..1
Map Application Guidelines..1
Check Boxes for Left-Handed Users (11/11/93)...3
How to Indicate Busy State? (6/7/94)..3
Pick Lists should not obscure origin button (6/24/94)...........................4
Making Tap and Double-Tap Do Different Things (7/25/94)................4

Introduction

This document addresses Newton User Interface 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.

Map Application Guidelines

The following guidelines are for map application designers. They discuss:
•how to design map application interfaces
•how to implement controllers and other functions related to map navigation
•what styles apply to designing user interfaces.

Map Content

Currently Newton has a 80-dpi black and white display, with no support for gray-scale or color. This means that the actual map pictures should be very clear, and have distinct elements, so a user can clearly see all the details. A crowded map with lots of filled pixels will make it hard for a user to navigate. Make sure that the digitized maps are of high quality.

Shading can help in discerning parts (such as water areas), but too much shading will make the map look very crowded, and it's hard to read labels printed on top of shaded (not to speak of blackened) areas. Also, LCD displays perform poorly with large shaded areas.

It it important to use a distinct and clear font. Simple fonts are better than fancy ones; because too many pixels in the font can be confused with map background areas. Helvetica is a very good font for map labels.

Bitmaps distorted to show non-regular elements (such as crooked streets) can cause distraction for the user. Whenever possible, stick to grid-oriented layouts.

Layers

To avoid having too much information on one map you can implement layers of information. The user interface could provide a way to add or swap layers, so that a user could select what was presented on the screen. Of course this requires that additional map information be stored on the card, and that the information between the layers be consistent.

A potential problem with the layer approach is that a user may want to see elements of two or more layers at the same time. Another problem might occur if labels on the different layers overlap. Th layering technique should be used on a case-by-case basis.

Overviews

Another technique to avoid a lot of information on the screen is to have an overview, with the ability for a user to zoom in to view the finer detail. If the zoomed area is just a bitmap enlargement of the same map however, the contents may not help the end user. It's better to provide a more detailed map of the same area.

Navigation Mode and Ink Mode

Sometimes a user may want to navigate around the map display, for instance, to scroll or to zoom in to a specific area. Your application might also provide a way to annotate map information by storing ink information drawn by the user. If this is the case, the application should provide a specific ink mode for drawing and storing ink information, and a navigation mode for navigating on the map using a pen. (Much like the Newton Book Reader)

Navigation - Sidewise

The recommended ways to navigate on the map (we assume that creative application developers will explore other user friendly concepts) are:

1. Use the BookMaker navigation button (see the Scroll Over sample). You might consider making the button float, so the end user could move it if needed, but this may complicate the interface and there is nothing wrong with a button locked to the lower left or right corner.

2. Use the hand metaphor and bitmap to scroll the map with a simulated hand movement (as in HyperCard and MacPaint) The Scroll Over sample illustrates this technique as well.

Should we use the universal scroll arrows on the Newton button bar? In general these are used to scroll to another section of the application, and in this case, a map is embedded inside the application, so using the (vertical only) scroll arrows may not occur to the user. It's probably best to avoid using the arrow buttons for map applications.

Navigation - Zoom In and Out

The currently recommended way is to use two special buttons that indicate zoom in and zoom out, and example set is shown below. With this button, tapping the left box (distant mountains) moves the user up, showing less detail, zooming out. Tapping the right box (near mountains) moves the user in, showing more detail.

You could also implement a power user zoom by zooming in with a single tap, and zooming out with a double tap. Note however that this should not be the only way to zoom in the application.

Should we use the universal overview button on the Newton button bar? The overview button indicates a mode that is a meta-view of the current view. Use the overview button if you want to show the end user a higher level view (single level) from which the end user could navigate to other parts of the map or application.

Indicators

Any special indicator symbols on the map (showing cities or other distinct elements) should not look like controllers. If they do provide control functionality, it should work. If there is no controlling, then tapping on an the indicator should do nothing.

Try to make the indicators look as much as possible like the real thing. For instance, a camping place indicator could look like a small tent instead of a triangle.

If the background is not too dark, try to fill the indicators so they are clearly drawn.

Check Boxes for Left-Handed Users (11/11/93)

Here's a technique for adapting to left-handed users, so that their hands won't obscure the views. First, as is suggested in Newton Interface Guidelines, place your display views close to the top of the screen. Next, add a preference checkbox (leftyFlag) to let users specify whether they want their displays located on the left or right side of the main view.

Just set up two extra Rectangle slots in your display view: viewBoundsLtHanded, and viewBoundsRtHanded. viewBoundsLtHanded will contain the viewBounds for left-handed users, and viewBoundsRtHanded has the viewBounds for right-handed users. Then all you need to do from your viewSetupFormScript is test leftyFlag, and assign the real self.viewBounds to either self.viewBoundsLtHanded or self.viewBoundsRtHanded. This is much less work (and takes less memory) than trying to calculate a new viewBounds dynamically using a script.

How to Indicate Busy State? (6/7/94)

Q: Does Newton have the equivalent of a busy cursor (wristwatch) or progress box? The nearest thing we've seen is the little messages that come up at the top of the Find dialog.

A: For situations like this we advise developers to display a text message like our "connecting..." or "sending...", (the way we let users know what's happening when they're sending something from the Out Box).

The application can display the status message in a container view that's already open, or it can display the message in a status slip.

A status slip does not need a title, but may have a text button named Stop if the application allows the user to stop the action in progress. A status slip should have a rounded-corner black border. On a Newton MessagePad, the border is two pixels thick. (It will end up looking like a large button. Please note that while the picture in the 0.5 UI Guidelines is incorrect, the description is correct.)

The status message is centered inside the status slip, and the message should have blank space above and below it so that users don't confuse the status slip with a button. The message begins with a present participle describing the lengthy action in progress (such as preparing, updating, or stopping); mentions the object of the action, if there is an object; and ends with three periods — not a period, a hyphen, a dash, or ellipsis points.

We don't want developers to use our alert box for this, because it it was designed to alert users to serious problems or errors -- it's a bad idea to re-use an element for a different purpose. It would probably also be a bad idea to invent a different kind of alert box -- it might conjure up the same expectations that users have about our alert box and thus create confusion.

Pick Lists should not obscure origin button (6/24/94)

Q: I have a picker button in my application. Sometimes the pick list I pop up with DoPopup is wide enough that the list can not pop to the left or right of the button. Instead it obscures the button. Is this OK?

A: It is better for the user to see where the popup comes from. In this case, you should pop the pick list below the button. You should also try to make the left edge of the picker flush with the left edge of the button. As usual, the button should remain highlighted while the picked is visible.

Making Tap and Double-Tap Do Different Things (7/25/94)

Q: I have a view that needs to process taps and double taps. I notice that the aeTap gesture is always sent before aeDoubleTap. I don't want this, since I want to perform different operations for each of these gestures. I can implement a delay when a tap gesture is seen, and not perform the tap operation until the double-tap time has elapsed, but is there a better way?

A: This same problem is often faced in Desktop computer UI design. The same solutions used there also apply to Newton.

The best solution is to avoid the problem entirely. The problem occurs when "tap" and "double tap" mean different things in your interface. This can be confusing for users, since timing becomes important, and unexpected things will happen if the user gets the gesture wrong. As a rule of thumb, the operation performed for "tap" should not be destructive, nor take you to another place (unless over an obvious navigation control.) If this rule is followed, then the second tap that makes the "double tap" gesture can initiate the double-tap action, building on top of the first tap.

In rare cases when this can't be done, the solution is to make sure a double tap hasn't happened before performing the single-tap behavior. (The Macintosh has an example of this in the Finder, when you click on the text of an icon a short time passes before the text becomes editable. The pause is there so you can click a second time and open the icon. If you wait too long for the second click, it instead moves the insertion point.)

Getting double-tap parsing "right" may take some work. The Q&A Written Input and Recognition item entitled "Double tap details" has some useful information if you wish to attempt this. However, it's a lot easier and more reliable to avoid the issue, and find some other interface to the operations.

End of DTSQA

UTILITY FUNCTIONS

TABLE OF CONTENTS:

UTILITY FUNCTIONS 1
Introduction..................................2
StuffPString & StuffCString are Broken (9/15/93)..................................2
International Currency Settings (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p15-23)..................................2
Unicode Character Information (9/15/93)..................................3
Discovering Which Fonts Are Installed (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p4-14)..................................3
Values of Face Slot Consts in fontSpec Frame (9/15/93) (Obsoleted by Newton Programmer's Guide p4-11)..................................4
Day Strings from the Toolbox (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p15-6)..................................4
TotalMinutes Mysteries (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p15-34)..................................5
Time Zones Info (9/15/93)(Obsoleted by 1.0 Newton Programmer's Guide p15-31)..................................5
GetDateStringSpec (11/11/93)..................................5
Converting Float Numbers to Integers (9/15/93)..................................6
Not A Number (NaN) and Infinity Testing (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-43)..................................6
Creating Delays (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-78)..................................6
Battery Level (10/11/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-71)..................................7
Bnot Documentation Error (10/9/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-26)..................................7
Finding the right Dial String (10/18/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-75)..................................7
Setting User Call Options (10/18/93)..................................7
Temperature Information (12/2/93)..................................8
StrReplace Hangs with Read-only Strings (12/11/93)..................................8
Stuff- Functions (12/16/93) (Obsoleted by 1.0 Newton Programmer's Guide chapter 17)..................................8
Erfc() Wrong (12/17/93)..................................11
Smart- String Functions (12/18/93) (Obsoleted by 1.0 Newton Programmer's Guide, chapter 17)..................................11
StrTruncate Problems (12/18/93)..................................12
Binary Object Allocation (12/18/93)..................................13
GetDateStringSpec Problems, kFormatShortTime (12/18/93)..................................13

StrMunger hangs with Read-Only Strings (12/18/93)............................14
RegisterAlarm (12/18/93)...14
StringFilter (2/18/94)...15
Compile Function Limits (2/23/94) 15
Compile Function Output Stored to a Soup (2/23/94) 15
NumberStr & FormattedNumberStr (3/10/94) 16
SmartString Warnings (4/22/94) 16
TextBounds New Function for Finding Text Size (6/9/94).......................16
StrCompare, StrEqual Differences,Exact String Compares (6/9/94)...17
Infinity Dangers in NewtonScript (6/17/94)..18
AddDeferredAction and Input Queues (6/17/94)...................................18
StringToNumber, NumberStr incorrect with 15+ digits before decimal (7/12/94) 18
CHANGED
TimeInSeconds zero value wrong on some ROMs (2/24/95)....19
NEW
IsReadOnly global function. (3/1/95) 20
NEW
Converting Single Floats To Double Floats. (3/1/95)....................20
NEW
ParamStr and paramaterless strings. (4/5/95) 21

Introduction

This document addresses Newton Utility Function 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.

StuffPString & StuffCString are Broken (9/15/93)

Q: I have had problems with the StuffPString and StuffCString routines. What could I be doing wrong?

A: StuffPString is currently undocumented. Do not use this function. The ROM StuffPString function stuffs strings properly, but unfortunately also changes an equal amount of memory after changing the string itself. This might cause frame heap problems later.

The ROM StuffCString function is also broken; it fails in some cases where it shouldn't. It does not normally modify memory improperly.

All the StuffXXX functions in ROM may overrun and modify memory improperly if you pass them a destination binary object that is too small (for example, size 0).

All the ExtractXXX functions in ROM may return undefined data if the source binary object you pass is too small (for example, size 0)

The compile time versions of all these functions are "fixed" in NTK 1.0 final.

International Currency Settings (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p15-23)

Q: What are the plans for handling international currency settings (number of decimal places, thousands separator character, decimal separator character, and so on)? Are there any plans for routines which will use this sort of international information, if it is available?

A: There are two ways to handle currency settings: one function, formattedNumberStr, formats floating point numbers; another function, GetCurrencyString, will return a currency prefix and a currency suffix.

Here is a sample function that takes a floating point number and returns a correctly-formatted currency string :

```
GetCurrencyString: func(x)
begin
local s;
   s := GetLocale().numberFormat.currencyPrefix;
   // format the number with 2 decimal places
   s := s & formattedNumberStr(x, "%.2f");
   s := s & GetLocale().numberFormat.currencySuffix;
end;
```

Note: formattedNumberStr requires the first argument to be a float.

Note2: formattedNumberStr does not work correctly in the b1 build.

Unicode Character Information (9/15/93)

Q: Where can I find more about Unicode tables?

A: The following book provides a full listing of the world wide (non-Kanji) Unicode characters:

The Unicode Standard
WorldWide Character Encoding
Version 1.0 Volume 1
ISBN-0-201-56788-1

Discovering Which Fonts Are Installed (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p4-14)

Q: How do I determine which fonts are available in the system?

A: You can use the global variable fonts to obtain a list of fonts that are currently available.

Fonts contains a frame that looks similar to the following example:

```
{_proto: {espy: {#1F3}, newYork: {#31B}, geneva: {#27F}}}
```

Thus, to obtain a list of font symbols for use in your fontSpecs, you can use the foreach operator to iterate over the list of slots in the global fonts (and its _proto chain.)

```
local f := fonts;
local a := [];
while f do
   begin
      foreach fontSym,font in f do
         SetAdd(a, fontSym, TRUE);
      f := f._proto;
   end;
```

This code fragment returns an array that is similar to the following example.

```
-> [espy, newYork, geneva]
```

Values of Face Slot Consts in fontSpec Frame (9/15/93) (Obsoleted by Newton Programmer's Guide p4-11)

Q: What are the literal values of constants used for the face slot in the fontSpec frame?

A: Here's a list of the constants:

```
normal          = 0
bold            = 1
italic          = 2
underline       = 4
outline         = 8
superscript     = 128
subscript       = 256
```

Do not use any condensed-and-extended values (if you are assuming that you could use those as well because the values are the same as in the Macintosh Toolbox). These constants are not supported in current and future platforms, so any code you write assuming that the constants work will most likely break.

Day Strings from the Toolbox (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p15-6)

Q: How do I get to the strings for the days of the week in ROM? (for example, "Monday", "Tuesday") I know they are in there and I do not want to hard code them in.

A: You can get the day strings from the ROM by using the GetLocale() global function. Here is the call and what it returns. (NOTE:The call returns information based on the Locale set by the user, this example shows the US locale):

```
GetLocale().longdateformat
#3AF625   {versionNumber: 1,
           longDofWeek: ["Sunday", "Monday", "Tuesday", "Wednesday",
      "Thursday", "Friday", "Saturday"],
           abbrDofWeek: ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri",
      "Sat"],
           terseDofWeek: ["Su", "Mo", "Tu", "We", "Th", "Fr", "Sa"],
           shortDofWeek: ["S", "M", "T", "W", "T", "F", "S"],
           longMonth: ["January", "February", "March", "April", "May",
      "June", "July", "August", "September", "October",
   "November", "December"],
           abbrMonth: ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
      "Aug", "Sep", "Oct", "Nov", "Dec"],
           longDateOrder: 11702986,
           dayLeadingZ: 1,
           longDateDelim: ["", ", ", " ", ", ", ""]}
```

TotalMinutes Mysteries (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p15-34)

Q: I have problems using the TotalMinutes function—every time I pass a value to this function I get a –48406 error (Expected an integer). What is going on?

A: Most likely you are passing an arbitrary frame, rather than a valid Date frame. The following code fragment reproduces this error.

```
z := {x:60};

y := totalminutes(z)
Exception: "evt.ex.fr.type;type.ref.frame",
{errorCode: -48406,
 value: NIL}
```

The point here is that you can't always expect the error codes to point out the problem directly. In this case, the function assumed that the frame passed had a valid structure, and when this was not the case it used a nil value for a calculation, thus causing the "expected an integer" exception.

Time Zones Info (9/15/93)(Obsoleted by 1.0 Newton Programmer's Guide p15-31)

Q: How do I set the "I'm Here" location in Time Zones or get the latitude and longitude?

A: You can set the Time Zones location by calling WorldClock:SetLocation(newLocation). "newLocation" is a frame with a name, latitude, longitude, country, and areacode. country and areacode are optional. You can get the current location using userconfiguration.location.

GetDateStringSpec (11/11/93)

Q: Does GetDateStringSpec work? I get a -48808 error (undefined global function) when I try to use it.

A: GetDateStringSpec is a compile-time function that is not currently available from NTK. You can add it, though. Paste the following into your Globaldata file:

```
func GetDateStringSpec( elemArray )
begin
   local spec, element;
   spec := 0;

   foreach value in elemArray do
      begin
         element := value[1] << kElementTypeWidth + value[0];
```

```
                spec := spec << kElementShift + element;
            end;
        spec
    end;
```

Please note that the array uses the kElement<foo> constants for the element type, and the kFormatDefault..kFormatNumeric constants for the format type. kFormatLongDate..kFormatSecond appear to be either totally unused or calculated improperly; I'd avoid them. Finally, kIncludeAllElements is used in the LongDateStr(), ShortDateStr(), and TimeStr() functions, and NOT in the GetDateStringSpec function.

Converting Float Numbers to Integers (9/15/93)

Q: How do I convert a float to an int? Using trunc seems to make an integer into a value that causes graphics functions to crash.

A: Trunc returns a floating point number with no decimal part, not an integer. You should use the RIntToL function instead. (Or some combination of Round, Floor, and Ceiling) The syntax is:

```
myInt := RIntToL(floatNumber);
```

Not A Number (NaN) and Infinity Testing (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-43)

Q: I need a function that can take a real and tell me if it's a NAN and/or an infinity.

A: Further information on the following functions will be included in the final release of the documentation.

```
isnormal(x)
isfinite(x)
isnan(x)
```

Creating Delays (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-78)

Q: What is the recommended way to create delays with this battery-based device?

A: Use the function called `Sleep`(ticks). Note that you should avoid using small tick numbers when it is unnecessary but you should not purposely "freeze" the Newton. If the sleep number is large, consider using an `IdleScript` instead of using `Sleep`.

Battery Level (10/11/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-71)

Q: How can I find how much charge remains in the Newton battery?

A: Use the global function BatteryLevel. BatteryLevel(0) returns values between 0 and 100, where 100 is a maximally charged system and 0 is minimally charged. However. do not rely completely on these values, as the battery charge will always fluctuate depending on age, battery type and other parameters.

Bnot Documentation Error (10/9/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-26)

The Alpha documentation states that Bnot takes two parameters. Bnot actually takes just one parameter.

Finding the right Dial String (10/18/93) (Obsoleted by 1.0 Newton Programmer's Guide p17-75)

Q: How do I get the correct sequence of numbers to dial given a phone number?

A: You can use the MungePhone method. See the LondonCalling sample code for full details.

Setting User Call Options (10/18/93)

Q: How do I do something like the Options slip that the calling slip uses?

A: You can use the GetRoot().callOptionsSlip directly to have the user set things. It will give you a callback to tell you things have changed. See the LondonCalling sample code for full details.

Temperature Information (12/2/93)

Q: Is there a way to get access to the temperature information provided in the status bar clock? (The temperature that appears when you hold down the pen for a longer time.)

A: The temperature returned is based on a sensor in the battery case, so it's not very accurate; it may even be completely worthless, depending on the particular platform you're running on. In addition, this function may not be available on all platforms.

In any case, you can get it with BatteryLevel(2). Here's the formula:

```
local celcius:= BatteryLevel(2) >> 16;
local fahrenheit := (celcius *9) DIV 5 + 32;
```

Note: The MessagePad 110 does not support this. On that machine BatteryLevel(2)>>16 evalutes to 0 and I can assure you it's not that cold in my office.

StrReplace Hangs with Read-only Strings (12/11/93)

Q: When I call StrReplace, the machine hangs. For example:

```
func()
begin
   local theStr := "0101010101010101";          // just for testing
   local theAns := StrReplace(theStr,"1","1",nil);    // we hang here!
   print(theAns);                               // never get here
```

```
    end
```

Everything works fine if I go into the Inspector window and enter:

```
    theStr := "0101010101010101"
    StrReplace(theStr,"1","1",nil)
```

A: You are passing StrReplace a read only object, in this case a string literal in your package. Instead of generating a –48214 exception, the system is hanging. We consider hanging the machine to be a bug in StrReplace, and will fix this. In any event, you shouldn't be passing it a read only object. For testing, you can Clone the string literal to obtain a writable string.

Stuff- Functions (12/16/93) (Obsoleted by 1.0 Newton Programmer's Guide chapter 17)

ExtractUniChar

- grabs two bytes at the specified offset and returns the char

```
ExtractUniChar("abc",0);
#616      $a
```

StuffUniChar

- stuffs two bytes at the specified offset
- accepts a char or int as its 3rd arg

```
x := "\u00000000";
StuffUniChar(x,0,"\uF00F"[0]);
x[0]
#F00F6    $\uF00F

x := "\u00000000";
StuffUniChar(x,0,0x0AA0);
x[0]
#AA06     $\u0AA0
```

ExtractChar

- grabs one byte at the specified offset, returns the two char unicode char it makes from it

```
ExtractChar("\uFFFFFFFF",0);
#2C76     $\u02C7

//Note $a is at offset 1 in a unicode string
ExtractChar("abc",0);
#6        $\00

ExtractChar("abc",1);
#616      $a
```

StuffChar

- stuffs one byte at the specified offset
- you pass it a two-byte unicode value, it makes a one byte char from that value. That's the value it stuffs.
- accepts a char or int as its 3rd arg

```
x := "\u00000000";
```

```
StuffChar(x,1,Ord($Z));
x[0]
#5A6      $Z

x := "\u00000000";
StuffChar(x,1,-1);
x[0]
#1A6      $\1A
ExtractByte(x,1)
#68       26
ExtractByte(x,0)
#0        0
```

ExtractByte
- returns an 8-bit unsigned value

```
ExtractByte("\uFFFFFFFF",0);
#3FC      255
```

StuffByte
- writes one byte at the specified offset
- writes the low-order byte of its 3rd arg

```
x := "\u00000000";
StuffByte(x,0,-1);
x[0]
#FF006    $\uFF00

x := "\u00000000";
StuffByte(x,0,0xFF);
x[0]
#FF006    $\uFF00

x := "\u00000000";
StuffByte(x,0,0xAAFF);
x[0]
#FF006    $\uFF00

x := "\u00000000";
StuffByte(x,0,0x3FFFFFBA);
x[0]
#BA006    $\uBA00
```

ExtractWord
-returns a 16-bit SIGNED value

```
ExtractWord("\uFFFFFFFF",0);
#FFFFFFFC -1

//if you want unsigned use:
0x10000 - ExtractWord("\uFFFFFFFF",0);
#40004    65537
```

StuffWord
-writes 2 bytes at the specified offset

-writes low-order two bytes of its 3rd arg

```
x := "\u00000000";
StuffWord(x,0,0x3FFF1234);
x[0]
#12346    $\u1234

x := "\u00000000";
StuffWord(x,0,-1);
x[0]
#FFFF6    $\uFFFF

x := "\u00000000";
StuffWord(x,0,0x3FFFFFFF);
x[0]
#FFFF6    $\uFFFF
```

ExtractLong

-returns a 30-bit signed value
-reads 4 bytes at the specified offset, but ignores the high-order bits (first two)

```
ExtractLong("\uFFFFFFFF",0);
#FFFFFFFC -1

ExtractLong("\uC0000007",0);
#1C       7
```

StuffLong

-writes 4 bytes at the specified offset
-it uses the 30-bit signed value you pass it, and sign extends it to 30 bytes

```
x := "\u00000000";
StuffLong(x,0,-1);
x[0]
#FFFF6    $\uFFFF
x[1]
#FFFF6    $\uFFFF


x := "\u00000000";
StuffLong(x,0,0x3FFFFFFA);
x[0]
#FFFF6    $\uFFFF
x[1]
#FFFA6    $\uFFFA
```

ExtractXLong

-????

```
ExtractXLong("\u0000000F",0);
#4        1
```

Erfc() Wrong (12/17/93)

Q: Erfc(x)--the complementary error function--is wrong between -0.4687 and 0.

A: Use 1.0-Erf(x) instead.

Smart- String Functions (12/18/93) (Obsoleted by 1.0 Newton Programmer's Guide, chapter 17)

The toolbox has the following fast string functions for concatenation and string handling:

SmartStart(len)

This function preallocates space for a string with a pre-defined size (meant to be used with SmartConcat) and returns the reference for this string . For example:

```
s := SmartStart(100)
#441AFF9  ""
StrLen(s)
#0        0
length(s)
#190      100
```

SmartConcat(text, len, addText)

This function takes an existing text reference, adds text — based on how many characters added — and returns the length of the new text object. For example:

```
l := 0;
l := SmartConcat(s, l, "foo");
#C        3
l := SmartConcat(s, l, "bar");
#18       6
s
#441D9E9  "foobar"
```

SmartStop(string, len)

This function changes the real physical length of the string, turning it into a normal NewtonScript string. For example:

```
length(s)
#190      100
SmartStop(s, l)
#2        NIL
s
#440B5C1  "foobar"
length(s)
#38       14
```

StrTruncate Problems (12/18/93)

Q: Why do I have problems with StrTruncate? I get a "frame expected" error when I use this function.

A: StrTruncate requires a 'current font' in order to work, and looks for that font in the viewFont of the current message context. viewFont in the Inspector usually evaluates to NIL, hence the error.

You can fix this by passing a message to a frame with a viewFont slot, for example:

```
{
   viewFont: ROM_fontsystem10,
   msg: func() StrTruncate("abcdefg",25)
}:msg()
```

You could also create a viewFont slot in the Inspector itself by evaluating something like:

```
viewFont := ROM_fontsystem10
```

Note that StrTruncate will return different results (truncate the string in different places) depending on the current font.

Binary Object Allocation (12/18/93)

Q: How do I allocate a binary object?

A: The quickest way to get a new binary object is:

```
myObject := SetLength(Clone(""),LengthInBytes)
```

Then, you can do something like:

```
SetClass(myObject, 'samples)
```

to set the class of the binary object properly. (The example above sets the class to that of a sound sample).

You can also initialize the binary object to hex by disguising the hex as a unicode string, for example:

```
ralph.icon := {
   bits:
      SetClass(SetLength(Clone("\u000000000004000A000A00080020001B54000000AA
      000000AA000000950000005507800004BF840004000A000870120008880200009740C000
      82030000800C000040100000402000004020000040400000404000004040000040800000
      0040800000408000000408000000\u"), 104), 'bits),
   viewBounds: RelBounds(0,0,19,22)
}
```

GetDateStringSpec Problems, kFormatShortTime (12/18/93)

Q: I have trouble getting GetDateStringSpec to return what I need. I'm trying to print a time in the format HH:MM AM/PM (for example, 5:30AM, 12:25AM) I have tried the following calls to GetDateStringSpec:

```
GetDateStringSpec([[kElementNothing,kFormatShortTime]]);
GetDateStringSpec([[kIncludeAllElements,kFormatShortTime]])
GetDateStringSpec([[kElementHour,kFormatShortTime], [kElementMinute,
   kFormatShortTime], [kElementAMPM,kFormatShortTime]]);
```

None of these returned the desired format, and the documentation is rather unclear about how exactly to use these parameters.

A: kFormatShortTime doesn't work as documented, and isn't intended to be used with GetDateStringSpec in any case. The constants are meant to be used where you would use a result from GetDateStringSpec, for example, calls to ShortDateStr or LongDateStr. Basically, everything in the docs from kFormatLongDate down should be ignored.

You can do everything you need using just kFormatShort, kFormatLong, kFormatAbbr, kFormatTerse, kFormatNumeric, and kFormatDefault.

In this case, the correct constant would be:

```
GetDateStringSpec([[kElementHour, kFormatShort],[kElementMinute,
kFormatShort],[kElementAMPM, kFormatShort]]);
```

StrMunger hangs with Read-Only Strings (12/18/93)

Q: When I use StrMunger the Newton crashes without an error. For instance:

```
// theEditText string variable
// woSlash integer Variable

StrMunger(theEditText,woSlash,1," ",0,nil);
```

A: Like StrReplace, The StrMunger routine has a bug such that if handed a read-only string it hangs instead of returning an error code. If you need to obtain a writable string from a string literal, you can use the Clone or EnsureInternal functions.

RegisterAlarm (12/18/93)

Q: The Dates application uses alarms but I could not find good documentation on how to use these alarms. How do I do alarms correctly?

A: The Dates application uses a function called RegisterAlarm (not currently documented). However because of a design flaw, only one alarm can ever be registered. The Dates application uses this alarm, so if you try to call the global RegisterAlarm yourself, you will break the dates application.

The latest NTK platform file now provide a library for setting your own alarms. See the FalseAlarm sample code for details. Be sure to thoroughly read all the documentation. Using the library requires installing code that will permanently take up about 4K of your user's internal store.

StringFilter (2/18/94)

Q: StrFilter crashes, what's wrong?

A: There's a bug in StrFilter on the original MessagePad. It can be worked around by declaring a special local 'allDigits' and setting it to 1 in any function that calls StrFilter. For example:

```
local allDigits := 1;
StrFilter(...)
```

Compile Function Limits (2/23/94)

Q: Are there any limitations to the string passed to the Compile function?

A: The following limitations exist:

a) The maximum size of strings is 2 raised to 24.
b) Temporary data structures used during the compile must fit in the NewtonScript heap.
c) The actual function produced must fit in the NewtonScript heap.
d) NewtonScript uses only 7-bit ASCII, do not pass strings containing unicode characters.

Compile Function Output Stored to a Soup (2/23/94)

Q: If I store functions created in the Inspector into a soup I get -48216 errors (out of frames heap), while if I use Compile to produce the function, I don't get any frames heap problems.

For example:

```
// bad case
myFrame:={foo: func() print(99)}
#4418281  {foo: <CodeBlock, 0 args #4418249>}
mySoup:add(myFrame)
Exception |evt.ex.outofmem|: (-48216) Ran out of Frames Heap
memory

// good case
myFrame.foo := compile("print(99)")
#4419521  <CodeBlock, 0 args #4419521>
mySoup:add(myFrame)
#44199C1  {foo: <CodeBlock, 0 args #44196C9>,
_uniqueID: 3}
```

A: Functions "close over" their current lexical and message context, including them in the code block produced. Within the Inspector, the lexical and message contexts are the "globals" frame on the Newton, which is very large. When the system attempts to DeepClone the frame prior to storing it in the soup, it's really attempting to make a copy of essentially the whole ROM in the frames heap.

The lexical and message contexts are defined to be empty for functions created with Compile, so duplicating these functions is not usually a problem. Functions created in NTK similarly have no lexical or message context (unless they are deliberately nested within another function.)

NumberStr & FormattedNumberStr (3/10/94)

In some ROMs, NumberStr and FormattedNumberStr can return strings like "42.", with a trailing decimal separator but no trailing digits. You might consider trimming the last character from the string to work around this problem. Doing so without testing for the specific problem will lead to worse problem when executing on Newtons that do not have this bug.

To properly work around this problem, you can test the last character of the string produced against the decimal separator from the current GetLocale frame, and remove it only if they match.

SmartString Warnings (4/22/94)

Q: When I catenate to the end of a string produced with SmartStart, the newly added characters do not appear. What's wrong?

A: The string returned from SmartStart has extra space allocated on the end of the string. The concatenation operator (&) does not handle these strings properly.

Always call SmartStop before treating a SmartStart string as a real NewtonScript string. The general string routines do not always behave as expected when passed a smart string. In particular, note:

```
s := SmartStart(100);
#4410AE9  ""
s & "A"
#4410F31  ""      // Oops!
SmartStop(s, 0);
#2        NIL
s & "A"
#44119C1  "A"
```

TextBounds New Function for Finding Text Size (6/9/94)

As of the MessagePad 110, there is a new call called TextBounds that will determine the rectangle required for a piece of formatted text. This function MUST be called from a script executing in the context of a view.

You can check for the existence of this function as follows:

```
if functions.TextBounds exists then
   TextBounds(...)
else
   MyOwnTextBoundsBasedOnWrapItUpSample(...)

TextBounds(<text>, <funky-font-frame>, <bounds>)
```

<text> the text to 'wrap'
<funky-font-frame> see description below (justification is ignored)
<bounds> bounds frame with width or height 0

<funky-font-frame> This frame contains both a font specification and, optionally, a justification, the format is:

```
{font: <font-frame>, justification: <justify-symbol>}
```

<font-frame>: must be a font frame as per NPG (NOT a condensed number)
<justify-symbol>: ['left|'center|'right] if slot not present or slot is nil, defaults to 'left

The function returns bounds that <text> would wrap into based on the width or height in the passed <bounds> frame which is non-zero. The <bounds> frame is destructively modified, with the width or height that was 0 set to the wrapped width or height.

If one of width or height are 0, the bounds returned reflects wrapping the text to fit the constrained (non-zero) dimension.

If both width and height are 0, the bounds set and returned is based only on the explicit carriage returns in the text.

See the sample code NewTextFuncs for an example

StrCompare, StrEqual Differences,Exact String Compares (6/9/94)

Q: StrEqual and StrCompare make (different!) case insensitive comparisons. Are there corresponding case-sensitive functions?

A: There is no case sensitive string comparison routine, but there may be in the future. In the mean time, you can define your own. Have it check for the future routine and use it if present. For example:

```
DefConst('kStrExactCompareFunc, func(s1, s2)
begin
   if functions.StrExactCompare then
      return StrExactCompare(s1, s2);

   local l := StrLen(s1);
   if l <> StrLen(s2) then
      return NIL;
   for i := 0 to l-1 do
      if ord(s1[i]) <> ord(s2[i]) then
         return NIL;
   return TRUE;
end);
```

Warning: StrCompare and StrEqual differ with respect to accented character. For example:

```
StrEqual("a", "á")
#2        NIL
StrCompare("a", "á")
#0        0
```

Finally, it's interesting to note that the NewtonScript language defines the operators <, <=, >, and >= on strings as content comparisons (equivalent to StrCompare.) = and <> are reference comparisons. For example, "a" < "b" is always TRUE.

Infinity Dangers in NewtonScript (6/17/94)

Q: There seems to be no way to say x := ∞ or x := NAN(40) in NewtonScript. I think the standard for that is "Infinity" or "Inf", and NAN(40) or just NAN or some such. (I usually don't care about the nan number and I lump them all together. It is important, however, that I can input both +∞ and -∞.)

A: You can use the floating point functions to produce the infinities. For example, log(0) will produce –INF, and asin(1.1) will produce INF. Be warned, however, that the floating point emulator in the ARM -based Newtons has problems with infinity comparisons, for example, asin(1.1) > 0 is NIL.

AddDeferredAction and Input Queues (6/17/94)

Q: If an action is added via AddDeferredAction, will it take precedence over a user action? Is there some way to "peek" into the input queue to see if something is already pending for the task?

A: The order of event processing can vary. There is no way to peek into the input queue to check for pending events.

AddDeferredAction will put a type of event in a first-in-first-out queue which also contains user actions. Generally, if you call AddDeferredAction and then the user clicks, the deferred action will occur first. However, because the system watches for user input in a separate process, it is possible for the separate "inker" process to receive user input before the NewtonScript process gets to your call to AddDeferredAction. In this case, the user event will be processed first.

StringToNumber, NumberStr incorrect with 15+ digits before decimal (7/12/94)

Q: StringToNumber appears to have a bug in which it returns the real 0.0 if the string passed in represents a number greater than 1e+14. For example:

```
//15 zeros to the right ofdecimal
StringToNumber("1000000000000000.0");
#440AB71  0.00000
/14 zeros to the right ofdecimal
StringToNumber("100000000000000.0");
#440AF01  1.00000e+14
```

There is a straightforward workaround that splits the string if StringToNumber returns real 0.0, converts each piece, then constructs the proper real. This solves the problem, but our question is: Can we rely on StringToNumber returning 0.0 in this case, or are their other quirks in it?

A: There is a bug in both StringToNumber and NumberStr that returns incorrect results when there are more than 14 digits to the right of the decimal.

You can use FormattedNumberStr instead of NumberStr, but there is no equivalent for StringToNumber. You also can not rely on this bug being present in future Newtons. You need to check and see if the bug exists, and if so, parse the string yourself:

```
constant kStringToNumberFail := "1000000000000000.0";

if (StringToNumber(kStringToNumberFail) < 1.0) then
begin
   // do something to fix the problem
end;
else
   // all is fine
```

CHANGED: TimeInSeconds zero value wrong on some ROMs (2/24/95)

Q: TimeInSeconds seems to return a value that's off by a few days. What's wrong?

A: On earlier ROM versions, (specifically the Original MessagePad and MP110), the function `TimeInSeconds` returned a value based on a starting time of January 3, 1993 at 10:09 AM rather than January 1 at Midnight. The function could be corrected by adding `209340` to the result, however, doing this on a machine with the corrected ROM (for example, the MP100) will give an incorrect time.

Commonly, the time in seconds is only needed to get the number of seconds into the current minute. `TimeInSeconds MOD 60` will be correct on all ROMs. To obtain an accurate count of the number of seconds between midnight January 1, 1993 and the current time, you can use this function:

```
NewTimeInSeconds := func()
begin
    local now := TimeInSeconds();
    now + if (Time() - 46811520) * 60 - now > 86400 then
                 209340 else 0;
end
```

86400 is the number of seconds in one day, 46811520 is the number of minutes between 1904 and 1993, and 209340 is the number of seconds from midnight January 1 to 10:09 AM, January 3.

Note: A previous Q&A release suggested using this code:

```
      (Time() - 46811520) * 60 + TimeInSeconds() MOD 60
```

This calculation could result in values that are up to 59 seconds off if the functions `Time` and `TimeInSeconds` were called in different minutes.

NEW: IsReadOnly global function. (3/1/95)

Q: How can I find out if an object is in the NS heap, and therefore writable, or in my package, and read only?

A: You can use the global function `IsReadOnly` to determine whether an array, frame, or binary object is writable. (Immediate values such as integers are always writable.) This is a different function than the store method `IsReadOnly`, and has different arguments.

`IsReadOnly` takes a single argument, which must be a referenced object, that is, an object whose primitive class is array, frame, or binary. It returns `TRUE` if the object is read only, and `NIL` if it is writable. To ensure that some object (like a viewBounds slot) can be modified, you should use something like this:

```
   if IsReadOnly(viewBounds) then
      viewBounds := Clone(viewBounds);
```

This function should not be used to determine the location of an object, that is, whether it is in the heap, in ROM, or in protected memory. The NewtonScript language could permit read-only objects in the NS heap, or writable objects that exist in other locations.

NEW: Converting Single Floats To Double Floats. (3/1/95)

Q: How can I convert a 4-byte Apple SANE-compliant single floating point data type into a NewtonScript real data type?

A: NewtonScript uses the Apple SANE double format for floating point numbers. These are implemented as 8-byte binary objects of class 'real and contain 1 sign bit, 11 bits of biased exponent, and 52 bits of fraction. In contrast, the Apple SANE single format contains 1 sign bit, 8 bits of biased exponent, and 23 bits of fraction.

The function below extracts the various fields from a 4-byte single and converts the value according to the formula for SANE single representation as found on page 2-16 of the Apple Numerics Manual (second edition). Note that, due to accumulative rounding errors in multiplication and division, an exact conversion is not possible for all values input.

```
DefConst(  'kTwoToThePostive23,
           SetClass( SetLength( "\u4160000000000000\u", 8 ), 'real ) );
           // scalb(1, 23)

DefConst(  'kTwoToThePostive23_Times_TwoToTheNegative126,
           SetClass( SetLength( "\u3980000000000000\u", 8 ), 'real ) );
           // scalb(1, 23) * scalb(1, -126)

DefConst(  'kPositiveInfinity,
           SetClass( SetLength( "\uFFF0000000000000\u", 8 ), 'real ) );

DefConst(  'kNegativeInfinity,
           SetClass( SetLength( "\u7FF0000000000000\u", 8 ), 'real ) );
```

```
DefConst(  'kQuietNaN,
           SetClass( SetLength( "\u7FFFFFFFFFFFFFFF\u", 8 ), 'real ) );

DefConst(  'kSingleFloatToDoubleFloatFunc,
           func(singleArray)
           begin
              // Converts a 4-byte SANE single into a NewtonScript
              // native 8-byte SANE double.
              // singleArray is an array of 4 elements where each
              // element is 1 byte of a 4-byte single precision floating
              // point number and is passed most significant byte in
              // element 0 to least significant byte in element 3.
              // Each element is assumed to be in the range 0 <= n <= 255.
              //
              // For example:  1.23 is passed in as [0x3f, 0x9d, 0x70, 0xa4]
              // Note that an array of bytes is used rather than an integer
              // because NewtonScript uses only 30-bits for integers.
              //
              // Refer to the Apple Numerics Manual (second edition p. 2-16
              // and 2-17) for more info.

              local positive := ( BAND( singleArray[0], 0x80 ) = 0 );
              local exponent := BOR(   BAND( singleArray[0] << 1, 0xfe ),
                                       BAND( singleArray[1] >> 7, 0x01 ) );
              local fraction := BOR( BOR(  BAND( singleArray[1] << 16, 0x7f0000 ),
                                           BAND( singleArray[2] << 8, 0xff00 ) ),
                                           BAND( singleArray[3], 0xff ) );
              local answer :=
                 if exponent = 0 then
                    if fraction = 0 then
                       0.0
                    else   // denormalized
                       fraction / kTwoToThePostive23_Times_TwoToTheNegative126
                 else if exponent < 255 then  // normalized
                    ( ( fraction / kTwoToThePostive23 ) + 1 )
                    * scalb(1, exponent - 127)
                 else
                    if fraction = 0 then
                       if positive then
                          return kPositiveInfinity
                       else
                          return kNegativeInfinity
                    else
                       return kQuietNaN;
              if positive then
                 answer
              else
                 0.0 - answer;
           end );
```

NEW: ParamStr and paramaterless strings. (4/5/95)

Q: `ParamStr` produces unexpected results, strings will missing characters, boxes, and occasionally even integers. What's wrong?

A: `ParamStr` has problems if there are no substitution symbols (^0, ^1, etc) in the string passed as the first argument and one or more strings are passed as substitutions. `ParamStr("replace ^0", ["this", "not this"])` is fine, `ParamStr("nothing to replace", [])` is fine, but `ParamStr("nothing to replace", ["this"])` will produce strange results. This situation is usually easy to avoid. By the way: the 1.0 docs imply that ParamStr modifies its first argument. This is not the case.

End of DTSQA

VIEWS

TABLE OF CONTENTS:

VIEWS 1
Introduction....................2
Sizing and Positioning The Base Application View (9/15/93)....................2
IrisOpenEffect versus ZoomOpenEffect (9/15/93)....................3
Base Application View Setup (9/15/93) (obsoleted by 1.0 Newton Programmer's Guide p3-5)....................3
tabSetup Functions and Context Issues (9/15/93)....................4
Lightweight clParagraphViews (9/15/93) (Obsolete, Documented in NPG 1.0 Final 4-15)....................4
Saving clEditView contents to a Soup (10/4/93)....................5
KeyboardButton (10/8/93)....................5
SetupPopMark and ClearPopupMark (10/14/93) 5
Grow Effect (11/10/93)....................6
viewFormat Value Changes (11/10/93) 6
GetHiliteOffsets (11/10/93) (Obsoleted by 1.0 Newton Programmer's Guide p2-89)....................6
Rejecting View Additions (11/10/93) 7
Resizing clParagraphView (11/10/93)....................8
Keyboard Icon Popups (11/11/93) (Obsoleted by 1.0 Newton Programmers Guide p2-100,103)....................8
Masking Clicks From Views (11/11/93) (Obsoleted by Newton Programmers Guide, page 2-15 and 2-16)....................8
clMathExpView, clMathOpView and clMathLineView? (11/12/93)..8
vfPen and vfInset -- Compile Time Only Functions (11/15/93).........9
Icon Animation (12/19/93) 9
TextChanged Method and Keyboards (12/19/93)....................9
KeyIn (12/23/93) 10
Custom Fills (1/4/94) (Obsoleted by 1.0 Newton Programmers Guide p2-32) 10
Telling if a View is Highlighted. (1/13/94) 10
PICTs in Popups (2/3/94)....................10
Sibling and Parent Justification (3/10/94)....................11
Warning Using a Method for keyResult (3/28/94)....................11
Numeric Input Fields, Spaces and Recognition (4/8/94)....................11
Scrolling clEditViews with SetOrigin (4/11/94) 12
view
Hilite Requires viewJustify (4/20/94)....................12
clParagraphViews Trim to Parent's Right Edge (4/20/94)....................12
Repeat Rate for clKeyboardViews (4/20/94) 12

How does Dragging Really Work? (6/7/94)....13
New call... DirtyBox (6/7/94)....13
Children Bigger than their Parents? (6/9/94) (Obsoleted by Newton Programmers Guide p.2-18)....13
Declaring Multiple Levels (6/9/94)....14
vjSiblingCenterV/vjSiblingCenterH and Larger Ssiblings (6/9/94)...14
keyHighlightKeys for clKeyboardViews (6/9/94)....15
View Borders and Dragging (6/9/94)....15
Constraints on KeyboardsSizing to the View (6/9/94)....15
Adding Editable Text to clEditViews. (6/9/94)....16
viewLineSpacing and User Font (6/10/94)....17
How to do a Spacer Row in a Keyboard (6/13/94)....18
Performance Issues when Dirtying Views (6/20/94)....18
Performance Issues with Filled Views (6/20/94)....19
How to Reflow Views (6/21/94)....20
Using Full Justification in Printing (6/21/94)....22
How to Use GetItemMark and SetItemMark (6/21/94)....22
Order of viewQuitScript execution (6/30/94)....23
Declared Views Are Always in Memory (6/30/94)....23
clEditViews and typing/PostKeyString (7/5/94)....24
Use UnicodeFS/GS or view
SetHilite to change caret position. (8/19/94)....24
Overriding parent-inherited view methods like Close. (9/22/94).....25
SetOrigin and shapes (or other non-views). (9/23/94)....25
ViewEffects and complicated viewQuitScripts. (10/17/94)....25
viewFrontMost Testing, GetView, scrolling (11/4/94)....26
Immediate Children of the Root View are Special (11/17/94)....26
ModalDialog and screen 'freezes' (12/1/94)....27
viewClickScript Inheritance and User Input (1/4/95)....27
CHANGED
Making clRemoteView work (2/2/95)....28
NEW
Using clRemoteView with clEditView data (2/2/95)....29
NEW
Order of view
ChildViewFrames result. (4/12/95)....30

Introduction

This document addresses Newton View system 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.

An entry that has changed since the last distribution of this document is distinguished by the prefix NEW at the beginning of the subject heading , or if the entry has changed, marked with CHANGED.

Sizing and Positioning The Base Application View (9/15/93) (Obsoleted by 1.0 Newton Programmer's Guide p2-18)

Q: How do I know the right rectangle size for my base application view? Are there any global variables I can use?

A: You can use information returned by the GetAppParams function to automatically configure the size of your application's base view for different Newton screen sizes. GetAppParams returns a frame containing the coordinates of the drawable area of the Newton screen. The return result from this function typically looks like the code fragment below.

```
{appAreaTop: 0,
 appAreaLeft: 0,
 appAreaWidth: 240,
 appAreaHeight: 336,
 buttonBarPosition: bottom}
```

You can use the appArea*XXX* coordinates to size your base application view relative to the size of the screen of the device on which your application is installed, as in the following code fragment. This code would normally be placed in the base view's viewSetupFormScript script.

```
// in your Project Data
constant kMaxAppWidth := 240 ; // MP100 size
constant kMaxAppHeight := 336 ; // MP100 size

// in your viewSetupFormScript

local aa := GetAppParams();
self.viewBounds :=
      RelBounds(aa.appAreaTop,aa.appAreaLeft,
      MIN(aa.appAreaWidth, kMaxAppWidth)
      MIN(aa.appAreaHeight, kMaxAppHeight))
```

Once you have sized your application's base view according to the screen of the Newton device on which it is installed, you can use the buttonBarPosition returned by GetAppParams to position your base view relative to the button bar. It is common for the main application view to be positioned flush with the button bar. The GetAppParams function returns the value of buttonBarPosition as one of the symbols 'top, 'left, 'bottom, or 'right.

IrisOpenEffect versus ZoomOpenEffect (9/15/93)

Q: I've tried using the IrisOpenEffect, but it looks exactly like the ZoomOpenEffect. Why?

A: The two effects really are quite different, although the difference is subtle.

IrisOpenEffect changes the size of an invisible "aperture" covering the view, revealing an ever-increasing portion of the full-size view as the aperture opens. ZoomOpenEffect, on the other hand, actually "grows" the image of the view from a point in the center until it fills the screen; that is, the entire view appears to expand from a point in the center of the screen.

IrisOpenEffect also has the revealLine bit set, causing a border to surround the animation. The revealLine bit is usually not set when using ZoomOpenEffect because the outer edge of the view is always on the edge of the animation. Set this bit only when the view being zoomed doesn't have a border.

Base Application View Setup (9/15/93) (obsoleted by 1.0 Newton Programmer's Guide p3-5)

Q: Which slots and scripts must be implemented in a custom prototype application template if templates capable of containing simple child templates are to be derived from it?

A: There are really only two requirements for a base application view.

• The Application box must be enabled (checked) in viewFlags.This determines where the system sends messages such as those used for scrolling.

• The view must store the symbol 'base in its declareSelf slot.

```
declareSelf := 'base;
```

The Close/CancelBox searches the view hierarchy to find something having a declareSelf value of 'base and closes it.

You need not be concerned with the _proto or _parent slots. The _proto slot determines the system prototype (such as protoApplication, protoCloseBox, and so on) used by the view, which determines its behavior and appearance. Because you are creating your own template , you do not need this slot.

The _parent slot affects template enclosure. Because the template you are creating is a base level template, it is attached to the root view; however, you must not set the value of the _parent slot directly. The Newton development environment does all the relevant setup for you.

tabSetup Functions and Context Issues (9/15/93)

Q: The tabSetup function that is passed to LayoutTable does not seem to have any context when it's called. In my case, I discovered that a variable I had defined globally—a slot in my base view—wasn't being found. What's the deal?

A: This is indeed the case with the tabSetup function. You have to pass the context with this call, as in the following example.

```
tabSetup: func(prototemplate, slot2, otherStuff)
begin
	// ...
end,
```

Lightweight clParagraphViews (9/15/93) (Obsolete, Documented in NPG 1.0 Final 4-15).

When you create a clParagraphView having a particular set of characteristics,the system automatically makes a lightweight clParagraphView for you. This view has less overhead and is faster than a full-blown clParagraphView. To create such a view, your clParagraph view should have the following characteristics:

• The vReadOnly flag is set.

• The vGesturesAllowed and vCalculateBounds flags are *not* set.
• There are no styles or tabs in the paragraph.

Saving clEditView contents to a Soup (10/4/93)

Q: How can I save the contents of a clEditView (the clParagraphViews and clPolygonViews containing text, shapes, and ink) to a soup and restore it later?

A: Simply save the viewChildren array for the clEditView, probably in the viewQuitScript. To restore, assign the array from the soup to the viewChildren slot, either at viewSetupFormScript or viewSetupChildrenScript time or later; then do RedoChildren.

You shouldn't try to know "all" the slots in a template in the viewChildren array. (For example, text has optional slots for fonts and tabs, shapes have optional slots for pen width, and new optional slots may be added in future versions.) Saving the whole array also allows you to gracefully handle templates in the viewChildren array that don't have an ink, points, or text slot. In the future, there may be children that represent other data types.

KeyboardButton (10/8/93)

Q: How do I make a button like the Keyboard button in the paperroll?

A: Make a picture button with viewFormat of 0 and icon set to the ROM object keybuttbitmap. In the buttonClickScript of the button, call alphaKeyboard:Toggle(). To get the other keyboards, use numericKeyboard, phoneKeyboard, and dateKeyboard.

SetupPopMark and ClearPopupMark (10/14/93)

The function SetPopupMark sets the check mark for an item in a list of popup items. You would use it in a pickActionScript like this:

```
// make a modifiable copy
labelCommands := Clone(labelCommands);

// check the item chosen
SetPopupMark(labelCommands, actionCode);
```

The function ClearPopupMark clears all the marks from popup items. You would use it in a script like this:

```
labelCommands := Clone(labelCommands);      // make a modifiable copy
ClearPopupMark(labelCommands, actionCode);  // clear all the marks
```

ClearPopupMark has the side effect of throwing away all slots in each item frame except the item slot. If you have unpickable items (a pickable slot set to nil), it would be changed to pickable!

Note also that if you want to use a mark other than the check mark, you need to do this programatically (as described in the discussion of DoPopup NPG page 2-64).

Grow Effect (11/10/93)

Q: I would like to resize a number of child views by dragging a 'hot spot' in the bottom right corner of the view, similar to the grow icon on the Macintosh. I don't see any way to create a grow rectangle while the user is resizing the child view. Is there a way to implement this? I'm trying to use the view:Drag method.

A: The view:Drag method does not really do what you want. It takes the bits on the screen inside the rectangle (the view boundary), and blits the contents to make it follow the pen (that is, the system captures the bits corresponding to the view and continually draws them to the screen based on the pen position).

The technique for this is to track the pen yourself using the viewClickScript, watch the values of GetPoint and StrokeDone, and draw an XOR:ed gray rectangle. See the "Size Does Not Matter" sample for more details.

Concerning the user interface, it is good policy to allow dragging when taps occur near the edge of the view, and to extend this to resizing at the lower right corner, and perhaps at any corner. Stay tuned for more user interface guidelines about this.

viewFormat Value Changes (11/10/93)

Q: How can I change the values in the viewFormat slot of a view without closing and re-opening the view?

A: Just use the SetValue function and the view will be redrawn for you with the new format settings. For example:

```
SetValue(myView, 'viewFormat, 337)
// 337=vfFillWhite+vfFrameBlack+vfPen(1)
```

Because `SetValue` calls `Dirty` for you, you don't need to `Dirty` the view when you use `SetValue` to do something the view system knows about, such as changing `viewBounds` or text slots in the view.

GetHiliteOffsets (11/10/93) (Obsoleted by 1.0 Newton Programmer's Guide p2-89)

Q: I would like to find out which item was selected in a view. How is this done?

A: Use a function called GetHiliteOffsets. You need to call this function in combination with the HiliteOwner function. When you call HiliteOffsets, it will return an array of array. Each of the outer array represents selected subviews, as in:

```
x:= gethiliteoffsets()
#440CA69    [[{#4414991}, 0, 2],
            [{#4417B01}, 0, 5],
            [{#4418029}, 1, 3]]
```

Each of the three return values contains three elements:

Element 0: the subview which is highlighted. This is usually a clParagraphView, but you must check this. clPolygonViews will not be returned here even if HiliteOwner returns a clEditView when a clPolygonView child is highlighted.

Element 1: This is the start position of the text found in the text slot of a clParagraphView.

Element 2: This is the end position of the text found in the text slot of a clParagraphView.

To verify that your view is a clParagraphView, check the viewClass slot of the view. The value returned (dynamically) sometimes has a high bit set that you need to take into consideration using a mask constant, vcClassMask, as in:

```
theViews.viewClass = clParagraphView  OR
theView.viewClass - vcClassMask  = clParagraphView
```

If a graphic is highlighted and HiliteOwner returns a clEditView, check its view children for non-nil values of the hilites slot.

Rejecting View Additions (11/10/93)

Q: How can I prevent a view from being added? I would like to control what is happening inside the viewAddScript.

A: The viewAddScript must make sure that a view is made from the template it's passed, either by doing it inside the viewAddScript, or by returning nil, so the view system will do the view. Currently, however, there's a bug in the system, so if you return true the application hangs.

Practically, all viewAddScript can do just now is to modify the template, and it should always return nil.

Here is a possible workaround: modify the template to be invisible, and return nil; then later remove the view that was made, perhaps using AddDeferredAction.

Resizing clParagraphView (11/10/93)

Q: I would like to add text into a clParagraphView and then later have the height of the view adjusted to show all of the text. How should I achieve this?

A: The WrapItUp sample shows how this is done.

Keyboard Icon Popups (11/11/93) (Obsoleted by 1.0 Newton Programmers Guide p2-100,103)

Q: I want to use a keyboard icon to popup a keyboard like in the Notepad. How do I do this without having to recreate my own keyboard? I just want to use the same keyboard that comes up when you double tap into the field.

A: In the root view there are four slots called 'alphaKeyboard, 'numericKeyboard, 'phoneKeyboard, 'dateKeyboard, and each contains a view for that keyboard. Use SetKeyView to set your view as the one that gets key events, then GetRoot().alphaKeyboard:Open(). (Replace alphaKeyboard with one of the other keyboards if appropriate.)

You might consider looping through the other three keyboards and closing them if they're already open. (The built in applications don't do this, but it looks strange to me to have more than one keyboard on the screen... both keyboards will work if this happens, it just looks weird.)

Masking Clicks From Views (11/11/93) (Obsoleted by Newton Programmers Guide, page 2-15 and 2-16)

Q: When I open a protoFloatNGo that covers some other picture buttons, the buttons still get click messages if I happen to tap on the correct part of the protoFloatNGo. In other words, this proto allows clicks to pass through. How can I correct this?

A: You have to set the vClickable viewFlag and provide a viewClickScript that returns TRUE in order to prevent a view from passing clicks through to views beneath it. See the Views Q&A "viewClickScript inheritance and user input" for some possible problems and the workarounds.

clMathExpView, clMathOpView and clMathLineView? (11/12/93)

Q: I found the following view class definitions in the NTK definition file: clMathExpView, clMathOpView, and clMathLineView. What are they? Are these private view classes, or can I use them?

A: Currently these view classes are unsupported, and DTS does not make any guarantee that they will work properly with the current Newton platforms. We will provide more information about these classes when they are supported.

vfPen and vfInset -- Compile Time Only Functions (11/15/93)

Q: How do I make vfPen(x) and vfInset(x) work? I keep getting 'undefined global function' exceptions.

A: These are compile time only functions. They work if you use them inside an Eval slot, but you can't use them in scripts during run time.

Page 2-31 in the Newton Programmers Guide talks about the vfPen values you get. If you want to set the pen size for a shape, I would suggest using the pensize slot of the styleFrame. In the case of pen sizes for views, the pen thickness is N pixels * 256. (See the View chapter for more details of how to create values for the view slots).

Icon Animation (12/19/93)

Q: Is there an elegant way to animate icons in the Extras folder on startup, the same way the inbox and outbox icons operate?

A: You need to create a mask for the icon, similar to the way ICN# resources work on the Macintosh. This alternate icon is also a PICT (such as the icon resource itself). This PICT should be placed in the same file as the application icon (using ResEdit for instance), and the PICT should have the same name as the application icon with the exception that a ! character should be at the beginning or end of the file name (myIcon and !myIcon).

TextChanged Method and Keyboards (12/19/93)

Q: Why is the textChanged method of a field called every keyPress on the keypads? I don't want to know what the user is typing until the user is done (closed the keyboard). Are there any workarounds?

A: This is not acceptable user interface design because the user does not ever need to close the keyboard. The user is allowed to keep the keyboard open, have several keyboards open (yes, this is possible), and to do other actions to a field during typing, such as dragging something from the clipboard.

Since the on-screen keyboards are not dynamic and not modal, tracking the actions of the keyboard other than typing is not supported.

You could theoretically create your own keyboard window, but this would not handle German systems, and you would need to build in a lot of dictionary features.

Otherwise, unfortunately, there is no workaround for this behavior. One suggestion is to change the design, so that you assume that text can change often and through many input methods (dragging, typing,recognition). Perhaps you could implement a 'Do Calculation' button or something similar, in order not to surprise the user in cases where you change things based on the input.

If your textChanged method is slow, you could set a flag true in your textChanged method, and during an idleScript you could do whatever you want. If you use this solution, make sure that you terminate the idleScript at the earliest possible stage, in order to save batteries. The same goes for initializing the idle script; just initialize it before you need this one to check the flags.

KeyIn (12/23/93)

Q: How can I open the keyboard view with the Option key down?

A: Use the global function KeyIn(keyCode, state). keyCode is a value representing the *physical* key to press, state is a Boolean, TRUE for down, NIL for up. Some useful keycodes: Option i s0x3A, Shift is 0x38, and Caps Lock is 0x39. You may need to send a Dirty message to GetRoot().alphaKeyboard.

Custom Fills (1/4/94) (Obsoleted by 1.0 Newton Programmers Guide p2-32)

Q: When, how, and where do I set the viewFillPattern slot so that I can have a custom fill pattern in my view?

A: The viewFillPattern should just be a slot in the view itself. The following works in an evaluate slot for horizontal stripes (very ugly):

```
SetLength(SetClass("\u00FF00FF00FF00FF", 'pattern), 8);
```

You could also get the pattern out of a resource.

There is a problem with the NTK 1.0b7 viewFormat slot editor; it does not properly set the vfFillCustom flag. To work around this, you'll have to set the viewFormat slot in an AfterScript instead. You'd probably have an AfterScript with something like this:

```
thisView.viewFormat := vfFillCustom + vfPen(1) + vfFrameBlack;
```

Telling if a View is Highlighted. (1/13/94)

Q: How can I tell if a given view is highlighted?

A: Check the vSelected bit in the viewFlags. vSelected should not be set by your application, but you can test it to see if a view is currently selected (that is, highlighted.) BAND(viewFlags, vSelected) <> 0 is TRUE if the view is selected.

PICTs in Popups (2/3/94)

Q: Can I put a PICT in a popup?

A: No. But you can use bitmaps derived from PICTs at compile time.

Popups, created through DoPopup, or protoPicker, will NOT handle PICT binary objects in their list of items.

However, they will handle bitmaps. The distinction is subtle, and important. If you use the call "GetPICTAsBits" to get the picture from the resource file, you will get a frame representing the picture. It will be converted from a PICT to a black and white 72-dpi bitmap on the Mac at compile time, and rendered on the Newton at 80 dpi at run time.

If you use "GetResource" or "GetNamedResource" you will instead get a binary object with the PICT data in it. This PICT can be rendered to the Newton screen by the Newton at run time. However, only clPictureViews have the code in them to properly display PICT binary objects.

Sibling and Parent Justification (3/10/94)

Q: My understanding is that I can simultaneously use both sibling and parent justification for a view and that if the view has no siblings (if it's a first child, or an only child) then its parent justification bits are used. However, this doesn't seem to work if I use full parent justification.

A: That's correct. If you're using a sibling justification then you can't use parent full justification along with it. You can use any of the other parent justifications.

Warning Using a Method for keyResult (3/28/94)

The Newton Programmer's Guide states that you can use a method for your keyResult in a Key Definitions array. What it does not tell you is that this method will be called every time the keyboard view redraws (things like an Open, SetValue, Dirty... will cause this).

Also, the method will be called either once or twice depending on the setting of the keyResultsAreKeycodes slot; true means call two times, otherwise call once.

If you use a keyResult method, make sure that it does not cause side effects (like printing characters to 'viewFrontKey).

Numeric Input Fields, Spaces and Recognition (4/8/94)

Q: I have a numeric input field. I discovered recently that if you want to write "12,345" but instead write "12 345" (as some people do), the text is recognized as "12 345" but StringToNumber on the text returns the value 12.

A: For numeric fields, it helps a lot if you set the vNoSpaces flag in the TextFlags (not viewFlags) slot. Setting this flag will solve your problem.

Scrolling clEditViews with SetOrigin (4/11/94)

Q: I am trying to scroll a clEditView in my application. The child views look correct after using the SetOrigin function but newly-created views and view dragging do not seem to account for the new origin coordinates. How do I scroll my clEditView?

A: The clEditView class was not designed to be scrolled, so SetOrigin does not currently work correctly with this view class. You can achieve the same effect by creating a simple clView, making your clEditView a child of the clView, and then scrolling the clView using the SetOrigin function.

view:Hilite Requires viewJustify (4/20/94)

Q: When I call the view method Hilite(TRUE) for some views, it doesn't work.

A: The view method Hilite requires that the highlighted view have a viewJustify slot. If no value for viewJustify is found in the view or its proto chain, Hilite can fail.

clParagraphViews Trim to Parent's Right Edge (4/20/94)

Q: I have a clParagraphView with centered contents that is partially off the right edge of its parent. At run time, the view is shortened so its right edge is the same as its parent's, which causes the contents to be centered in the visible area.
A: clParagraphViews are constrained by the system to not hang off the right edge of their parents. (Otherwise this might allow words to become inaccessible as the user edits the view's contents.)

If you do not want this behavior, make sure your paragraph view is a "lightweight" text view by making sure

- The vReadOnly flag is set
- The vGesturesAllowed and vCalculateBounds flags are NOT set
- There are no styles or tabs in the paragraph

Repeat Rate for clKeyboardViews (4/20/94)

Q: What is the key repeat rate for keyboards, and can I change it?

A: The keyboard repeat rate is set to 200 msec in the MessagePad product, and can not be changed from NewtonScript. It is a 200 msec delay from the time the keyPressScript returns to the time it is called again, so any activity performed in the keyPressScript can slow down the key repeat rate.

How does Dragging Really Work? (6/7/94)

Q: In the Notepad, the user can select and drag a word from one note to another. How can I implement this feature in my application?

A: Selecting and dragging out from a view is part of the built-in behavior for user-created children of a clEditView. This is also true when you have a text view with both the vClickable and vGesturesAllowed flags set. In this case, a text view is any view that descends from a clParagraphView (for example, protoStaticText).

Dragging into comes as part of editable clEditViews and clParagraphViews (and their descendants).

Dragging is already implemented; you can drag selected information (either text or graphics) from a view into any other type of view that supports it. In other words, you can drag a piece of text from the Notepad to anything in your application that supports written text. This all is handled by the system.

New call... DirtyBox (6/7/94)

A: With the MessagePad 110 (and the MP 100 with 1.3 ROM), there is a new call that you can use to dirty portions of views. This can save some update time. (circumstances to be investigated)

view:DirtyBox(box);

view - view that receives the dirtyBox message
box - rectangle to dirty in **global** coordinates

You can use DirtyBox anywhere you would use Dirty.

Note: You should test for the existance of this function before using it. Use this code:

```
if GetRoot().DirtyBox exists then
begin
      local theBox ;
      // figure out the dirty are in global coordinates and then
//do  .. your code here
      view:DirtyBox(theBox) ;
end;
else
      view:Dirty();
```

Children Bigger than their Parents? (6/9/94) (Obsoleted by Newton Programmers Guide p.2-18)

Q: On our main application, we opened a FloatNGo bigger than the application. The close box of this FloatNGo was outside of the application bounds. When users tap on the close box, the protoFloatNGo is not closed, and some other application gets the tap. What's wrong?

A: Even though views can sometimes draw outside their parents (which is not a guaranteed behavior, by the way) they cannot receive input when outside the bounds of their parent (and grandparent, and so on.)

If you need to make a floating window larger than your app, use BuildContext to create the view as a child of the root view. Save the view that is returned from BuildContext and make sure it is closed when your application quits.

Declaring Multiple Levels (6/9/94)

Q: Call the main application view viewA. ViewB is a child of viewA and is declared to viewA. ViewC is a child of viewB and is declared to viewB. ViewB and ViewC are both initially invisible. This causes the ViewC slot in viewB to be nil when the application is first run. Is there any way to access viewC without first opening and then then hiding it?

A: The built-in declare mechanism will not work without opening the view. The declared view frames are not created until the view they are declared to is opened. You may consider trying to declare viewC to viewA, but this will actually illustrate a problem with the declare mechanism--it can get confused in this case because viewC's parent (viewB) may not have been created when the view frame for viewC needs to be allocated.

Depending on what sort of access you need to viewC, you could choose alternative such as

- promoting the shared data from viewC to viewB, where it can be accessed.
- writing your own equivalent of the declare mechanism, with a slot called myViewC in viewB. Have viewC's viewSetupFormScript copy data from myViewC into the view frame being created.

vjSiblingCenterV/vjSiblingCenterH and Larger Ssiblings (6/9/94)

Q: When I use vjSiblingCenterV and a view is vertically larger than its prior sibling, the top edge of the larger view is aligned with the top edge of the sibling, not centered. How can I make my (larger) siblings center justified?

A: What you describe is a bug in the sibling center justification code. It incorrectly assumes that siblings should never be above or left of their prior sibling, even if they are larger.

Commonly you want all the siblings to be centered (or offset) along a single line, and this can be accomplished using only parent relative center justification. If you do require sibling relative center justification, you will need to write code in the viewSetupFormScript for the sibling that computes its viewBounds at run time. The previous sibling can be found as the last element in the :Parent():ChildViewFrames() array.

keyHighlightKeys for clKeyboardViews (6/9/94)

Q: The documentation suggest to me that the keyHighlightKeys slot can contain data about keys that should stay highlighted. How does this work?

A: The keyHighlightKeys slot is an array of references to keys that will be drawn as highlighted, as if held down. The keyHighlightKeys slot elements should be references to the keyResults slots from the keyDefinitions. For example: [myKeyboard.keyDefinitions[3][15]]

View Borders and Dragging (6/9/94)

Q: Is there a reason why a protokeyboard I have made won't drag if I click in the gray border of its base protoFloater? Ink goes straight through it to views underneath. If I click carefully in some white space in the protoKeypad, it will drag.

A: View borders are drawn outside of the bounds of the view itself, and so taps in the border area are actually taps outside the view. However, immediate children of the root view are handled specially by the view system. Unlike other views, their borders are considered part of the view and so taps in the borders will be sent to them.

One way to work around the problem is to make your (draggable) keyboard a child of the root view. A good way to create a view like this is to use BuildContext. Here's what I did in my code:

```
baseView:viewSetupDoneScript:= func()
begin
     self.myKeyboard := BuildContext(pt_myKeyboardProto);
end;
```

BuildContext will take care of making the root view the parent, and storing the reference in your baseview will make it available from anywhere in your application. Forcing the root view to 'adopt' your keyboard might seem a little rude, but there is a precedent - print formats actually have to be defined in the root view. This is a similar case. Make sure your viewQuitScript closes and nils out the keyboard, for example:

```
 myKeyboard:?Close(); myKeyboard := nil;
```

Another way to work around the problem is to encase your protoKeyboard in a slightly larger invisible clView which actually handles the dragging.

Constraints on KeyboardsSizing to the View (6/9/94)

Q: I am having a problem with dynamically adjusting the size of keyboards. According to the documentation, adjusting the size of my keyboard view should cause the keys to size correctly to the bounds of the view. This does not happen. If I set the viewbounds of the keyboard (a full alphanumeric keyboard) to anything less than 224x80, the keys scrunch up only taking up about half the view (horizontally). They seem to size fine vertically. Note: this happens even if I set the viewbounds to 222 (only 2 pixels shorter.)

A: It turns out the the documentation does not give the full story. The final size of the keys in a keyboard is constrained by the smallest fractional key unit width you specify in the keyboard. To understand this explanation, you really need to understand the explanation of key dimensions give in pages 4-26,8 of the 1.0 Newton Programmer's Guide.

In addition to calculating the size (in key units) of the longest key row, the clKeyboardView also finds the smallest key unit specified in the keyboard and uses this to constrain the final horizontal size. It works out a minimal pixel size for the keyboard and makes sure that the final keyboard size is an integral multiple of this value. For example, if the smallest size is 10 pixels, then the final keyboard can be 10 pixels or 20 pixels, but not 15 pixels. If the view is 15 pixels, the keyboard will be 10 pixels.

The calculation for this minimal size is:

$m = w * (1/s)$

m - minimal size
w - width of the longest keyboard row in key units
s - numeric equivelent for smallest keyboard unit specified in the keyboard:
(keyHUnit = 1, keyHHalf = 0.5, keyHQuarter = 0.25, keyHEighth = 0.125)

So for the ASCII keyboard, the longest row is 14 key units, the smallest key unit used is keyHQuarter, so the minimal width for the ASCII keyboard is:

$m = 14 * (1 / 0.25) = 14 * 4 = 56$ pixels.

The keyboard will always be an integral multiple of 56 pixels in width. Notice that 224 pixels is exactly 4 * 56. By changing the width to 223, the keyboard now becomes 168 pixels wide.

Adding Editable Text to clEditViews. (6/9/94)

Q: How can I add editable text to a clEditView? If I drag out a clParagraphView child in NTK, the text is not selectable even if I turn on vGesturesAllowed.

A: clEditViews have special requirements. To create a text child of a clEditView that can be selected and modified by the user (as if it had been created by the user) you need to do the following:

```
textTemplate := {
   viewStationery: 'para,
   viewBounds: RelBounds(20, 20, 100, 20),
   text: "Demo Text",
};
AddView(self, textTemplate);
```

The view must be added dynamically (with AddView), because the editView expects to be able to modify the contents as the user edits this item. The template (textTemplate above) should also be created at run time, because the editView adds some slots to this template when creating the view. (Specifically it fills in the _proto slot based on the viewStationery value. The _proto slot will be set to protoParagraph) If you try to create too much at compile time, you will get -48214 (object is read only) errors when opening the edit view.

The minimum requirements for the template are a viewStationery of 'para, a text slot, and a viewBounds slot. You can also set viewFont, styles, tabs, and other slots to make the text look as you would like. (See the Notarize sample code for additional relavant information.)

The way viewStationery is handled will change in future Newton versions, and we cannot guarantee that the above code will continue to work.

viewLineSpacing and User Font (6/10/94)

Q: I want to set the viewLineSpacing for my view based on the current font, which is the userConfiguration.userFont. But this is an integer. How do I get the viewLineSpacing?

A: The bit format of the integer form of the fonts is documented (see p. 4-11,14 of the 1.0 Newton Programmer's Guide,) so you just need to apply a little bit of bit-math to get the point size.

To get the "middle" 10 bits from an integer, do:

```
BAND(x>>10, 0x3FF)
```

To get an integer with the "middle" 10 bits from another integer, do:

```
BOR(BAND(x, 0x3FF003FF), y<<10)
```

For example:

```
BAND(simpleFont18>>10, 0x3FF)
#48       18

BOR(BAND(simpleFont18, 0x3FF003FF), 9<<10) = simpleFont9
#1A       TRUE
```

However, for viewLineSpacing you want to have the spacing based on the height of the font, rather than it's point size. The height includes space for descenders and leading. For example, "userFont18" actually has only 17 pixels of ascent, and both the 9 and 10 point fonts have a FontHeight of 12.

The global functions FontHeight, FontAscent, FontDescent, and FontLeading give you actual font measurements, and work with both integer and frame descriptions of the font.

How to do a Spacer Row in a Keyboard (6/13/94)

Q: I put a spacer row in my keyboard using the keySpacer descriptor, but sometimes my keyboard does not draw correctly. How do I fix it?

A: In certain circumstances, the keyboard view is misinterpreting the spacer row you added as the last row in the keyboard. This happens when another view causes the key rows up to the spacer row to become dirty.

The workaround is to specify your spacer row as a dummy key that is not drawn:

```
[keyVQuarter, keyVQuarter,
   nil, nil, keyFramed*1 + keyLeftOpen + keyRightOpen +
   keyTopOpen + keyBottomOpen + keyHUnit]
```

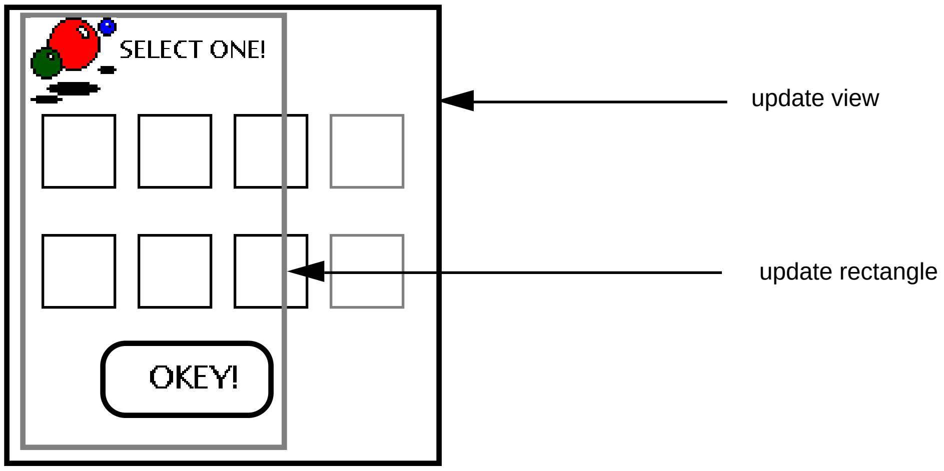
This gives you a single key that has no content, no action and a frame that is not drawn.

Performance Issues when Dirtying Views (6/20/94)

When you dirty a view -- for instance by using the SetValue call on a view that will cause the view to change -- the view is not drawn immediately. The system will keep track of what is dirtied. The system will also periodically perform an update on all the dirtied views.

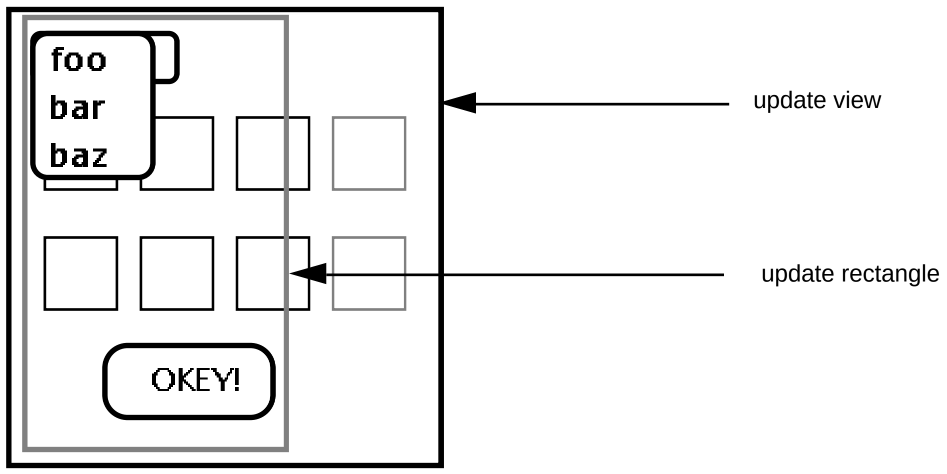
The view system will keep track of all the dirtied views by creating an enclosing rectangle of all such views. It will then just redraw views that intersect within this update rectangle. The view system also keeps track of a special view that it uses as a starting point when it looks for views to be redrawn. It will examine this view and all its descendants (children) and will look for views that intersect with the update rectangle. Any such view is redrawn.

When a view is dirtied the view system finds the closest common ancestor to the update view and the dirtied view and makes this specific ancestor the new update view.



If we do a SetValue or modify any of the upper views, and a SetValue on the button, then the update view is the parent of these views. When we do an update, all the views inside the update view will be drawn.

If we want to optimize the redrawing in this case, we could call RefreshViews directly after we called SetValue on the upper views. This way the upper level views will only be redrawn (update rectangle). Also the possible parent will eventually have less views to check than in the case of a big update rectangle. Similarly we could call Dirty and RefreshViews on the button at the bottom, we will get a much smaller update rectangle.



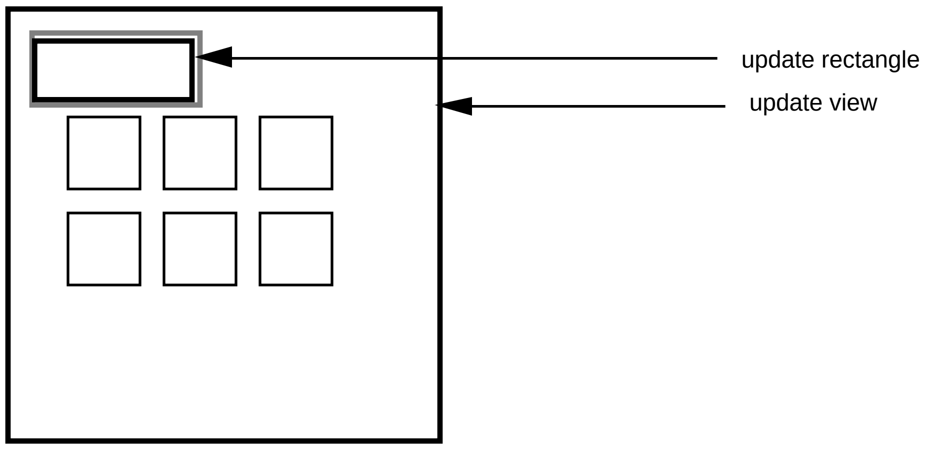
baz

In this second case we have a PickActionScript that will will also do a SetValue on the OKEY button. This means we have yet again a bigger update view and all the parent view are drawn. A possible optimization is to do a RefreshViews after the SetValue call in order to make update rectangle smaller (shaded rectangle in the picture).

A general rule is that if you update or modify one part of the screen you should consider using RefreshViews to avoid any of the parent views becoming the update view (in order to keep the update rectangle small). However, RefreshViews is an expensive call, so you need to do performance tests in order to figure out if this is an optimal solution. For instance, use Print statements with Ticks information inside your viewDrawScripts.

Performance Issues with Filled Views (6/20/94)

When you call dirty on a view, the closest filled view parent ancestor will become the update view. For instance the application base view is filled. This means that if the view system does not find an ancestor with a filled view, it will ultimately use the application base view as the update view! In other words, the parent as well as any overlapping children views will be redrawn.



If the top view is not filled, the update view will be the view that has the filled flag, in this case the heavy border view.

If the view is filled, the update rectangle itself is the only view actually updated.

You could provide any fill pattern to a view in order to make the updating faster. However, you don't want to provide fill patterns to every single view, becausue this increases the time it takes to draw the views. Consider providing fill patterns just to views that you want to dirty.

How to Reflow Views (6/21/94)

Q: Is there a way to reflow all my clParagraph, ink, and clEditViews so that the text wrap and resize correctly and views move upward to accommodate them?

A: Yes. The reflow routine is helpful for this. Sample code using this function will be available soon. Here is some preliminary documentation on Reflow:

```
Reflow( childrenTemplates, specs, srcBounds, destBounds )
```

- childrenTemplates	an array of input view templates (childTemplates)
- specs	a frame of options for reflowing
- srcBounds	the original bounds frame for the view templates (usually the value of :Parent:LocalBox())
- destBounds	the destination bounds frame which represents the size of the layout output. For instance, views which are centered will be centered between destBounds.right and destBounds.left.
- return value	an array of output view templates (outputChildren)

Reflow takes an array of child view templates and returns an array of templates where each outputChild has a _proto slot pointing to its corresponding childTemplate, whose bounds have been changed to fit within the specified destination bounds, and other slots will be added to override justification or formatting information. This function is used to expand a group of view templates, usually text, to fit within wider margins in preparation for printing or faxing (since the printing width is usually wider than the screen width). The order of the childrenTemplates is important. The views are reflowed within the destBounds based on the order in the childrenTemplates array. The return value of this function does not necessarily return as many entries as in the childrenTemplates array. If some children did not fit entirely within the destBounds, they are not included in the return value. (see "Multi-page printing using Reflow" below)

Behavior by child type:
• clParagraphView's left and right bounds are adjusted to fill the whole page. The bottom bounds are changed only if the view's vCalculateBounds bit is set in its viewFlags slot.
• All other views are horizontally centered within the destBounds rectangle.

If views overlap or are close to touching, they are considered part of a group of views whose coordinates should not be changed. If Reflow seems not to change the bounds coordinates of the new children, try making the children further apart. Also, check that the "graphicsGutter" and "textGutter" slots in the specs frame are set at appropriate numbers. If you set graphicsGutter to 100 and your original views are spaced with 80 pixels of vertical space between them, they will still be considered "overlapping" because 80 < graphicsGutter.

Specs is a frame with slots as follows:

reflowFont - optional. The font integer or font frame representing the font which will be used for determining text wrapping and display/printing. This is helpful if the display font is not available on printers. The font metrics will be used to determine text wrapping. If this slot is set to nil or does not exist, userFont12 will be used. If there is a 'styles' slot (see the clParagraphView documentation), the styles array in the new childFrame will be modified to accomodate the new font.

unistyle - If a childTemplate is a clParagraphView with a 'styles slot: •If unistyle is set to the symbol 'Font, then the styles array is used in the outputChild but all font families within the styles array are changed to the reflowFont. For instance, bold attributes will be preserved, but the font family and font size will be set to the font/size set in the reflowFont.

• If unistyle is set to the symbol 'All, then the styles slot in the outputChild is set to nil and the reflowFont will be the only font and size in this output child.

• If unistyle is nil, no modifications to the styles array is made. Any differences in font sizes and attributes will be preserved.

• If a childTemplate is not a clParagraphView or does not have a'styles slot, the setting of this slot has no effect on the corresponding outputChild.

graphicsGutter - The number of pixels above non-text children (clView, polygons, ink) which will count as part of the object. This is used to determine whether objects overlap. A standard number for this parameter is 16.

textGutter - The number of pixels above every text item baseline which will count as part of the the text child. This is used to determine whether objects overlap. A standard number for this parameter is -16.

viewlinespacing - The value of the viewLineSpacing slot in the new childTemplate views. Astandard number for this parameter is 30. During printing, the printer font used will sometimes modify the viewLineSpacing. (more details are needed)

How to use it in your print format's viewSetupChildrenScript:

```
     local kids := ReFlow(stepchildren, {unistyle: nil, graphicsGutter:
   16,
                 textGutter: -16, viewlinespacing:30}, fields.targetBox,
                 self:LocalBox());

     if kids then stepChildren := kids;
```

Multi-page printing using Reflow:
Reflow might return fewer children than you passed in the childrenForms array. If you pass 10 children to Reflow and the outputChildren array contains only 4 elements, then if you are doing multi-page printing, you should create your own reflowStartAt slot so that you can ensure that you start printing the next page with the 5th child. To do this, initialize the reflowStartAt variable to 0 before printing, then in your printNextPageScript, increment reflowStartAt by the number of elements in the outputChildren array. Do not pass all of your stepChildren to Reflow. The elements of the childrenTemplates array should only include the children who have not been reflowed yet. That will ensure you will only layout the current page's children.

Using Full Justification in Printing (6/21/94)

Q: It seems like full justification does not always work for print layouts. Why?

A: The main print layout view does not necessarily work with "full" justification because of a problem with the way that printing margins were designed. In order to obtain the correct bounds of the print layout, a child of the print layout can use the code in its viewSetupFormScript:

```
        self.viewbounds := :Parent():LocalBox();
```

to obtain the full bounds of the print layout including the correct margins. Children reflowed using the Reflow function have a similar problem. Consequently, do not set vertical or horizontal "full" justification in the viewJustify slot or else the view will extend beyond the edge of printable region of your page. You can use the :Parent():LocalBox() code to change view size in a view's viewSetupFormScript.

Child views of reflowed children should not have this problem.

How to Use GetItemMark and SetItemMark (6/21/94)

Q: When I use the DoPopup auxillary methods SetItemMark and GetItemMark, I get an error. What am I doing wrong?

A: According to the current documention, the return value of DoPopup is the picker view. The return value is actually a template created by DoPopup which is used to create the picker view. The SetItemMark and GetItemMark methods are not methods of the return value of DoPopup because these functions requre the object to be a view. (For instance, it calls SyncView which will not work if it is just a template). You can use GetView to actually find the view which responds to these methods. Although pickactionscript should normally be defined in a normal view script editor, here is some Inspector code which illustrates the use of these methods:

```
   self.pickactionscript := func(item) print("picked item: "
         && item);
   x := dopopup([{item: "one", mark: $\u221A, pickable: true},
       {item: "two"}], 0,0,self);
   GetView(x):SetItemMark(1, $@);
```

If you wanted to get the character mark for the first element (an offset of zero)

```
   GetView(x):GetItemMark(0);
```

The arguments for the *ItemMark functions are currently undocumented. SetItemMark takes (itemnumber, character) and GetItemMark takes (itemnumber) where itemnumber starts from 0. Note that the GetView(aTemplate) function sometimes must search through the entire view hierarchy to find the view, a process which may take a long time.

Order of viewQuitScript execution (6/30/94)

Q: When a parent and its children are closed, their viewQuitScripts are not run in any useful order. The manual says not to rely on children existing when a viewQuitScript is called, but can I rely on the parent view still being there?

A: The order in which the viewQuitScripts are called is undefined. You can rely on the view that is being sent the Close message having its viewQuitScript called first, with its children still intact, but the viewQuitScripts for the children are called when the children are actually disposed, in an undefined order. Note that this is a side effect of disposing the view class objects and might change on future Newton products.

If you need to do cleanup processing that relies on children existing, you can create your own viewQuitScript-like script (called, maybe, viewCleanupScript), that can call the viewCleanupScripts for the children recursively and in whatever order is appropriate. For example:

```
    foreach child in :ChildViewFrames() do    child:viewCleanupScript();
```

Declared Views Are Always in Memory (6/30/94)

If you close a declared view, but the view in which it's declared is still open, then the view memory object for the closed view will still exist. For instance, if the closed view contains big data objects, they will continue to take space in the NewtonScript heap.

To avoid this situation, nil out any such slots in the viewQuitScript method of the view. If you need to re-declare these slots, use the viewSetupFormscript method.

In general try to nil out any slots that are not needed in order to minimize use of the NewtonScript heap.

clEditViews and typing/PostKeyString (7/5/94)

Q: When I call `PostKeyString("sometext")` and the destination view is a `clEditView`, only the first character of the string appears in the edit view. However, if I add a `viewChangedScript` to a `clEditView`, and then type some keys using a built in keyboard, the script does not get called for the first key, but gets called for every character after the first.

A: `ClEditViews` function by creating children to hold the data. When you tap on a `clEditView` and type, a new `clParagraphView` child is immediately created to hold the text. Sometimes when you write or type "close enough" to an existing `clParagraphView` child of a `clEditView`, the new word or letters are added to that child and no new child view is created.

`PostKeyString` sends one character at a time to the view you pass it. When a `clEditView` recieves the first character, it creates a new `clParagraphView` to hold it and makes that new view the `'viewFrontKey`, so subsequent characters will go to it. The `clEeditView` then stops accepting key input, and so the rest of the string is lost.

You can work around this by calling `PostKeyString` in a loop. For example:

```
local s := "sometext";
for i := 0 to StrLen(s)-1 do
    PostKeyString('viewFrontKey, s[i]);
```

When a `clEditView` creates a new `clParagraphView` child to hold new text, the `viewChangedScript` is not called. Subsequent text goes to the new child, which inherits the `viewChangedScript` from the `clEditView`, and so the script appears to be called for following keys. However, the message context (`self`) within the script will be the child `clParagraphView`, not the `clEditView` as you might expect.

The `viewAddChildScript` for the `clEditView` will get called when a new child is created, so by watching both this script, the `viewDropChildScript` for deletions, and the `viewChangedScript`, you can reliably detect all keys pressed.

Use UnicodeFS/GS or view:SetHilite to change caret position. (8/19/94)

Q: How can I move the insertion point in text?

A: You can move the insertion point forward or backward a character at a time by using `PostKeyString` to send strings consisting of the characters `unicodeFS` (backward) or `unicodeGS` (forward). To move the insertion point to a given character offset, use `view:SetHilite` with the same character offset for the start and end range.

Examples:
```
PostKeyString('viewFrontKey, unicodeFS&""); // move caret back a space
PostKeyString('viewFrontKey, unicodeGS&""); // move caret forward a
space
GetView('viewFrontKey):SetHilite(0,0,TRUE); // move caret to beginning
```

Overriding parent-inherited view methods like Close. (9/22/94)

Q: How can I override a view method such as `Close`. If I create a new method on my view called `Close` I can't call `inherited:Close` from within it, so now my view can't be closed.

A: The problem with `inherited:Close` is that the inherited feature does not search the `_parent` chain, and the real `Close` method is implemented using the `_parent` chain, not the `_proto` chain.

Replacing the view methods is no problem, and while sort of an advanced technique, it can be done. You just have to use a trick to call the original function. Create a new slot in the view frame which contains the value from the (`_parent`) inherited method. For example, your close method might look like this:

```
func()
   begin
      // do some processing...

      // call the inherited close method.
      self.inheritedClose := GetVariable(:Parent(), 'Close);
      :inheritedClose();
   end
```

SetOrigin and shapes (or other non-views). (9/23/94)

Q: I have a `clView` with some shapes in it (drawn in the `viewDrawScript`). When I do a `SetOrigin` on the `clView`, the shapes do not move. Is `SetOrigin` broken?

A: `SetOrigin` only works on view children. It will not offset shapes. However, you can use a trick to scroll for you. Use two `clViews`, one inside the other. The inner one contains the pictures, that is, the `viewDrawScript`. It is arbitrarily large.

Send the `SetOrigin` message to the outer `clView` and the inner one will scroll, which is what you want. Note that you probably do not need to send the dirty message.

ViewEffects and complicated viewQuitScripts. (10/17/94)

Q: My application does a lot of things in its `viewQuitScript`. Consequently, my view appears to close immediately but then the Newton appears dead during the long calculations. I would like the view to look "open" until the `viewQuitScript` is done. How can I do this?

A: When your view has a "view effect", the close effect currently occurs as soon as the user taps the close box. It might be that your application has a `viewEffect` slot. If you want the view to appear open until after your "closing" code has executed, you could:

a) Put your code in the `viewHideScript` instead of the `viewCloseScript`
b) Remove the view's viewEffect and manually force the the effect at the end of the

viewQuitScript using the Effect command.

'viewFrontMost Testing, GetView, scrolling (11/4/94)

Q: What's the best way to direct scrolling messages to something other than my application's base view?

A: Scrolling and overview messages are sent to the frontmost view; this is the same view that will be returned if you call `GetView('viewFrontMost).`

The `viewFrontMost` view is found by looking recursively at views that have both the `vVisible` and `vApplication` bits set in their `viewFlags`. This means that you can set the `vApplication` bit in a descendant of your base view, and as long as `vApplication` is set in all the views in the parent chain for that view, the scrolling messages will go directly to that view. The `vApplication` bit is not just for base views, despite what the name might suggest.

If your situation is more complex, where the view that needs to be scrolled cannot have `vApplication` set or is not a descendant of your base view, you can have the base view's scrolling scripts call the appropriate scripts in the view you wish scrolled.

Immediate Children of the Root View are Special (11/17/94)

Q: In trying to make a better "modal" dialog, I am attempting to create a child of the root view that is full-screen and transparent. When I do this, the other views always disappear, and reappear when the window is closed. Why?

A: Immediate children of the root view are handled differently by the view system. They cannot be transparent, and will be filled white unless otherwise specified. Also, unlike other views in the present Newton OS, their borders are considered part of the view and so taps in the borders will be sent to them.

This was done deliberately to discourage tap-stealing and other unusual view interaction. Each top level view (usually one application) is intended to stand on its own and operate independently of other applications.

So-called "application modal" dialogs can and should be implemented using the technique you describe with the transparent window as a child of the application's base view.

ModalDialog and screen 'freezes' (12/1/94)

Q: I use a ModalDialog for renaming something in my application. If the user clicks the CapsLock key, some Newtons freeze and require a restart. I also get the same problem if I open up a protoFloatNGo.

A: ModalDialog works by limiting input to the topmost view. On older units, the CapsLock key will open a small view in the top left corner of the Newton. This prevents click input (though you can still write into the key view). You can get a similar problem by selecting text and placing it on the clipboard. Also note that if you open a keyboard, you will not be able to click any buttons in your modal dialog.

In fact, if you open any view that comes up on top of the modal dialog (that is, any view with vFloating set), all screen input will be redirected to the new view.

The only solution is to not use ModalDialog for dialogs that require text input via keyboard.

viewClickScript Inheritance and User Input (1/4/95)

Q: I have some `protoInputLine` children of a `protoFloatNGo`. When I make the floater draggable by turning on `vClickable` and adding a `viewClickScript` which calls `Drag`, my input lines also become draggable and do not take input. How can I make the floater draggable?

A: The problem is caused because `viewClickScript`s are inherited via the `_parent` chain, so the input line (which has to have `vClickable` flag set to enable recognition) now appears to have a `viewClickScript`. The `viewClickScript` usually does something that gets in the way of user input, or returns `TRUE,` which ends the input cycle.

You might consider working around the problem by adding an explicit `viewClickScript` to the input view, which does nothing but return `NIL`. While this does prevent the parent's `viewClickScript` from being inherited by the input line, it will not solve the problem, because the `NIL` return value tells the view system to allow input to "fall through" to the parent view. In the drag case, adding such a `viewClickScript` to the input field will prevent the input area from dragging, but will cause the parent to be dragged when you try to write in the input area.

You might also consider working around the problem by adding an explicit `viewClickScript` to the input view which returns `true`. While this will prevent the input from falling through to the parent view, it also tells the view system that you have handled the input, and so recognition will not occur.

A practical workaround is to create a new child of the parent view to do the parent's input handling. That is, create a sibling of the input line which appears first in the `stepChildren` list, and so is "below" the input line. This new child can be a simple `clView` which entirely covers the parent view. (Hint: use parent full justification.)

This new child can now have the `vClickable` flag set and can provide a `viewClickScript` which may have side effects or return `true`. Because it is not in the `_parent` chain for the input line, neither the inheritance problem nor the tap fall-through problem are encountered.

In the dragging case, the new child's `viewClickScript` might look like this. Note that we are sending the Drag message to the common parent of both the input line and the "tap-catching" child.

```
func(unit)
begin
   :Parent():Drag(unit, :Parent():Parent():GlobalBox());
   true;
end
```

Similar problems exist for the `viewStrokeScript`, `viewGestureScript`, and `viewWordScript`, and the work around is similar.

CHANGED: Making clRemoteView work (2/2/95)

Q: How do I use `clRemoteView`? I get a "`Exception |evt.ex.abt.bus|: 8`" when I try to open a `clRemoteView` a second time (after closing it once).

A: You might well ask "How do I use `clRemoteView?`" <I just did.> ...and well you might. The documentation does not quite cover all the restrictions of the `clRemoteView` class.

The `clRemoteView` class draws its first child view scaled so that child's bounds fit exactly within the remote view's bounds. With versions of the operating system up to and including 1.3 , `clRemoteViews` can only be used to shrink the view; they cannot be used to magnify it. The remote view simply does not show up if it is larger than its child.

The `clRemoteView` works by setting up drawing transform information when it is opened. This transform is based on the remote view's global coordinates and on the child view's global coordinates. The transform is used when drawing the view's child, which results in scaling.

Because `clRemoteView` bases its transform only on the first child, and draws only the first child, errors result if you try to give a `clRemoteView` more than one or less than one child.

A `clRemoteView` only sets up this transform information when it is first opened; after it has been opened, moving the view or the child causes strange results.

A common `clRemoteView`-related misconception is that you can point the remote view at an existing view; that is, that you can put some other view (not template) in the remote view's `viewChildren` or `stepChildren` slot, and the remove view will scale that other view into itself. This is false. Just like any other view class, a `clRemoteView` creates its children when it is opened, and closes them when it is closed. If you try to put a view frame (as opposed to a template) in the `viewChildren` or `stepChildren` slot of a remote view, you'll get bus errors and other view system exceptions. Don't do this.

NEW: Using clRemoteView with clEditView data (2/2/95)

Q: What's the best way to set up a clRemoteView so that it scales a clEditView and the children? When I try it, sometimes text doesn't show up.

A: Take a look a the "Thumbnail" sample from the DTS group.

It's common to want to set up the contents of a `clRemoteView` at run time, possibly based on the contents of some other view, like a `clEditView`. Print Preview, for example, uses a `clRemoteView` with a print format child to show a thumbnail view of the printed page.

Recall that `clEditViews` work by creating new `clParagraphView` and `clPolygonView` children to hold the text, shapes, and ink that users create. You can get templates for these dynamically created children by looking in the `viewChildren` slot of the `clEditView`.

To make a `clRemoteView` reflect a "live" `clEditView`, you can share the child templates between the edit view and the remote view. That is, when you open the remote view, set its child's `viewChildren` slot to the same array that is in the `clEditView`'s `viewChildren` slot. Normally this could be quite a problem, but `clRemoteViews` are "read only" and so you don't need to worry about both views trying to update the same array.

It's actually more complicated than this, because of a `clRemoteView` interaction with a view system optimization. The optimization is done because `clParagraphViews` are expensive, and the view system attempts to avoid drawing them when they are not visible. To figure out if a view is visible, the current operating system looks at the bounds of the view's parent and the view's grandparent. (Only two layers are checked--another optimization.) If the paragraph view is not within the bounds of its parent or grandparent, it is not displayed.

Unfortunately, the `clRemoteView`'s bounds are usually significantly smaller than the bounds of its child. It must be that way for scaling to happen. The view system optimization that prevents drawing clipped views doesn't take into account the transformation that `clRemoteViews` add, and so text views inside the `clRemoteView`'s scaling child may not show up.

To work around this optimization (which really helps most places) you can add two new layers between the remote view and the scaling child. These wrapper views provide a parent and a grandparent to the scaling child, which fools the view system into thinking that all the children of the scaling view really are visible, and so they're drawn, and scaled into the `clRemoteView`.

These wrapper views aren't used for anything else, they're just there to fool the view system optimizer, so it's best to make them as lightweight as possible. Plain old `clView`s with minimal formatting work well. This doesn't make a noticable impact on performance.

Here is an example of a `viewSetupChildrenScript` from a `clRemoteView` which kidnaps the templates for user-added children from some other `clEditView`.

```
func()
begin
   local box := theEditView:LocalBox();
   self.viewChildren := [
      {  // wrapper layer 2
         viewClass: clView,
         viewBounds: box,
         viewFlags: vVisible,
         viewChildren: [
            {  // wrapper layer 1
               viewClass: clView,
               viewBounds: box,
               viewFlags: vVisible,
               viewChildren: [
                  {  // this is the "real" container for edit view
stuff
                     viewClass: clView,
                     viewBounds: box,
                     viewFlags: vVisible,
                     viewFormat: theEditView.viewFormat,
                     viewLineSpacing: theEditView.viewLineSpacing,
                     viewChildren: theEditView.viewChildren,
                     }
                  ]
            } // end wrapper 1
         ]
      } // end wrapper 2
   ];
end
```

NEW: Order of view:ChildViewFrames result. (4/12/95)

Q: What is the order of the views returned by `view:ChildViewFrames`?

A: `view:ChildViewFrames` returns the views in the same order they appear in the view heirarchy, from back to front. The most recently opened views (which appear on top of the heirarchy) will be later in the list. Views with the `vFloating` flag (which always appear above non-floating views) will be at the end of the array.

End of DTSQA

WRITTEN INPUT AND RECOGNITION

TABLE OF CONTENTS:

WRITTEN INPUT AND RECOGNITION....1
Introduction....1
Dictionary Selection (9/15/93)....2
Create Custom Dictionaries? (9/15/93)....2
Control of Dragging and Scrubbing (10/20/93) (obsoleted by Newton Programmers Guide 1.0, page 6-5)....2
Sloppy Clicking (10/20/93)....2
"Wrong" character recognition (11/4/93)....3
vGesturesAllowed vs. vStrokesAllowed (11/4/93) (Obsoleted, Documented in Newton Programmer's Guide 1.0, on page 6-4/5)....3
Self-made Recognizers (11/11/93)....3
LookupWordInDictionary (11/11/93) (obsoleted by NPG1.0)....4
Custom Dictionaries and Resets (11/11/93)....4
GetDictionaryData and SetDictionaryData (11/11/93) (Obsoleted by NPG 1.0 page 6-26)....4
32-Character Limit with AddWordToDictionary, LookupWord, LookupWordInDictionary (12/22/93)....5
Deferred recognition? (4/22/94)....5
Adding Words to Custom Dictionaries, Limits and Errors (6/3/94)...5
Custom Dictionaries and the Names Application (6/7/94)....6
Remove Words From the Personal Word List (6/7/94)....6
Capitalization Limit on AddWordToDictionary (6/7/94)....7
Sleep While Strokedone! (6/7/94)....7
Don't Use vMathAllowed (6/9/94)....7
No Spaces, vNoSpaces, vSingleUnit (6/15/94) (moved from the Q&A on Views)....8
Don't Save the "unit" Parameter to Recognition Scripts. (7/5/94)......8
Handy Way to Include Dictionary Data in Applications. (7/19/94)...8
Double-Tap Details (8/26/94)....9
Tap & Stroke Times (10/17/94)....10
NEW
SetValue and Recognition viewFlags (4/19/95)....10
NEW
Modifying Global Recognition Settings (4/19/95)....10

Introduction

This document addresses Newton Written Input and Recognition 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.

Dictionary Selection (9/15/93)

Documented in Newton Programmer's Guide 1.0, on page 6-16.

Q: If the development environment indicates “Custom Dictionary” as an input choice, does it mean that Newton will use both its own dictionary and a custom dictionary?

A: When you set the vCustomDictionary flag, you must provide a slot called dictionaries in your application view. This slot contains an array of dictionary IDs. Each element in the array can specify a custom dictionary of your own or one of the system dictionaries.

Q: How does one specify that the system is to use only a certain dictionary or set of dictionaries?

A: System dictionaries can be turned on and off with the appropriate flags in the viewFlags slot. If you wish to use your own dictionary, turn on vCustomDictionaries and add your dictionaries to an array in the dictionaries slot. If none of the system dictionary flags are on, only the dictionaries in the array will be used.

Create Custom Dictionaries? (9/15/93)

Q: How would one create a custom dictionary on a PCMCIA card?

A: The current Newton ToolKit cannot build custom dictionaries at compile time. Stay tuned for more information concerning this

Control of Dragging and Scrubbing (10/20/93) (obsoleted by Newton Programmers Guide 1.0, page 6-5)

Q: How do I specify which gestures are allowed in a text field?

A: The vGesturesAllowed bit in the field's viewFlags slot enables and disables the dragging and scrubbing of text, as well as enabling and disabling highlighting, taps, carets, and line gestures.

Sloppy Clicking (10/20/93)

Q: If the end user clicks in the wrong view, this wrong view handles the click instead of the frontmost view. What is going on?

A: This is actually a feature of the view system, called sloppy-clicking! When you click on the screen and there’s no view under the pen that accepts clicks, the view system tries to locate a view near the click and send the message there instead. Because this only happens when no view under where the user taps accepts clicks, you can turn it off by allowing clicks in your background window. This feature is disabled entirely in any view that has no viewClickScript method.

"Wrong" character recognition (11/4/93)

Q: If I try hard enough, I can get views that should only recognize letters (vLettersAllowed+vClickable) to recognize numeric characters. Why is this?

A: This happens because the symbols dictionary is always enabled for any view which can accept text recognition. The symbols dictionary includes all alphabetic characters, along with numbers and some punctuation. This means you can write virtually anything in any view, if you try hard enough. The reason you have to "try hard" to make this happen is that recognition is weighted towards the appropriate characters (as defined by the viewFlags set for the view) so usually the appropriate match results. But if the input stroke is vague enough, a number might be returned as an alphabetic character.

Q: On the other hand, vClickable+vNumbersAllowed seems to always turn any input into a numeric value, which is what I would expect.

A: It does, but only because single digit numbers have a much higher probability for this view than single letters. However, you can get a letter recognized here if you try hard enough. Try letters like "w" which do not look much like numbers.

vGesturesAllowed vs. vStrokesAllowed (11/4/93) (Obsoleted, Documented in Newton Programmer's Guide 1.0, on page 6-4/5)

Q: What's the difference between the vGesturesAllowed and vStrokesAllowed viewFlags?

A: You can use either flag to invoke a method that provides application-specific handling of gestures, but vGesturesAllowed supplies the system-defined gestures tap, double-tap, highlight and scrub, while vStrokesAllowed does not provide any behavior that you don't implement yourself.

Self-made Recognizers (11/11/93)

Q: I would like to build recognizers for gestures and objects others than those built into the MessagePad.

A: Currently there's no support to add recognizers using the Newton ToolKit. Stay tuned for more information concerning this.

LookupWordInDictionary (11/11/93) (obsoleted by NPG1.0)

Q: Can you tell me what the undocumented function LookUpWordInDictionary does?

A: It's used for managing custom dictionaries. It will return TRUE if the passed word is in the dictionary, and NIL otherwise. The first paramater is the dictID (returned from NewDictionary) the second is the word (a string).

Custom Dictionaries and Resets (11/11/93)

Q: Is there a way to make custom dictionaries pervasive across resets? At the moment, when user resets, the dictionaries are lost. For example, if there were a system message that would be sent to applications upon startup after reset, maybe our application could trap that and reload the custom dictionary. Is there such a system message?

A: Your application's installScript will be called again after a reset, so your installScript could check to see if the dictionary existed and could recreate it. This is a good place to check anyway, since it will also be called when you insert a card with your application on it, or when you install the application for the first time.

GetDictionaryData and SetDictionaryData (11/11/93) (Obsoleted by NPG 1.0 page 6-26)

There are a pair of functions, GetDictionaryData and SetDictionaryData that will get a binary object that represents the dictionary. This binary object can be saved to a soup. This is how the system saves the user dictionary (in the system soup). Here's a quick sample (from the Inspector)

```
d := NewDictionary('custom)
#320      200
AddWordToDictionary(d, "Foobar")
#1A       TRUE
AddWordToDictionary(d, "baz")
#1A       TRUE
data := GetDictionaryData(d)
#44102A1  <dictdata, length 20>
DeleteWordFromDictionary(d, "baz")
#1A       TRUE
LookupWordInDictionary(d, "baz")
#2        NIL
SetDictionaryData(d, data)
#2        NIL
LookupWordInDictionary(d, "baz")
#1A       TRUE
```

32-Character Limit with AddWordToDictionary, LookupWord, LookupWordInDictionary (12/22/93)

Q: Very long words do not seem to be recognized by my Newton. What could be the problem?

A: AddWordToDictionary, LookupWord and LookupWordInDictionary all enforce a 32-character limit on the size of the input string. This limitation might disappear in future system releases, but currently you should not assume that strings bigger than 32 characters will be handled by these functions.

Deferred recognition? (4/22/94)

Q: How do I access the deferred recognition in the MP100 and MP110?

A: It's all automatic. Deferred recognition only works in clEditViews, and is generally only useful when the view uses the vAnythingAllowed recognition flag.

The user must double-tap selected "ink" children of the clEditView to initiate deferred recognition. The deferred recognition uses the current user-specified recognition settings. The recognized text is added to the edit view as if the user had just written it. That is, a new clParagraphView child is added, or the recognized text is appended to a nearby clParagraphView. After the recognized text has been added, the original clPolygonView with the "ink" is removed as if the user scrubbed out the ink.

If you have viewAddChildScripts and viewDropChildScripts, they will be called with the recognized text and old ink views. Words added to nearby paragraphs cause that paragraph's viewChangedScript to be called with the 'text and sometimes 'viewBounds slots being updated. These are currently the only hooks into the deferred recognition system.

Adding Words to Custom Dictionaries, Limits and Errors (6/3/94)

Q: There seems to be a limitation on adding more than about 1,000 words to the personal word list using the AddtoUserDictionary function.

A: There are limits on adding words to any custom dictionary. It is possible to run out of system memory for dictionaries and if you run out of that memory, bad things can happen. Do not rely on a specific number as the limit for the maximum number of words in the dictionary.

Q: Using AddtoUserDictionary to add more than about 1,000 words does not give an error or any indication that there is a malfunction, but the words just do not appear in the personal word list. Are you aware of this problem?

A: The return value from AddToUserDictionary tells you if the add was successful. Note that there are two reasons why add might return nil: the word might be already in the dictionary; or you might have hit a resource limitation.

Custom Dictionaries and the Names Application (6/7/94)

Q: How can I make the built-in Names application use my custom dictionary?

A: The built-in Names application does not support the use of custom dictionaries. In your own application you can set the vCustomDictionaries flag for any view that is to use dictionaries of type 'custom for word recognition.

Q: Is there another way to make the Names app recognize new words?

A: Currently, the only supported way to cause the Names application to recognize new words is to add them to the user dictionary; however, we strongly recommend that you do not add entries to the user dictionary because each additional entry in this dictionary reduces the amount of RAM available to the user. For more information, see the section "About the User Dictionary" on page 6-11 of Newton Programmer's Guide 1.0.

Q: Why doesn't the vNameField flag cause my application to recognize names in the built-in Names application?

A: The built-in Names application stores its information as entries in the cardFile soup. The vNameField flag specifies the use of built in system dictionaries for names that are common in the user's current locale bundle; it does not specify use of the cardfile soup. In fact, you cannot use the cardfile soup as the basis for recognition of written input.

Remove Words From the Personal Word List (6/7/94)

Q: Is there a function to remove words from the Personal Word List. If there is not, is there a workaround to remove words from that list?

A: The user dictionary is a RAM- based dictionary that you can modify. The following code will remove a word:

```
          DeleteWordFromDictionary(kUserDictionary, "myWord");
```

Note that you should use the function AddToUserDictionary("myWord") to add the word, rather than AddWordToDictionary(kUserDictionary, "myWord"). The former does more comprehensive checking to make sure the words are "safe" to add.

Note that the DeleteWordFromDictionary function does not necessarily make permanent changes in the system soup so that the word deletion is persistent. You must use the SaveUserDictionary function to save changes to the system soup.

Please also note that the constant kUserDictionary was not available in early versions of NTK. You can add it as follows:

```
          constant kUserDictionary := 31;
```

Capitalization Limit on AddWordToDictionary (6/7/94)

Q: My application sometimes crashes my MessagePad while my application is manipulating custom dictionaries. Sometimes, the crash occurs shortly afterward I have manipulated the custom dictionaries. Why?

A: One possibility is that you've added a word that begins with a capital letter to a RAM -based dictionary using AddWordToDictionary. Because of the current implementation of the recognizers, words added to a RAM custom dictionary must not begin with capital letters.

Note that the function AddWordToUserDictionary does not have this limitation. (It updates other internal data structures not present on custom dictionaries to indicate whether a word should be capitalized.)

If what you want to do is ensure that the recognized words become capitalized in the view, you can use the viewFlags flag called flag vCapsRequired to tell the recognition system to report the capitalized form of the word after recognition takes place.

Sleep While Strokedone! (6/7/94)

Q: Can the StrokeDone routine make strange things happen to my Newton? It seems to cause power faults or battery problems sometimes.

A: If you write loops similar to "while not StrokeDone(unit) do nil;", the processor will be in a tight loop, which uses the battery much more heavily than usual. It makes sense to provide a sleep statement inside the loop, with, for instance, a value of 1 or 10 ticks(1 tick is 1/60 of a second) which should be enough. For instance, use Sleep(1) in your StrokeDone loop. A sleep for a time of

1/60 of a second each time in the loop will cut down the power consumption significantly without sacrificing responsiveness.

Don't Use vMathAllowed (6/9/94)

Q: I have a particular view where, as the user writes into it, recognition gets slower and slower and eventually I get the Cancel/Restart dialog. There's no problem when using the keyboard. What's wrong?

A: There is a known problem with the recognizer used with vMathAllowed in the 1.05 System Update, and possibly in other systems. Do not use vMathAllowed.

The flag adds recognition for a few symbols not supported by vNumbersAllowed, namely (, +, -, x, /, and =. (Coincidently, the last 6 symbols in the "Letter Styles" list in the preferences roll.) If you need to add recognition for these symbols, create a small custom dictionary and use it.

No Spaces, vNoSpaces, vSingleUnit (6/15/94) (moved from the Q&A on Views)

Q: I have a protoLabelInputLine that accepts numbers only. I would like to prevent the entry of any spaces into this field. I tried to use vNoSpaces, but the field still accepts spaces? What should I do?

A: You have to set the textFlags field to vNoSpaces, and make sure that the vSingleUnit entryflag is set.

Don't Save the "unit" Parameter to Recognition Scripts. (7/5/94)

Q: Because I want to do a fair amount of processing, I'm trying to get information about a stroke unit in my application base view's viewIdleScript rather than in the tapped view's `viewClickScript`. When I use the unit paramater later, I get Exception |evt.ex.abt.bus|. What could be wrong?

A: The "unit" paramater that is passed to the various input-related scripts is really just an offset into a table used by recognition, and is only valid during the recognition process. Once the various scripts have completed and the user interaction is complete, the recognition system will reclaim the memory used.

If your application later tries to do something with the unit, it will follow a bad memory pointer, which generally causes bus errors. Because the reclaimed data is unspecified, there's some small chance that significant data could be destroyed if you do this.

You must collect all the user-input related data during the scripts provided. However, you don't necessarily have to do all the processing at that point. For example, you could call `GetPointsArray` during the `viewClickScript` and use the result later in a `viewIdleScript`.

Handy Way to Include Dictionary Data in Applications. (7/19/94)

Q: I need to create a custom dictionary with a few words. Doing it using a loop and `AddWordToDictionary` each time my application runs seems slow. Is there a better way?

A: There is a better way. If the dictionary you want to create does not change, you can create the dictionary once and then call `GetDictionaryData` to get a binary object that represents that dictionary. To restore the binary object later, call `SetDictionaryData` on a newly created custom dictionary.

Note that there will soon be some tools for creating dictionary parts. These are ROM- (or package-) based dictionaries and will not take up RAM like the custom dictionary solution.

In the meantime, here is a handy function for getting some NewtonScript source that will create a given binary object (and an example using the dictionary data.)

```
// Execute in the NTK Inspector
HexDump := func(o)
begin
   constant a:= "0123456789ABCDEF";
   write("SetClass(SetLength(\"\\u");
   for i := 0 to length(o)-1 do
      begin
         write(a[BAND(ExtractByte(o, i)>>4, 15)]);
         write(a[BAND(ExtractByte(o, i), 15)]);
      end;
   write("\\u\", ");
   write(length(o));  write("), '");
   write(classof(o));
   write(");\n");
end;

d := NewDictionary('custom);
AddWordToDictionary(d,"foo");
AddWordToDictionary(d,"bar");
AddWordToDictionary(d,"baz");

call HexDump with (GetDictionaryData(d));

// produces this code:
// SetClass(SetLength("\u610F6246610072707A3066006F006F30\u", 16),
'dictdata);

// In your package you can do
DefConst('kMyDictBinary,
SetClass(SetLength("\u610F6246610072707A3066006F006F30\u", 16),
'dictdata));

// Then to create the dictionary at run time, do:
d := NewDictionary('custom);
SetDictionaryData(d, kMyDictBinary);
// Don't forget to dispose the dictionary d when you are done with it.
```

Double-Tap Details (8/26/94)

Q: What does the system use to determine if a tap is the second part of a double tap? Can I use the view method IsDoubleClick?

A: In general, you should watch for the `aeDoubleTap` gesture in the `viewGestureScript` to reliably handle double taps. However, in the interest of documenting the *current* behavior (which is subject to change) here is some useful information.

A second tap is considered part of a double tap if it is close enough in time and distance to the first tap. The distance criteria in the MessagePad is 6 pixels. A variable number of ticks is used for the timeout. The time criteria is controlled by the global userConfiguration.timeoutCursiveOption. This defaults to 45 ticks, but can be set by the Transform My Handwriting slider in the Handwriting Style entry in the preferences roll. It ranges between 15 and 60 ticks. The gesture recognizer measures time between actual pen events, and will be reliable even if the main NewtonScript thread is busy. The distance between taps is measured based on the midpoint of the start and end points of the stroke. Note that a stroke longer than 6 pixels will not currently be recognized as tap (or as the second tap).

The view method IsDoubleClick should be not be used. It is undocumented, and behaves differently from the gesture recognizer. IsDoubleClick tests only the time between calls to that function; it does not actually test user input at all. It uses the global userConfiguration.doubleTime for its timeout criteria, which happens to be 30 ticks and cannot be set by any user interface in the MessagePad.

Tap & Stroke Times (10/17/94)

Q: If I call `ticks` in a `viewClickScript`, I get the time that the script was executed rather than the time that the pen went down. Is there a way to get an accurate time for the (beginning of the) user input?

A: You can use the previously undocumented functions `GetUnitStartTime` and `GetUnitEndTime`. Both functions take a stroke unit as a parameter and return the time of the start or end of the stroke in ticks. Do not attempt to use `GetUnitEndTime` until the input has finished (`StrokeDone` returns TRUE), as unpredictable behavior will result (usually a bus error.) Also, as with the other recognition-related functions, care must be taken to avoid calling these functions with a stale stroke unit. See the Q&A entry "Don't Save the 'unit' Parameter to Recognition Scripts, (7/5/94), above.

NEW: SetValue and Recognition viewFlags (4/19/95)

Q: Why can't I simply use `SetValue(myView, 'viewFlags, newVal)` to change recognition settings for a particular view?

A: You can modify your view's `viewFlags` recognition flags at run time using `SetValue`. Then you must call the global function `ReadDomainOptions` to tell the recognition system to notice the change.

`ReadDomainOptions`, which takes no arguments, tells the recognition system to forget any hints it may have stored, forcing the recognition system to recheck `viewFlags` when necessary.

NEW: Modifying Global Recognition Settings (4/19/95)

Q: Why can't I simply change `userconfiguration` settings to change the global recognition settings?

A: You can modify the global recognition settings (used by clEditViews with the `vAnythingAllowed` flag) by changing the `userconfiguration` slots `doTextRecognition` or `doShapeRecognition` to `TRUE` or `NIL`. Then you must call the global function `ReadDomainOptions` to tell the recognition system to notice the change.

`ReadDomainOptions`, which takes no arguments, tells the recognition system to forget any hints it may have stored, forcing the recognition system to recheck `viewFlags` when necessary.

As always, you should use the function `SetUserConfig` to modify any user configuration settings.

The resulting code might look like this:

```
call kSetUserConfigFunc with ('doTextRecognition, TRUE); // or NIL
ReadDomainOptions();
```

End of DTSQA