Newton® Technology
JOURNAL

Volume III, Number 1 *January 1997*

Inside This Issue

New Products

Introducing MessagePad 2000 and eMate 300

by Maurice Sharp and Bob Ebert, Apple Computer, Inc.

New Product Overview

In the fall of this year the Newton Systems Group, part of the Information Appliance Division of Apple Computer, Inc., will announce two new products based on the Newton OS. The new products, which will begin shipping in the first half of 1997, are the MessagePad 2000 and the eMate 300. This article provides an introduction to the hardware, OS enhancements, and new applications that are part of these products.

MessagePad 2000

The MessagePad 2000 is a cost-effective, powerful, easy-to-use hand-held computer than can provide targeted solutions in both the horizontal and vertical markets.

Based on the latest StrongARM RISC processor, the new device runs the Newton OS up to ten times faster than previous devices. It further adds to the power of Newton OS 2.0 by providing sound in to the existing sound out capability, a grayscale display, IrDA interface support, auto docking, and improved support for text editing. There are also two PC Card slots, enabling solutions that require both a memory card and a communications card.

continued on page 3

New Technology

Introducing Works for Newton

by Henry Cate, Apple Computer, Inc.

A new application framework is included in the new products from Apple. The framework, called Works, is intended to provide a simple yet powerful shell for productivity applications, much like desktop products with similar names.

Four applications will be initially available in the Works framework: a word processor, a drawing tool, a spreadsheet, and a calculator with graphing capabilities. This article will give you the necessary background to begin developing additional applications for the Works framework.

Works Goals

Simple Environment

An important requirement for the eMate 300 product was a simple environment where students could perform the kinds of tasks that currently require a desktop computer. The Works framework addresses this need. It uses a document metaphor that clearly separates individual projects.

When the eMate 300 device is set to the simplified classroom mode, the title bar and filing interface are hidden to prevent work from appearing to be lost. Works keeps the user data separate when in the multi-user mode. Works is also tightly integrated with the new classroom connection server.

Unlike desktop productivity applications, Works on the Newton platform is extensible through the stationery mechanism. Third parties

continued on page 7

Newton Technology JOURNAL

Volume III, Number 1 *January 1997*

Published by Apple Computer, Inc.

Jennifer Dunvan • *Managing Editor*

Gerry Kane • *Coordinating Editor, Technical Content*

Gabriel Acosta-Lopez • *Coordinating Editor, DTS and Training Content*

Technical Peer Review Board

J. Christopher Bell, Bob Ebert, David Fedor, Ryan Robertson, Jim Schram, Maurice Sharp, Bruce Thompson

Contributors

Bob Ebert, Maurice Sharp, Jim Schram, David Fedor, J. Christopher Bell, Bruce Thompson, Henry Cate, Vernon Huang

Produced by Xplain Corporation

Neil Ticktin • *Publisher*

Jessica Courtney • *Production Editor*

InfoGraphix • *Design/Production*

Editor's Note

Letter From the Editor

by Jennifer Dunvan

HAPPY NEWTON YEAR!!!

Whew! 1996 has been a landmark year for the Newton Platform, and we've taken a giant step forward, both as a technology and as a developer community. Hopefully most of you have had the opportunity by now to test the new Newton devices - the Newton MessagePad 2000 and the Apple eMate 300 — at the Newton Technology Conference last November or at a seed site. Both models were quite the hit at Comdex in Las Vegas!

We had a record number of attendees for the 1996 Newton Technology Conference, and according to preliminary reports, the feedback we've received is overwhelmingly positive. It seems that you appreciated the shorter conference length (2.5 days plus a day of lab time), and the General Technology session showcasing an array of technical talks was a hit. The "New to Newton" track was a sell-out, and the Business track allowed small business owners to hear about both marketing and technical information. Of course, the highlight of the conference was who else but our own talent extravaganza, the DTS Players ("Read the charges: $249 for a skit or $5 for a song!") If you have more feedback about the conference, please let me know at <dunvan@newton.apple.com>, and we'll be sure to consider it when planning for this year's conference.

During Comdex, both the eMate 300 and MessagePad 2000 took Byte's Best of Comdex Finalist honors and were demonstrated to thousands of people. Current MessagePad owners were blown away by the sheer power of the MessagePad 2000. In addition to speed, other popular features were the extra PC slot and grayscale display. And nearly everyone who walked through the Apple or Newton booths and caught a glimpse of the Apple eMate 300 had to stop and take a closer look at the unique and interesting design. Typical observations ranged from, "It's cute!" and "It looks like the front of a car!" to "Can I get one in [fill in the color]?" and "My kid would love this!" Cameras flashed, people "ooh"ed and "ahh"ed, and nearly everyone asked where they could get one for themselves, especially college students, parents, and mobile business people. Show-goers seemed most impressed by the eMate 300's durability, projected price, and extraordinary battery life.

In this first Newton Technology Journal issue of 1997, we cover everything you need to know to port your applications to the new hardware, and we've even boiled it down to a convenient list in the "Updating Your Application" checklist on page 23. The article "New Product Overview" offers you quick access to an understanding of the new features of both products. You'll find in-depth information on several topics, with "MessagePad 2000 Screen Size and Gray-Scale Graphics" and "Introducing Works for Newton". Don't miss part 1 of the excellent article "Data Structures — Storing and Retrieving".

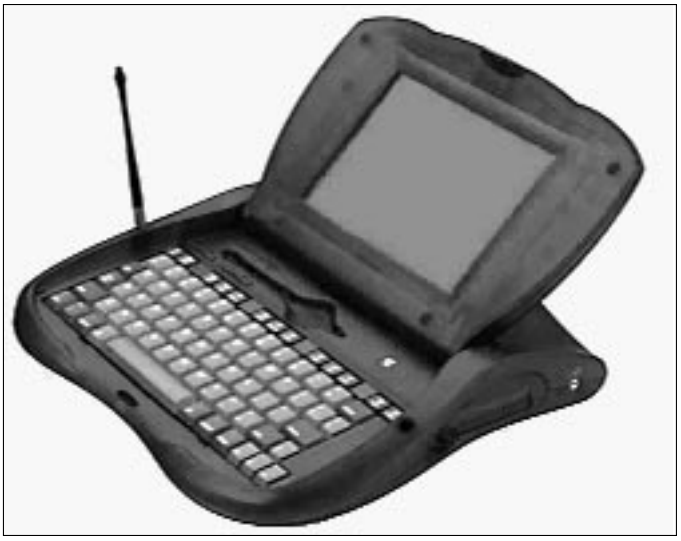
It is an exciting time here in the Newton Systems Group while we prepare for the release of the MessagePad 2000 and the Apple eMate 300. As Newton developers, you are the future of these products. Thank you for your continued support, encouragement, and firm belief in the quality and future of Newton technology. From all of us in the Newton Systems Group, we wish you outstanding success and a great year!

Jen Dunvan
<dunvan@newton.apple.com>

continued from page 1

Introducing MessagePad 2000 and eMate 300

The power and features of MessagePad 2000 will appeal to new types of customers in addition to existing fans of Newton products. It will also allow developers to create whole new classes of solutions. Between the Apple product bundles and these new solutions, we can take the Newton platform to the next stage of growth.



eMate 300

The eMate 300 is a rugged, low-cost computer with a large grayscale screen and embedded keyboard and is intended for use in the education market.

The education market is a new one for the Newton OS. The US education system alone consists of one hundred thousand schools with 48 million students, 2.8 million teachers, and nearly a million administrators. The US government in Goals 2000 has set a student to computer ratio for the classroom of three to one. To achieve this, nearly 12 million computers will need to be purchased for schools in the next four years.

At less than half the price of a low end desktop or one third the price of a laptop, the eMate 300 makes these goals more achievable because it offers nearly all the productivity of a traditional computer. Long battery life, durable and lightweight hardware, and connectivity to traditional computers set the expectation that the eMate 300 will be extremely popular with students, teachers, and administrators.

Common Features

The MessagePad 2000 and the eMate 300 are both built around the same basic hardware design and OS. Here are the features common to both products.

Based on Newton OS 2.0

The new operating system is an extension of the 2.0 release of the Newton OS. Applications written for existing Newton products will work on both the eMate 300 and MessagePad 2000 devices. Data can be freely exchanged between the new devices and existing products.

Selected enhancements have been added to the OS to improve support for the new hardware. Support for the keyboard, larger grayscale display, and additional communications hardware is fully integrated. A basic view class tuned specifically for word processing has been added, and a new set of applications and a framework are also provided.

New Silicon

In order to make it easier for Apple and its licensees to create devices based on Newton technology, Cirrus Logic, Inc. and Apple Computer, Corp. worked together to produce a core Newton Technology chipset. The result is the Apple Newton PDA Chipset. There are four basic chips in the set: a Digital System Controller (DSC), an Analog System Controller (ASC), an IR Controller (IRC) and a PC Card Controller. Any Newton device will add ROM, RAM and I/O (ports, screen, speaker, PCMCIA, etc.).

The eMate 300 and The MessagePad 2000 are the first Newton devices to be introduced using this new chipset. Although the units differ radically from each other in physical appearance, thanks to the chipset their schematics are nearly identical.

Grayscale

Both eMate 300 and the MessagePad 2000 feature half-VGA sized four-bit grayscale displays that will show sixteen shades of gray. The Newton OS has been extended to support both grayscale and color while still maintaining full backward compatibility. All existing calls now support both monochrome and color objects, and APIs have been added to access the new gray/color features.

Color values are specified using an RGB triplet where the values are either five or eight bits, for 16 or 24-bit color. Color can be used for patterns, drawing and text. The system can display color PICTs and bitmaps that have one, two, four or eight bit depth, and there is support for antialiasing monochrome bitmaps.

Newton Toolkit and Newton Press will be updated to seamlessly support grayscale and color images as well as the larger screen sizes.

Keyboard

Extensive enhancements have been made to improve support for keyboards. Menus and buttons allow key equivalents and keyboard navigation, and standard key combinations are handled by the system.

Event handlers can be added for command keys, or for any key combination, and like other NewtonScript structures, the key definitions can use inheritance chains to reduce memory requirements. Menus and command keys can use the same definition frames to further simplify design and coding.

In the new OS, any view may accept keyboard input, and views are now notified when they gain and lose keyboard focus. All views also get a chance to preprocess any key input, for fully customizable interfaces.

Sound

The updated Newton OS features enhanced sound output and has added sound input. Most of the new functionality is provided to developers by a new proto, protoSoundChannel, which makes it possible to: specify a direction (input or output); queue up sounds for playback or recording; and

pause or stop recording or playing. It is also possible to specify the channel for the sound: line-in, or the MessagePad 2000's embedded microphone; or speaker, line-out, or the eMate 300's headphone jack. Another new proto is (protoSoundFrame) can be used to access sound data, such as finding the playing time of a sound, and there are new user interface elements for recording sounds.

There is support for playback of up to four simultaneous sounds. The OS will also support recording or playing 8-bit (Mac standard) or 16-bit linear encoded (CD quality) sounds.

IrDA

Existing Newton devices use the Sharp ASK IR communications protocol. The new devices use the Infrared Data Association's (IrDA) standard for infrared communication. This is a cross-platform standard supported on both Windows-based and MacOS computers, as well as other devices such as IR printers. The new devices also support ASK for backward compatibility.

The IrDA implementation in the Newton OS includes the link access protocol specification (IrLAP), the link management multiplexer protocol specification (IrLMP), and the hardware serial infrared interface specification (SIR). SIR includes communications rates up to 115 kbps, and also supports both primary and secondary roles (that is, the Newton device can be either a client or a server.)

IrDA is fully integrated into the Newton endpoint communication paradigm. Existing code can be easily modified to take advantage of IrDA. Unlike the ASK protocol, IrDA simulates a full duplex connection.

Text Engine

A new view class has been added to greatly improve performance with large blocks of text. Text is displayed in a paged, WYSIWYG manner that can scroll very quickly. Text blocks up to the limit of available user storage can be viewed and edited with no delays.

The text is fully styled, including multiple font faces and sizes in various styles. Pictures of any size, in the form of Newton shapes, can also be embedded in text.

Rulers allow formatting on a paragraph-by-paragraph basis. Margins, line spacing, indenting, and justifications are all supported, as are left, right, center, and decimal tabs.

Interfaces provide developers with the ability to read and manipulate any text attributes. It's also easy to convert between text positions and screen locations useful for determining when a word has been tapped. Editing commands such as cut, copy, paste, and clear are built in, and there are calls for search and replace that work whether the text is displayed on screen or stored in a soup.

The text engine is virtually a complete word processor, and can be used as a base class for other applications such as HTML viewers, hypertext browsers, email editors, reference engines, and so on.

Spell Checker

Using the built-in dictionaries, the new OS now features the ability to check spellings and offer alternatives for misspelled words. APIs allow developers to integrate spell checking with their own applications.

NewtonWorks

A new extensible application framework has been added to facilitate classroom use. Designed for the eMate 300 but also available on the MessagePad 2000 platform, the framework, dubbed "NewtonWorks" or simply "Works," is a shell for basic productivity applications such as word processing, drawing, math and graphing, and spreadsheets.

Works uses a document metaphor. Each paper, drawing, graph, spreadsheet, or whatever appears as a single self-contained document. Works provides the framework for viewing and editing each document, and for switching between documents.

Developers can add new types of documents to the Works shell by registering stationery, much like extending the Notes application on existing Newton devices. Works-specific features have been added on top of the stationery APIs to allow for application specific help and preferences, custom data in title slips, Works-wide searches, and custom tools. Only a single viewer/editor needs to be provided, and multiple print or routing formats may also be added.

Like the Notes application, Works applications must be designed to store document data in a single soup entry. The Works framework manages the data storage so that applications developers don't need to worry about soup design. The Works framework also provides both horizontal and vertical scrolling, title bars, and a status bar that can be extended with custom buttons and menus.

The Works framework also allows applications to customize each other, by providing a registry for tools. The built in Works applications all support various forms of third party extensions using this tools registry, and some allow additional extensions.

Word Processor

A standard word processor application is provided within the Works framework. In addition to providing a ruler, button and menu user interface to the Text Engine, the word processor features a spell checker and quick sketch editor.

A spell checker and quick sketch tool are integrated with the word processor as standard Works tools, and other tools can be added using the same facility.

Drawing

Also provided within the Works framework is a draw-style graphics application. The application provides the usual range of shape creation tools in a tool bar: lines, squares and rectangles, circles and ovals, curves, and polygons, along with a shape recognizer tool. Each shape may be drawn with a variety of line styles or fill patterns, using the grayscale capabilities of the devices.

Adding to these standard tools, the drawing stationery also provides a set of pictures that can be "stamped" onto a drawing. The set of stamps, and the available fill patterns can easily be extended through draw application interfaces.

The Drawing editor supports the standard Works feature of an extensible Tools menu, and APIs are also provided to allow developers to extend the tool bar with custom drawing tools, and provide custom editing of the shapes created.

Because the contents of a drawing are based on standard NewtonScript shapes, drawings can be easily integrated with other applications. Cutting and pasting between a drawing and a word processor document is a simple operation.

Calculations

Another Works drop-in is stationery that provides graphing calculator features. Users can enter and evaluate variables and equations using standard

mathematical notation, and display the resulting functions in a graph or a table view. Polar and parametric graphs are supported, and the interface makes it easy to zoom in to look at details or zoom out to get a big picture.

Teachers or other experts can easily create and distribute pages of pre-defined equations or constants, or templates for homework. Developers can further extend the set of built-in functions and constants to provide custom functions for specialized financial, statistics, engineering or science applications.

The math environment, equation editor, and graph and table displays are provided as prototypes that can be used in other applications.

Newton Interconnect

A major limiting factor in obtaining a thinner profile for MessagePad devices has been the DIN9 port used for serial and AppleTalk communications. In order to facilitate thinner devices, Apple has adopted a new standard connector for the Newton platform. The result is Newton Interconnect.

The connector is a JAE 26-pin connector which requires a JAE 26-pin custom plug. The height of the connector is much less than a DIN9. In addition to the serial/LocalTalk port, the pins allow access to power in/out, audio line-in and line-out and another serial input that has no LTC Line Driver. There is also an auto docking pin that tells the unit it has been connected.

MessagePad 2000 will come bundled with a DIN9-to-Interconnect adapter, and eMate 300 includes both ports, so current cabling does not need to be replaced.

Value Series Flash

Advances in technology have enabled much cheaper FLASH memory. This new technology, called "Value Series" FLASH allows RAM PC cards to be products for about half the cost of current cards. The Newton OS supports this type of memory. The eMate 300 hardware fully supports only the new type of card, though it will allow read only access to older (12V) memory cards. The MessagePad 2000 will read and write both (5V) Value Series and (12V) existing PC cards.

Serial Number Chip

Both eMate 300 and MessagePad 2000 come with an on board serial number chip. Each device will have a unique serial number that can be accessed through a secure NewtonScript API. This provides developers with the ability to easily register or copy protect their software, or track individual units.

MessagePad 2000

MessagePad 2000 is a major step forward in the evolution of the MessagePad product line. It is a cost-effective, powerful, easy-to-use hand-held computer than can provide targeted solutions in both the horizontal and vertical markets.

MessagePad 2000 Provides:

- The fastest hand-held computer experience!
- More memory for applications and capturing data.
- Long battery life.
- Access to networks, Internet and Intranet sites.
- PC (Mac and Windows) integration.
- A wide base of developers with powerful development tools.

MessagePad 2000 Markets

For the horizontal market, MessagePad 2000 is a cost-effective hand-held computer for highly mobile professionals. It combines a powerful processor, long battery life, instant on, general business applications, and connectivity to both the desktop and the Internet.

For vertical markets, MessagePad 2000 is the most extensible, mobile, and powerful hand-held computer for business enterprises seeking a cost effective solution to improve productivity, reduce costs and enhance the overall quality of service by capturing information at the point-of-activity.

Apple is focusing on the Home Health Care vertical market. We are working with SIs, VARs, and ISVs/IHVs to ensure that a whole product solution is available.

MessagePad 2000 Hardware

The processor is a DEC StrongARM SA-110, which is a 32 bit RISC processor, running at 161.9MHz. The processor operates at 1.7 volts. The display is a 480x320 non-glare transflective grayscale LCD display with an electroluminescent backlight, covered by a touch-sensitive tablet which is optimized for use with the pen. Part of the screen is used for a software button bar.

The unit has an 8MB ROM on a replaceable card, 1 MB of DRAM for system use, and 4 MB of Flash RAM for programs and data. There are two Type II PC card slots that support 3.3 or 5.5 volt operations at up to 500mA. There is also an internal serial connector that has access to three serial lines. The case has punchouts for an RJ-11 modem jack next to the interconnect port and another punchout for a ribbon cable along the cover hinge. There is also a small amount of room inside the unit for a modem or other serial card.

The unit has a front-mounted microphone, speaker, and an AC adapter and Newton Interconnect port along the top. The internal serial connector can be routed to the Interconnect port. Also along the top is the infrared transceiver. There will be an optional keyboard that plugs into the interconnect port.

The unit can run on 4 AA batteries or an optional NiMH rechargeable battery pack. Battery life is from 20 to 62 continuous hours of use depending on backlight and PC card power requirements. The NiMH batteries fast charge in about an hour.

MessagePad 2000 measures 8.25"L x 4.7"W x 1.25"H (210.3mm x 118.7mm x 27.5mm) and weighs only 1.8 lbs (0.82 kg).

Additional MessagePad 2000 Applications

Available with MessagePad 2000 are a number of extra applications and services to provide a whole-product solution for our customers. Among these are:

- Newton Connection Utility for MacOS™ and Windows™.
- NewtonWorks Word Processor (keyboard required).
- Newton Internet Enabler 1.1.
- Web Browsing software.
- Internet mail client software.
- Spreadsheet software.

eMate 300

eMate 300 Market

The eMate 300 has been designed in cooperation with educators to meet the specific needs of students. It works as a companion to MacOS and Windows-based PCs in a Distributed Learning Environment.

A Distributed Learning Environment is one that extends the reach of learning, and of computing technology, beyond the classroom to libraries, labs, homes, and the outdoors. The eMate 300's rugged, portable design and long battery life make it well suited for these situations. It's ideal for taking notes in a classroom or library, gathering data in the field, or getting on-line anywhere.

The Information Appliance Division plans to introduce eMate 300 in a variety of areas. Building on Apple's strong presence in education, the first products will be sold directly into the K-12 school system. Consumer and University products are expected to follow.

eMate 300 Hardware

The eMate 300 uses a clamshell design which protects the case and keyboard when the unit is not in use. Thick plastics with reinforcing ribs, soft rounded corners, a steel chassis, and a shock mounted display provide protection against drops, twists and other abuses. Field replaceable components ensure minimal downtime if the inevitable occurs. The display, tablet, and backlight assembly, along with the keyboard and battery are all field replaceable, and the total weight with batteries is under four pounds.

The processor is an ARM 710a, which is a 32 bit RISC processor clocked at 25 MHz. The display is a half-VGA 480x320 transflective grayscale LCD display with an electroluminescent backlight. The display is covered by a touch sensitive tablet, optimized for use with the pen, and a 17mm sub-notebook keyboard is built in. The eMate 300 does not use screen space for a software button bar, instead using a groups of keys above the numbers on the keyboard as a button bar replacement. There are keys for Extras, scrolling and overview, undo, find, and assist, and new keys have been added for docking, beaming, and closing slips.

The unit contains an 8 MB ROM on a replaceable card, 1 MB of DRAM for system use, and 2 MB of FLASH memory for programs and data. There is a single standard Type II/III PCMCIA card slot that supports 5V cards drawing up to 500mA, and an internal RAM/FLASH upgrade slot. The device is capable of using existing (12V) FLASH RAM cards in a read-only mode.

The four AA cell NiMH rechargeable battery pack provides about 28 hours of high usage between recharges, and fast-charges in about an hour. The battery space is designed to allow for a larger four A cell battery pack, which approximately doubles the capacity. There is a power adapter plug on the side in addition to charging contacts on the base, and the Newton Interconnect port can power the unit and charge the battery. A LED mounted on the rear indicates battery condition.

The unit automatically powers off when the lid is closed, and powers on instantly when the lid is opened. There are also power and backlight control keys on the keyboard. Sliders above the keyboard control the display contrast and the speaker/headphone volume.

A built in infrared transceiver is included that is compatible with existing Newton protocols and also supports IrDA at 115Kbps. A Newton Interconnect 26 pin connector is available as an alternative to a standard DIN9 serial/LocalTalk port. The unit contains a speaker and a headphone jack, in addition to audio line-out and line-in connections in the Newton Interconnect port.

Simplified Interface for eMate 300

To simplify the Newton experience for students and ease configuration issues for teachers, a limited mode ("Simple" mode) is available on eMate 300. In Simple mode, the Newton filing interface is hidden, and the Extras drawer will show only a teacher-defined subset of the available applications. Third party applications can determine if the unit is in Simple or normal mode, and adjust their interfaces accordingly.

Multiple Users on eMate 300

To allow multiple classes to easily share hardware, the eMate 300 now supports multiple users. This feature is available in Simple mode only. When enabled, the unit requests a name and optionally a password when the unit is powered on. Once signed on, the user only works with their data; other users' data is hidden.

Teachers can pre-define sets of users, or allow students to create their own accounts, or the unit can be configured to require a password. A special teacher password unlocks full access to the machine.

Applications must be updated to support multiple users. We recommend that applications store data for each user in a separate soup. Each application can find out the current user, and use the appropriate soup or otherwise adjust the visible data accordingly. Applications are notified when the user is changed or an account is deleted.

eMate 300 Classroom Connection

To ease integration into a classroom environment, Apple is creating a specialized connection server intended to allow a classroom full of students to quickly retrieve their work at the beginning of class and move it back to the server at the end.

Unlike existing connection products, the server will support multiple connections at once. The user will connect via serial, IR, or LocalTalk and either upload the contents of the eMate 300 device to the server, or download data on a document-by-document basis.

On the server, each Works document will appear as a separate streamed object file. Developers may use desktop interfaces from Apple (part of the Desktop Integration Libraries or DILs) to read and write files in this format on the server. For example, XTND translators can be created to integrate these files with a variety of desktop applications. Translators for the built-in word processor and drawing applications will be included.

Data from Newton applications that are not integrated with Works will also be available on the desktop, in a slightly different file format. Each application soup will exist as a separate file on the desktop, and the same DILs interfaces can be used to read and write those files.

Conclusion

The two new Apple-labeled products are nothing short of amazing. Besides the incredible hand-held computing power and "gee whiz" appeal, they also deliver new capabilities for growth in markets where Newton devices are already present, and open up tremendous opportunities to expand into new markets like education and home health care. These new products in combination with the updated Newton OS will provide a solid foundation for future growth of the Newton platform.

Additional information about the Infrared Data Association can be found on-line at <http://irda.org/>.

NTJ

continued from page 1

Introducing Works for Newton

will be able to add tools to existing stationery, and register whole new types of stationery. New stationery should try as much as possible to maintain easy to understand interfaces and use established patterns for user interfaces such as command key equivalents and menus.

About Works

Like the existing Notes application, Works is written around the NewtApp framework for Newton applications. Those who have written stationery for the Notes application will find that writing for Works is very similar.

Works itself is similar to the Notes application in many ways. The headers, title slips, menus, status bar, and overview are all stock elements. There are some differences; however, for example Works has scroll bars. Because documents like papers or drawings are commonly larger than even the larger screen of the new devices, users need scroll bars to navigate their work. A find and replace feature is also new to the Works framework.

Using Works

Works is the default backdrop application on the eMate 300 platform, so it's always readily available. A common classroom scenario might have a student opening the eMate 300 to start work. A couple of keystrokes creates a new word processor document, and the student begins writing their paper.

When graphics need to be added, there are a couple of options. A "quick sketch" tool integrated with the word processor stationery allows a student to easily insert sketches. For more detailed drawings, a new drawing document is created and edited. (The student can also use the drawing stationery to create drawing documents.) When the drawing is done, the student copies the result to the clipboard, switches back to the paper, and pastes the drawing in. Common clipboard formats are used to aid in data exchange between Works applications and other applications in the system.

When the paper is complete, it can be printed from within Works and turned in. It could also be beamed to another machine, either to hand it in to the teacher or hand it off to another student for collaborative work.

If additional work is required to finish the document, it can be moved to a desktop computer via the classroom connection server. The document will appear on the desktop as a separate file, and translators will allow it to be opened in ClarisWorks and other popular desktop applications. Once on the desktop, additional finishing work such as adding color graphics or incorporating multimedia content can be done.

With the classroom connection server, documents can be moved off the eMate 300 unit when work is complete. Rarely will students in the classroom have more than a few documents at a time in Works, so the file management tasks in that scenario are minimal. Of course, Works will be used outside the classroom and on other Newton devices, and the filing user interface is available in those cases to help with document management.

Since the eMate 300 units will typically be more readily available than desktop machines, more students will be able to do individual work more quickly. Works is the vehicle for this type of productivity within the Newton OS.

Inside Works

This section gets into the technical details of extending Works. It provides a starting point for developers interested in creating Works applications.

When to use Works

Since Works is a NewtApp application, deciding to add stationery to Works is similar to deciding if you want to add stationery to Notes or some other NewtApp application. Additionally a good candidate for Works is any kind of document editor. If you can structure your stationery so it allows the student to view and modify some type of document, then you may want to use Works. This document can be any unit of electronic data, for example it might be a spreadsheet or a sheet of music. Also remember that though Works will be available to anyone running N2 or eMate 300, its primary focus is the education market. So consider the target market; will they often be using Works, or will they be more often in another NewtApplication?

Stationery with some extensions

To add an application to Works, the DataDef & ViewDef need all the methods and slots normally used in stationery, plus additional methods and slots. Note, Works is designed to only use one ViewDef, the 'default ViewDef. Other ViewDefs are allowed for routing.

To take advantage of the additional features Works provides, the DataDefs and ViewDef needs some additional slots and methods. In addition there are some APIs for registering and un-registering tools with a particular DataDef. We'll walk through some typical actions students would do with Works, and explore the APIs invoked.

New Document

When a student taps on the "New" button, Works checks to see what stationery is registered. To provide stationery to Works, the standard stationery APIs are used. These are the global functions `RegDataDef`, `UnregDataDef`, `RegisterViewDef`, and `UnregisterViewDef`. Check the Newton Programmer's Guide (NPG) - System 2.0 for general information about these functions. Note, the DataDef's superSymbol must be `'newtWorks`.

After a user selects a particular piece of stationery, the title slip comes up. You can put information in this slip. Works makes a conditional call to the DataDef's method `InfoBoxExtract`. This method is passed the target, a bounds frame, your ViewDef, and should return a shape. The shape can be anything. It might be a small sketch of a drawing, or the result of calling MakeShape on some text summary of the document. The shape will be appended to the bottom of other content that appears in the infoBox.

Preferences

One of the next things a student might do is tap on the information button and bring up the preferences. Additional preference information can be added by setting a `'prefs` slot in the DataDef. The frame in the `'prefs` slot is similar to a frame passed into `PopupMenu` as documented in the NPG 2.0. This frame must also contain a `'prefsTemplate`. This template contains the view template needed for the user to set the DataDef specific preferences. For example the

DataDef can add a "Special Prefs" choice to the information button which brings up the prefsTemplate containing document specific preferences.

When preferences is selected from the information slip, Works will set the slots `'target`, `'newtAppBase`, and `'viewDefView` appropriately in the `'prefsTemplate`, and then build the context. The template must read the appropriate information in the viewSetup method, and store the information in the `viewQuitScript` method. You can use `GetAppPreferences` to get and set your particular preferences.

When the global Works preferences change Works will conditionally call the ViewDef method `PrefsChange` with a preferences frame. Currently the frame has two Boolean slots, `'metricUnits` and `'internalStore`.

Searching and Finding

There are a number of methods available for assisting Works when a student does a search. There are two types of Find available in Works. The first is the standard Newton Find, a search which searches one or more applications. For example the student could search for a text phrase in both Works and Notes. The second type is a document find, specific to just Works, this allows the student to do things like find and replace in the current document.

When Works does a Newton find, it first uses the global function `FindStringInFrame` to see if the string is found in the entry. If `FindStringInFrame` does not find a match, then the DataDef's method `FindFn` is called to let the stationery do additional matching. The `FindFn` is passed the entry, the string to look for, and an offset. Currently the offset is always zero. If there is a match, `FindFn` should return `TRUE`, otherwise return `Nil`. Once Works finds a match, the DataDef's method `FindSoupExcerpt` is called. `FindSoupExcerpt` returns the text that will appear in the Find Overview. This method takes the same arguments as the `FindSoupExcerpt` documented in NPG 2.0.

If Works gets one match, or a particular entry is selected from the Find Overview, the ViewDef method `ShowFoundItem` is called. The method can update the view to show where the match is in the entry. Check the NPG 2.0 for information about what is passed to `ShowFoundItem`.

When a student does a Works Find, the ViewDef method `FindChange` is called. `FindChange` is passed two variables. The first is the action the student is doing, currently: `'find`, `'change`, or `'changeAll`. Depending on the action, the second variable will have information about the string to find, and possibly the string with which to do the replacing. The return value also depends on the action. FindChange is responsible for updating the view.

Scrolling

Often users are going to want to scroll, to see more of the document. If the set of methods below are defined, then Works will do the scrolling.

The ViewDef method `GetScrollableRect` returns a bounds frame if scrolling is to be done by Works. If `GetScrollableRect` returns a bounds frame, the scroll bars will be displayed. If `GetScrollableRect` returns `nil` then the scroll bars are not displayed, and no other scrolling methods need to be defined. `Nil` might be returned if scrolling is not allowed, or if the ViewDef will do its own scrolling.

A group of ViewDef scroll methods provide status information; they take no variables and just return data. `GetScrollValues` returns a frame of the current scroll values. The frame should have `'x` & `'y` slots with the current position as integer values. `GetTotalHeight` and `GetTotalWidth` each return an integer, corresponding respectively to the height and width of the view.

The second group of methods are implemented by the ViewDef to update the view in response to a scroll request. Two methods, `ViewScrollUpScript` and `ViewScrollDownScript`, need to update the image in response to an Up or Down arrow key. If these methods are not defined, then Works will do a default action of scrolling up or down one screen. The method `Scroll` is passed a frame with `'x` and `'y` slots. These are integer values which specify the number of pixels to scroll the image by. Note, the `Scroll` method must call the ViewDef's `UpdateAllScrollers` method, which we will discuss next.

The last scrolling method is `UpdateAllScrollers`. This ViewDef method is implemented by Works and is designed to be called by your code. It takes the view, and four Booleans. The Booleans in order are `TRUE`, if the total height changed, if the vertical scroller thumb needs to be updated, if the total width changed, and if the horizontal scroller thumb needs to be updated. If the width and/or height changed, then `UpdateAllScrollers` will call `GetTotalHeight` and `GetTotalWidth`, and recalculate its internal data structures. If the position of the thumb(s) changed, then `UpdateAllScrollers` will call `GetScrollValues`, and again recalculate its internal data structures.

Scrolling can be confusing, so we'll go into a little more depth on how the methods above work together. When a ViewDef is opened, or at any time that the total height or total width changes, the UpdateAllScrollers method should be called. It will call `GetTotalHeight` and `GetTotalWidth`.

`GetTotalHeight` and `GetTotalWidth` provide the total size of the object to be scrolled. The `GetScrollableRect` method provides a bounds frame with the size of the currently visible area. So for example let's say there is a 1000 by 1000 pixel area to be scrolled, and a 100 by 100 visible area. `GetTotalHeight` and `GetTotalWidth` would each return 1000, while `GetScrollableRect` would return `{left: 0, top: 0, right: 100, bottom: 100}`.

`GetScrollValues` controls the position of the scroll bar thumbs. The 'x & 'y values of the frame returned by `GetScrollValues` specify the scroller position in relation to the values provided by `GetTotalHeight` & `GetTotalWidth`. So with the example above, if `GetScrollValues` returned `{x: 500, y: 500}`, the scroll bar thumbs would be positioned exactly halfway along the scroll bar.

`Scroll` is the method which provides the actual scrolling functionality implemented by the ViewDef. After updating the image, `Scroll` should call the Works' method `UpdateAllScrollers`, which in turn will call the ViewDef's `GetScrollValues` method and correctly set the scroller position.

The Status Bar

To provide any custom buttons in the status bar, the ViewDef has two slots, `'statusLeftButtons` and `'statusRightButtons` which each hold an array of button frames. Check the NPG 2.0 for more information about button frames in a NewtApplication's status bar.

The ViewDef has a method, `UpdateStatusBar`, which is called when there is a change to the auxiliary button registry. This can occur if a package installs or removes an auxiliary button for Works. The `UpdateStatusBar` method should update the two status bar arrays.

There is also a newtAppBase method called `UpdateStatusBar` which can be called when there is a need to recreate the status bar. It will check for a currently active ViewDef, and call the active ViewDef's `UpdateStatusBar` method.

Providing Help

If the student taps on the information button, and then selects Help, the first thing Works will try is to call the optional ViewDef method, `DoHelp`. `DoHelp` can do anything special which needs to be done. If `DoHelp` returns `'loadHelp`, Works will also try to load the help book.

If `DoHelp` is not implemented, or if `DoHelp` returns `'loadHelp`, Works will use two ViewDef slots `'helpManual` and `'viewHelpTopic` to open a help book. The `'helpManual` slot should have a help book frame. Check the "Beyond Help" DTS sample for information about how to create a help book. The `'viewHelpTopic` can be set to specify what topic to open to when the help book appears.

Data Storage

The saving of the current entry is done by Works. Before saving the entry, the ViewDef method `SaveData` is called. There are two times `SaveData` will be called. First `SaveData` is called periodically by Works. Secondly if you want the entry to be saved call the method `StartFlush`, this will cause the method `EndFlush` to be called after a period of time. The method `EndFlush` will call the ViewDef's method `SaveData`.

The method `SaveData` is passed the current entry. It can update slots in the entry, and return a value indicating if the entry should be saved. `SaveData` should return `nil` if there is no reason to save the entry, or return the symbol `'NoRealChange` or `TRUE` to save the entry. If there have only been lightweight changes, `'NoRealChange` will cause the entry to be saved, but the modification time stamp will not be updated.

Using Tools in Works

Tools can be added for a specific type of stationery. A ViewDef has the responsibility for displaying the tools installed for that stationery type, and correctly calling the tool when it is invoked.

There are two Works methods, `RegNewtWorksTool` and `UnRegNewtWorksTool` to handle adding and removing tools from a type of stationery. The first is passed a tool symbol, and a tool frame. The frame is similar to the frames documented for `PopupMenus`, check the NPG 2.0 for more information. The toolsFrame has a slot, `dataTypeSymbol`, which specifies the stationery type it is registered to. For Paper and Drawing a second key part of the toolsFrame is `cmdFunc`, the method is passed a `viewDefView` (the main ViewDef view), and the `newtAppBase`. There are a few more slots needed in the tool frame; check the Works API documentation for more information.

These methods would typically be used in the install and remove scripts, and would be called like

```
GetRoot().newtWorks:RegNewtWorksTools (toolSym, toolsFrame);
GetRoot().newtWorks:UnRegNewtWorksTools(toolSym);
```

There are two additional Works methods. The first, `GetNewtWorksTools` which returns an array of the tools belonging to the specified stationery type. The second, `GetNewtWorksTool`, takes a tool symbol and returns the tool frame.

To be notified when tools are added or removed, add a `ToolsChanged` method to the viewDef. If the viewDef is the current view when there is a change in the tools register, the `ToolsChanged` method is passed the action and the tool symbol.

View Changes

When the view bounds change, for example if the horizontal scroller is no longer needed, or if the icon bar in N2 is moved from the side to the bottom, Works will call the ViewDef method `viewChangedScript`. This ViewDef method is also called anytime a SetValue is done on the view. Check the NPG 2.0 for more information.

Two help slots

There are two helpful slots in the Works base view. The first is `newtAppBase` which contains the Works base view. The second is `viewDefView` which contains the current ViewDef view. This can be nil if there is no currently active ViewDef, for example when Works is in overview mode.

Conclusion

Works is a built-in application framework that takes care of many basic programming tasks, allowing developers to focus on the core work of creating a document editor. Works' users are guaranteed a consistent means of creating and managing their work. Works is readily available, making it the framework of choice for document-centric applications, and especially those intended for the classroom.

For more information on Works

For more information on adding an editor to Works, check the reference document on the developer CD. A Works sample shows a barebones framework needed to add an editor to Works. This is a good starting place for your development. You can change the name, the symbol, and then plug in some functionality.

For more information on NewtApp & Stationery

A good place to start for learning NewtApp is the NewtApp overview article in the June 1996 Newton Technology Journal. The Newton Programmer's Guide has a chapter on NewtApp and a chapter on Stationery. Since adding to Works is done by creating stationery, the chapter on Stationery will be frequently used; however, the Stationery chapter builds on concepts in the NewtApp chapter so read the introduction of the NewtApp chapter. The sample "WhoOwesWhom" is found on the developer CD and shows how to add stationery to the Notepad. The sample "Cardfile Extensions" also shows some statio NTJ techniques in adding additional functionality to the Names app.

MessagePad 2000 Screen Size and Grayscale Graphics

by Bruce Thompson, Apple Computer, Inc.

An Introduction to the MessagePad 2000 Screen

The MessagePad 2000 platform provides a new screen for Newton applications. The new screen is one quarter VGA size (320x480 pixels) and provides 16-level grayscale. This new screen resolution provides exactly twice the screen real estate over the MessagePad 120 and 130. As a result, applications now have greater freedom in their choice of how to present information.

Along with the increased screen size comes grayscale. The MessagePad 2000 screen provides applications with a palette of 16 gray levels. Where images once had to be shown using dithered gray levels, the MessagePad 2000 screen provides real gray tones. In anticipation of possible future devices, the MessagePad 2000 screen is actually working with color values internally. The gray levels are simply a mapping from color values to gray.

The button bar has changed as well. Gone are the silk-screen buttons. Gone are the arrow buttons that don't quite make sense when the screen is rotated. Gone is the restriction to only rotate to a single portrait or landscape orientation. The MessagePad 2000 screen can now rotate to any of the four possible portrait or landscape orientations. The orientation that works best can be used. Don't like the button bar at the left of the screen in landscape? Now you can put it on the right side. With the demise of the silk-screen buttons comes another handy feature. Any application can be put on the button bar. Simply drag the icons you don't want on the button bar into the Extras Drawer and drag the icons you do want onto the bar.

What it looks like

Here's what the new screen looks like in landscape mode with the icon bar on the right.

< < Insert picture of MessagePad 2000 screen here, call out notable features: Arrows rotate with the screen, Assist off the button bar, Formulas on > >

How to look good

There are a few issues that developers must handle properly in order to have their applications look good on the MessagePad 2000 display. The main issue is dealing with the increased screen size and the soft button bar. Depending on how an application sizes its main view it may end up using only half the screen, or it may end up using the entire screen with interface elements in strange places.

As long as applications follow the recommended techniques for sizing the base view, most problems can be avoided. A naive way of sizing your base view is to set the viewBounds to hard numbers and to assume that the screen will be the right size to accommodate the view. When the MessagePad 110 was released, the screen size changed. This caused any application with hard viewBounds to look odd on the new screen size. Putting a viewSetupFormScript in your applications base view is the right way to avoid problems with the screen size. Here's an example.

```
func()
begin
  local  maxWidth := 200;
  local  maxHeight := 300;
  local  params := GetAppParams();
  self.viewBounds := RelBounds(
    params.appAreaLeft,
    params.appAreaTop,
    Min(maxWidth, params.appAreaWidth),
    Min(maxHeight, params.appAreaHeight)
  );
end
```

By using a maximum size that is reasonable for your application, you can be sure that all your careful layout won't be ruined by a suddenly increased screen size.

View justifications also need to be carefully planned so that increased screen size doesn't cause sub-views to suddenly change position.

Working with Gray Scale

Let's now turn our attention to the gray scale capability of the MessagePad 2000 screen. Gray scale is one of those features that can differentiate a run-of-the-mill application from one that's genuinely pleasing to the eye.

The new gray scale screen allows applications to display gray shapes, pictures and text. Along with the gray scale capabilities come several new features for Newton QuickDraw. These new features support anti-aliased bitmaps, Clipboard support and several new shape manipulations.

Gray QuickDraw can directly display 1, 2 or 4-bit color images. Pictures or shapes that use 8, 16 or even 32-bit color can also be drawn by interpolating the color values as grays. One thing to keep in mind is that sizes go up dramatically as larger bit-depths are used for pictures.

The Platform file for MessagePad 2000 will be able to directly import color pictures into packages to be displayed as gray. In the case of bitmaps, multiple versions of the bitmap at 1, 2, 4, 8, 16 or 32 bits per pixel can be included in the picture. The appropriate image will be displayed to match the display capability.

None of the built-in protos have been changed to take advantage of the new gray capability. There are a couple of reasons for this. The first and most obvious reason is space. Protos that use gray will simply take up more space than the current versions. More importantly, existing applications won't know that gray is being used. The results could be visually odd if the gray areas didn't blend reasonably. There is nothing preventing an application from defining its own protos that use gray to good effect.

All shapes can now include predefined color values for the gray tones or gray patterns that will be tiled into the shape. Fill and pen patterns can be defined.

Monochrome bitmaps can be anti-aliased by shrinking and gray interpolation. This is especially useful for fax or other scanned image viewing. If the bitmap being interpolated is not a bitmap (it has more than one bit per pixel) then anti-aliasing is not performed.

A number of functions are now provided for working with RGB data. A drawing application can pretend that it is running on a color display and the color values will be interpolated into gray levels. By working with RGB values an application can be ready for color in the event that future Newton devices provide color support.

There have been a number of capabilities added that have nothing directly to do with the new gray scale screen. These new capabilities make life a little easier for applications that are doing a lot of graphics work.

Clipboard data can now be converted into Shapes. This makes it easier for applications to allow users to transfer shapes between applications.

Shapes can now be directly flipped or rotated to allow applications more freedom in how shapes are manipulated. Bitmaps can now be displayed with a mask. This allows bitmaps to have odd shapes but to still occlude the background (more or less like Macintosh icons).

Ink is now easier to handle. First of all, you can get the list of point in either X,Y order or in Y,X (previously the only way to get the points), whichever suits your use of the data. Also, the number of points obtained can be tuned by specifying a distance between points.

Selection handles are now supported through the styles frame for a shape. You can also determine whether a point hits the selection handle. This helps applications that support direct user manipulation of shapes.

Summary

The MessagePad 2000 screen provides a lot of new screen real estate for applications that need larger areas to work with. The new gray capabilities and drawing function open a number of new arenas for application and user interface design.

NTJ

Newton Training

Developer Training Update

Our Excellent Time-Saving Courses are Moving to Where You Are!

As of January 1st, 1997, Newton Developer Training has expanded to offer two courses — Newton Essentials AND Newton Communications — and is now closer to your neck of the woods. See the schedule below and check out **http://devworld.apple.com/dev/newton/devservices/nsgtraining.html >** for registration information, including new links to earlier versions of our FREE on-line Newton Programming: Communications 2.0 and Newton Essentials 2.0 courses. Your feedback on these courses is welcome!

Newton Programming: 2.0 Essentials

Learn how to develop applications for Newton devices using the state-of-the-art development environment, Newton Toolkit for Macintosh and Windows, in combination with a very powerful, small, robust, object-oriented language, Newtonscript. The use of Newton Toolkit lets you interactively develop your applications without having to execute sequential edit, compile and link cycles. In addition, human interface guidelines for developing on PDAs are discussed.

Instructors: Neil Rhodes, Bill Worzel, or Adrian Browne
Duration: 5 days
Tuition: $1500

Newton Programming: 2.0 Communications

Learn the fundamentals of Newton Communications. Using a self-paced mentored approach, we cover information for Newton programmers who want to add communications code to their applications. Students will have access to code, articles, references and labs. A qualified instructor is available to work one-on-one with anyone having specific questions or problems. In addition, a module on the Newton Internet Enabler will be presented.

Instructors: Neil Rhodes, Bill Worzel, or Adrian Browne
Duration: 5 days
Tuition: $1500

Schedule

Class	Location	Date	Cost/Student	Registration
2.0 Essentials	Ann Arbor, MI	1/27/97	$1500	313-439-3828
2.0 Essentials	Palm Springs, CA	2/24/97	$1500	909-793-5995
2.0 Comms	Ann Arbor, MI	3/17/97	$1500	313-439-3828

Arroyo Software and Calliope Enterprises are both long-time providers of Newton programming training for Apple. Both companies provide instructors who are experienced Newton programmers as well as expert trainers. Arroyo Software authored the Newton Communications Course and Calliope Enterprises authored the Newton Essentials Course, both for Apple Computer, Inc. Educational and group discounts are available.

Onsite training can be arranged both within the United States and worldwide. Further information about both companies is available upon request.

Arroyo Software
214 West Main Street
Milan, MI 48160
(313) 439-3828
sobel@arroyosoft.com
http://www.arroyosoft.com/

Calliope Enterprises
700 East Redlands Boulevard, Suite 154
Redlands, CA 92373
(909) 793-5995
nrhodes@pobox.com
http://www.pobox.com/~neil/training.html

NTJ

Data Structures: Storing and Retrieving part 1 of 3

by J. Christopher Bell, Newton Developer Technical Support, llama@apple.com, http://www.doitall.com

Introduction

These articles focus on information useful to those who have mastered the basics of Newton data storage APIs. They assume that the reader is already familiar with NewtonScript and Newton data storage concepts in the Newton Programmer's Guide (NPG) and has some experience writing Newton applications.

The series consists of three articles entitled, "Data Structures," "Entry Caching," and "VBOs." The articles include some implementation-level details with which you can optimize and design your own applications. The first article in this series focuses on internal soup data structures and how to use them.

Under the Hood

By now, you have already used soup methods like `soup:AddXmit(entry)` and `unionSoup:AddToDefaultStoreXmit(entry)`. However, you may not know what goes on "under the hood" in the Newton OS to store and retrieve soup entries.

Without any extra data structures, finding an entry in a soup would be as inefficient as searching for a specific business card after a fan blew your business cards around your office. You might not be happy if you had to look at almost every card before finding the right one.

The "pick items in a random order" algorithm would be especially slow in the Newton OS because reading a soup entry from the store has a high overhead. While writing a NewtonScript frame to a soup, the frame is flattened into a binary object, similar to the process used in the `Translate` global function. While reading in a soup entry, the system reads the flattened frame from the store and un-flattens the frame into NewtonScript memory. Because of the un-flattening code and the memory management required to create the NewtonScript objects, reading in a soup entry is slow. The "Entry Caching" article later in this series will discuss the details of reading in entries.

The high overhead for reading in soup entries wouldn't be as much a problem if we read in an entry only when it was the "next" item we wanted. You might be wondering why we cannot read in all the entries into memory and sort them in place, like you might do on a desktop computer. Due to cost and mobility requirements, current Newton devices don't have enough memory to hold all the data of an average-sized soup. Instead, creating a cursor finds only the first entry. If you send cursor messages like `cursor:Next()` and `cursor:Prev()`, the system searches for adjacent entries.

In order to help your application minimize reading in entries, Newton soups maintain separate data structures. The system stores these data structures *in addition* to the entries, and these data structures do not order or modify the entries themselves. These data structures are called indexes and tags.

Indexes: from AA to zz

The most useful soup data structure is the index. An index provides quick access to soup entries as well as a means of retrieving entries in a defined order. A designated value from each item, called the index key, defines the item's sort ordering. For soup indexes, the key value could be derived from a slot value or multiple slots within the entry. Your code can use the index to find individual soup entries quickly (if you know the key value) or iterate through entries in a sorted order without reading in unneeded entries.

Most Newton applications create indexes when the soups are created. For instance, a bank check application might specify an integer index on a `checkNum` slot. This means that the soup entry's `checkNum` slot contains an integer that provides the key value for that index. This index allows the application to retrieve the entries in check number order (ascending or descending) or quickly find a check with a specific `checkNum`.

The system maintains each index as soup entries are added, deleted or changed. Or, you could add or delete indexes to soups when entries are already in the soup. If you create indexes for soups that already have entries, the system must iterate through all the entries, examine the key values, and place them in the index.

Note that some applications that add many entries achieve better performance if they do not set up indexes in advance. If an application receives data from a remote server and must add many items to a soup, updating the index information when adding every entry can slow down the connection with the server. If connection time is expensive, an application could set up a soup with no indexes, add many items, and then add the index after the connection is complete. Even without expensive connection times, adding many entries to an empty soup will be faster if you create indexes after adding all the entries.[1]

Because the system stores index information separate from the entries themselves, we may create multiple indexes to represent different sort orders. *Multiple indexes* are useful if you want to retrieve the data in two or more different sort orders.

If you wanted a single sort order based on *multiple slots*, you'd use a multi-slot index. For example, you could retrieve employee names in

[1] If the soup already contains entries, based on the number of entries already in the soup, it may or may not be faster to 1) remove the index 2) add many entries, 3) re-add the index. Or, you could add new entries to a temporary soup (with no indexes) during connections, and move the entries to the indexed soup afterward (this might take longer overall, but it might allow shorter connection times if that was important for your application)

alphabetical order by last name (the primary key), but use the first name (the secondary key) to decide ordering of employees with the same last name.

Multi-slot indexes do not define multiple indexes. Instead, the index extracts the key value for the index from two or more slots. Although this is oversimplifying the implementation, you can think of it as creating concatenated values for use as the key value for that entry. For instance, if the entries use separate slots for last names and first names, the system extracts both slots from each entry, creating single index key values like "Simpson*Bart" which define the sort order.[2]

A common misconception is that this is equivalent to multi-index queries, which are not currently supported. Multi-slot indexes define a single sort order. When you use a multi-slot index, you must create the entire index with the proper sort order specification before querying with that index.

An example of this misconception is trying to make an efficient query with the last-name-first index described above, retrieving all people with *first* names beginning with the letters A through C. You can think of the names as listed on a sheet of paper sorted by last name and then first name. Because there is only one sort order, the people with first names beginning with D through Z are interspersed throughout the sort order. The many undesired entries must be examined and then skipped over using a filtering method (which we will discuss later in this article), which is inefficient.

Indexes are B-trees

Indexes are implemented in a balanced tree structure called a B-tree. The important thing to know about B-trees is that finding an entry based on a key value is described as O(log n) in big oh notation (pronounced "order log n"), where n is the number of indexed entries. Moving from one item to the next item in sorted order can be done in O(1), or "constant time." For more information about big O notation and B-trees, see the cross-references section for more information (Baase, Sedgewick).

For instance, let us compare it to a hypothetical "brute force" index designed to minimize reading in entries. Imagine we created a simple index by storing each entry's key value and the soup entry unique identifier (unique ID) in elements of an unsorted array. We could iterate over the unsorted array for a particular key value but we might search most of the unsorted array for the desired key value. In comparison to this "brute force" index search time of O(n), a real soup index requires touching only a small fraction of nodes in O(log n) time, and subsequent searches for the next or previous entry in sorted order are O(1) time.

Note that for multi-slot indexes, search speed is approximately the same as for single-slot indexes. Multi-slot indexes use the same structure as for single-slot indexes, except that the system takes the key values from multiple slots in the entry.

When Entries Participate In Indexes

For single slot indexes, if the key value is non-`nil`, the entry will participate in the index. If the key is `nil` or not present, that index will not reference that entry.

For multi-slot indexes, the entry will participate in the index if any of the sub-keys are non-`nil`. If an entry is in a multi-slot index but a sub-key is `nil`, it will sort before any otherwise identical item with a non-`nil` value in that sub-key. For instance, in the multi-slot index for the employees soup described above, the following entries would appear in this order.

```
{last: nil, first: "Madonna"}  // nil value for primary key
{last: "Simpson", first: "Bart"}
{last: "Smith"} // nil value for secondary key
{last: "Smith", first: "Abigail"}
```

Urban Legends About Indexes

Here are few common misconceptions about indexes.

1) *Indexes do not order the entries themselves.* Soup entries themselves are stored as flattened frames in no defined order. The system copies the key values from the entries into the index structure separate from the soup entries themselves.
2) *Indexes are not free.* Indexes take time to maintain when entries are added, deleted, or changed. Indexes take space on your internal or PC Card store. Indexes are not automatically added for all slots in soups; you must add one index for each desired sort order.
3) *Indexes do not provide instant access to soup entries.* Finding an entry based on its key value takes O(log n) time (although it takes O(1) time to find the next/previous entry in the index). Note that if you are not filtering out unneeded entries in the index (see the next section), index operations are very fast, low-level operations in comparison to calling NewtonScript functions or reading in entries.
4) *Multi-slot indexes are not multi-index queries.* A multi-slot index must already be created for a particular set of slots before querying the index. When querying that index, there is only *one* sort order and only *one* key value derived from multiple slots.

Filtering: All You Ever Think About is Sets

Some applications want to query using an index but retrieve entries only if they satisfy certain criteria. You might say that you are looking for a certain "set" of entries in the index. For instance, you might want to find the set of all employees who "are married, don't have children, have the ABC health plan, but don't have a dental plan." You could add an index for every possible set-based query, but it would cost both speed and store space to maintain those indexes. Instead of using indexes to filter out unwanted items *and* define an order, you can use an index to define a sort order (and a range) and then filter out unwanted items with one of the query filtering methods.

When used on their own, filtering methods take O(n) time to find the next entry, since the system might examine and reject 999 out of 1000 items in the index before finding the first item in the desired set. Filtering methods do not create a sort order. If you want to retrieve the entries in a sorted order (or limit the range with query limiters like `beginKey` and endKey), you *must* create an index.

The simplest and least efficient filtering method is a `validTest`. A query's `validTest` function must determine whether the entry is or is not in the appropriate set. It is the most powerful filtering method because it can run an arbitrary function to determine whether the item should be filtered out. However, it is slow because the functions execute a

[2] You can also define directionality for multi-slot indexes. For instance, you could retrieve employee data sorted by last names alphabetically (ascending), and when there is a tie, show the employee salary sorted reverse alphabetically (descending).

NewtonScript function and usually require the system to read in every entry. Reading in entries is slow, so use this as a last resort.

A faster filtering tool is the `indexValidTest`. This is similar to a `validTest`, except that the NewtonScript function uses the index key(s) instead of the entry itself. This is faster because the system will not read in the entry. However, it is less flexible because the `indexValidTest` can only use the index key(s). Also, because index keys may be truncated after 80 bytes, the `indexValidTest` must be prepared for sub-keys to be truncated or missing.

The most important thing to note about textual queries is that strings are not flattened with the rest of the soup entry.

Tags: Not Just For Luggage

An even better solution for filtering entries is to use tags, Boolean flags you can attach to soup entries. Tags queries provide a way to test these Boolean flags using a simple syntax. Instead of providing a NewtonScript function to determine whether the item is in the set, the system checks the tags and determines whether to filter out the item. A query just using tags takes O(n log m) time to iterate through an index looking for appropriate tags, where n is the number of indexed entries and m is the number of entries in that soup with at least one tag. However, it is very fast (the constant is low) due to the help of an extra low-level data structure created especially for tags.

If we added the appropriate tags to the soup entries, we can create a `tagSpec` to search for the example "employees who are married, have no children, have the ABC health plan, but don't have a dental plan."

```
myQuery := unionSoup:Query({tagSpec: {
 all: ['isMarried, 'hasABCHealthPlan],
 none:['hasChildren,'hasDental]
}});
```

Since all the tags are Boolean, the tags mechanism can store tags for each entry using compact bit fields. The system stores these bit fields in a data structure called a tags data structure[3]. The tags data structure uses an entry identifier as the key value for a B-tree, with the tags bit fields as the data within each node.

As you might guess from our earlier discussions of B-trees, the system requires O(log m) time to find the tags for the entry and O(log n) time to find the first entry in the current index. Note again that n is the number of indexed entries and m is the number of entries in that soup with at least one tag. In a query that just uses tags, the system must repeat this process for potentially every entry (O(n) time) until your `tagSpec` matches the tags for an entry. This means that the system takes O(n log m) time to find the first match for a tags query. You might note that this may *sound* slower than `validTests`. However, a tags query does not need to execute NewtonScript code, read in entries, or manipulate NewtonScript objects, and the system uses efficient index and tags data structures to test entries. Overall, the result is *much faster* in comparison to any other filtering method we have discussed so far because of these extra data structures.

Textual Queries

Because it is similar to the other filtering types already discussed, I want to briefly mention textual queries. Like the other filtering methods, textual queries do not define an order. If you want to retrieve the entries in a sorted order (or limit the range with query limiters like `beginKey` and `endKey`), you *must* create an index.

The most important thing to note about textual queries is that strings are not flattened with the rest of the soup entry. There's a separate data structure for storing strings. Unlike the other soup data structures using for retrieving data, this data structure is created for all strings and requires no extra store space nor explicit creation.

You can think of the strings stored as one long concatenated string for each soup entry. While writing entries to the store, the system writes string data (from anywhere in the entry) to a separate place on the store.

During textual queries, the system need not examine individual frame slots nor read in the entry. Because of this, the system can perform textual queries at a low level using this structure. Textual queries might potentially search all indexed entries, but the text-searching speed is very fast in comparison to filtering mechanisms that use NewtonScript functions to evaluate whether an item should be filtered out.

If Speed Matters: Some Raw Data

To compare different filtering methods, here is some speed data with multiple methods and a similar search. The test is based on a 1000 entry soup with about 700 bytes of data in each entry, including 200 bytes of text, and string searches are performed on a 40 character `myString` slot. Each search answers the question: how many entries have its `myString` slot start with the string "`blah?`"

Note that all searches were designed to search the entire soup as the "worst case scenario" and these timing samples apply *only* to this particular set of test data. Speed of particular filtering methods will vary based on your data, the complexity of your searches, and on each Newton device. These tests were performed on a MessagePad 120 with Newton 2.0 OS.

A `validTest` query takes 2225 ticks. The `validTest` read in every entry and tested the `myString` slot with the `BeginsWith` function.

An `indexValidTest` query takes 354 ticks. The `indexValidTest` tested the indexed `myString` key and tested it with the `BeginsWith` function.

[3] You'll see other documentation that calls these "tags indexes", but they are not indexes in the way we've been discussing them. You cannot use them to retrieve items in a sorted order. However, to use tags you must treat `tagSpecs` as `indexSpecs` when used with `store:CreateSoupXmit(...)` or `soupOrUnionSoup:AddIndex(...)`.

A `text` query takes 654 ticks. It had to search 95 bytes of text in each entry.

A `words` query takes 255 ticks. The words query had to search 95 bytes of text looking for "`blah`" at the beginning of a word. It is faster than the text query because the test data had one long word in each string, so most of the time was just searching for word beginnings and not comparing text.

A tags query optimized for this test takes 128 ticks. When adding the data, we determined whether this particular entry had its `myString` slot start with the string "`blah.`" If it did, we added a special `hasBlah` tag to the entry. Since we were prepared for this query, we used tags to pre-calculate the filtering test and then did a tags query to retrieve the items.

An index query on the `myString` slot takes 5 ticks. The string index query used `beginKey` and `endKey` to limit the range in the index to the ones starting with "`blah.`"

An index query optimized for this test takes 1 tick. When adding an entry, we determined whether this entry had its `myString` slot start with the string "`blah`". If so, we set a new `hasBlahString` slot to an integer instead of `nil`. If we add an integer index on the `hasBlahString` slot, we can use an index to iterate over *only those entries* which match our required criteria (and in a certain sort order). In this example, there was only one entry that matched the desired criteria so the index was very small. Note that no query filtering methods were used.

In the previous tests, no more than one filtering mechanism was used at one time. In your own projects, you can use several filtering methods within one query, and also use range limiters like `beginKey` and `endKey` if desired. Check the NPG for full details about the various query options.

I strongly recommend that you experiment with several designs for your soups and combinations of query methods before deciding on an implementation. Also, remember that you can store data in multiple soups if that makes your queries easier to conceptualize and faster to query.

If Size Matters: Some Raw Data

Most of the previous information focused on information to help you optimize your code for speed. Some applications must make performance tradeoffs based on the size of the data and the soup data structures. Of the soup data structures mentioned in this article, the ones which take up extra space are indexes and tags.

Indexes always copy key values for the data. For integers, characters, and symbols, the copied data is small. For reels and strings, more binary data is copied to the index (8 bytes for reels, up to 80 bytes for strings). The maximum amount of data copied for a key value is limited. For instance, no more than 39 characters would be used in a string index, and for multi-slot indexes, sub-keys in multi-slot indexes may be truncated or missing in the index.

String indexes are particularly large, and this could significantly affect storage requirements for some applications. For instance, the Newton 2.0 ROM Time Zones soup is a package store part created in NTK. The Time Zones soup contains strings for city names and country names, and most of the names are short. Since the strings are indexed to retrieve city names in sorted order, the short strings are stored twice: once as part of the soup entry and once in the index. Including the size increase when converting ASCII strings to double-byte Unicode, the size of the Time Zones data was between 3 and 4 times larger in a package than in an ASCII text file.

On the topic of storage size, I want to mention an optimization used to store strings in soups (but not in string indexes). If the characters in the strings in an entry are Unicode characters with the high byte 0 (chars 0-255), only one byte is stored per Unicode character. The system transparently optimizes such strings when reading and writing soup entries to the store. This is the only transparent compression for soup data (other than the frames themselves).[4]

The system also tries to optimize the indexes themselves. When multiple entries have identical index keys (for instance, people with the same last & first names), the system stores the index information for those entries in a more compact way than saving a copy of the key for each entry.

Index size will vary on individual ROMs and vary based on the number of repeated index keys, but here are approximate index sizes on an Apple MessagePad 120 with Newton 2.0 OS. I have included index sizes for both short (15 chars) and long (40 chars) strings. Note that for some data types, I have included "with repeats" versions which means that the 1000 entries have only 1 of 10 possible values instead of unique values. For single-slot indexes of various index types using a 1000 entry soup, the index sizes are: Int: 15K, Char: 15K, Real: 17K, ShortString: 46K, LongString: 100K, LongString (with repeats): 4.5K, Symbol: 12.5K, Symbol (with repeats): 4.5K. Adding 2 tags each of 20 tags took 27K in the test soup.

Note that if you have read-only data, I strongly recommend that you consider storing your data in packages. See the "Lost In Space" article and sample for more information. For more information about making store parts and using package compression, see the NPG and the Newton Toolkit User's Guide.

Designing for Performance

There is too much variety in application data for me to recommend specific implementations. However, I can offer some things to consider when designing your soup structure.

1) What sort orders will you need? This may help you design your soup entry format and your indexes.
2) *What sets will you need to retrieve?* Determining what queries you'll make will help you design indexes and tags. I recommend that you use `indexValidTests`, `validTests`, and textual queries as a last resort only.
3) *Experiment.* Before implementing your entire application, test several implementations with real data or dummy data that accurately reflects the size and types of data in your final application.
4) *Determine if any data is read-only data.* If so, I recommend that you consider storing your data in packages. Check out the "Lost In Space" article and sample for more suggestions and information.
5) *Use the fastest filtering method that can do what you need.* For overall speed, here is the order of preference for filtering methods: tags, textual queries, `indexValidTest`, `validTest`. You might want to use more than one.

[4] There is another store optimization for storing frames called "frame map sharing" that is outside the scope of this article. Also, Virtual Binary Objects (VBOs) can be explictly compressed when created (see the NPG for more info), and symbols are shared in symbol tables. See the NTK Users Guide for more info about compression and object-sharing options for packages.

Conclusion

Here are the things that are the most important to remember.

1) Indexes are required to retrieve data in sorted order.
2) Indexes are stored as B-trees and take O(log n) to find an entry.
3) Multi-slot indexes define a single sort order and use a single B-Tree.
4) Tags are important for retrieving sets, but are not indexes. Tags do not define a sort order.
5) Tags are used in conjunction with an index, even if it is the default index.
6) Indexes and tags data structures aren't free. They take time and storage space to maintain, so decide carefully what you need.

I want to reiterate that if you are ever in doubt of how to optimize your application for speed and/or size, experiment with several different implementations. In the words of the composer Cole Porter, "Experiment. Make it your motto day and night."

Thanks to Bob Ebert for his help with this article.

Cross-references

Apple Computer, Newton Programmer's Guide *(NPG) 2.0 and* **Newton Programmer's Reference 2.0**, *1996. Apple Computer, Inc. Available on-line on Newton Developer CD-ROMs. The NPG 2.0 will also be published in hard copy by Addison Wesley.*

Baase,Sara, **Computer Algorithms**, *1988. Publisher: Addison-Wesley. Find more information on-line at http://heg-school.aw.com/cseng/authors/baase/compalgos/compalgos.html and http://www.amazon.com/exec/obidos/ISBN=0201060353*

Engber, Michael, **Lost In Space article** *and* **LostInSpace code**, *1995. Publisher: Apple Computer, Inc. Find them on Newton Developer CDs.*

Porter, Cole, **Experiment.** *Copyright 1993 (Renewed) by Warner Bros. Find more information about Cole on-line at my Cole Wide Web, http://www.doitall.com/cole*

Sedgewick, Robert, **Algorithms,** *1988. Publisher: Addison-Wesley. Find more information on-line at http://heg-school.aw.com/cseng/authors/sedgewick/algo-in-c/algo-in-c.html and http://www.amazon.com/exec/obidos/ISBN%3D0201066734*

NTJ

Marketing Strategy

NSG's commitment to Health Care

by Vernon W. Huang, M.D., Apple Computer, Inc.

A great deal of our developers are involved in health care applications. The Newton Group recognizes this, as well as the natural fit, for Newton technology in health care and the size of the health care market. Add to this the political pressures on health care to improve or to maintain the quality of care while cutting costs; and you have a compelling argument why the Newton OS has a chance at really making a positive contribution to the way health care is delivered.

In order to establish the Newton OS as the premiere Operating System for mobile computing in health care, the Newton Systems Group has formed a health care group comprised of full time resources as well as marketing and sales support. In this article, I will outline the events that lead to the formation of the health care group, the groups planned activities for this year, and how we plan to collaborate with developers.

Focus On Home Health Care

The majority of the health care group's focus, in the remainder of 1996 into 1997, will be on the home health care market. This is a sub segment of the overall trillion dollar health care market that is growing at a remarkable rate. Currently, home care comprises approximately three percent of the overall health care budget. By the year 2000, this should reach ten percent. The home care market has been growing at an annual rate of more than 25% over the last five years and is anticipated to grow at 15%+ annually until the year 2000.

In the United States, home care is the health care that is typically delivered in the home setting by nurses, technicians, and therapists. It can be in the form of wound care, physical therapy, infusion (IV) therapy, dialysis, or assistance in activities of daily living; such as bathing.

These nurses and allied health personnel, generally do all of their record keeping on paper and perform a great deal of redundant data entry. A typical nurse might spend two and a half hours per day on paperwork. Some attempts to automate these mobile professionals, have been made: using laptops, slate type computers, or proprietary devices. However, there is currently no clear leader in the industry. Thus, there exists a unique opportunity for Newton developers to deliver a whole product solution comprising of Newton OS devices remotely connected to a back end server. As in most dispatch applications, a real time connection is not usually a requirement. Data can be exchanged in batch sessions, usually via modem.

In order to achieve our goal, to be the recognized force in health care and home health care, the Newton Systems Group has allocated significant resources to the Healthcare Solutions Group.

In order to achieve our goal, to be the recognized force in health care and home health care, the Newton Systems Group has allocated significant resources to the Healthcare Solutions Group. Many of these activities involve co-marketing or collaborating with our developers.

New Health Care Solutions Guide

We are currently putting the finishing touches on our, *Newton in Health Care Solutions Guide*. Unlike the previous years, our new guide will be professionally produced by an agency. It will be printed in four colors. The Solutions Guide will cover success stories; home health care; medical education, reference and research; general medical software and developer tools and SI's with healthcare experience. Revisions will be made in the second and fourth quarters of 1997. Developers interested in receiving copies of the Health Care Solutions Guide or inclusion in future editions, should email <huang@newton.apple.com> and place the words "*solutions guide request* or *solutions guide entry*" in the subject of their email.

Distributing Your Marketing Materials

The Solutions Guide will be only one piece of printed material that we will be sending to our potential customers. We are also creating a health care folder which will contain the Solutions Guide, white papers, and developer supplied materials. If your company has a medical product or provides custom programming services for health care, we can help you distribute your marketing materials. To participate in this program, mail a sample of your letter size glossy product sheets to

Dr. Vernon Huang
Newton Systems Group
One Infinite Loop MS 305-3A
Cupertino, CA 95014

We'll do our best to include your materials in our mailings, but cannot make any promises until we see the content.

Industry and Trade Shows

In addition to the shows that the Newton Systems Group traditionally has representation at (Comdex, MacWorld), the Health Care Solutions Group will be participating in four shows this year beginning with the Annual Meeting of the National Association of Home Care (NAHC) in Nashville, TN. Developers interested in participating in our booth space at these shows, should make sure that we know of the status of your current products. In addition, your products can be demoed at these shows and at other sales opportunities if the Health Care Group has a sample copy of your application, even if your company can't participate in person.

Assistance Engaging Customers

Often times having a representative from Apple along can help assuage fears that a potential customer may have about the Newton Group's commitment to the health care market or Apple Computer's commitment to Newton. Please be certain to let us know if you have an opportunity to make a high level presentation to a qualified health care customer. With Time and resources being what they are, we can't always promise to be able to go in with you in person, but we'll do what we can to assist your efforts.

On-Line Marketing

As the Newton Web page gets overhauled, we plan to launch a "Newton in Health Care web" page. At this site, customers will be able to read about success stories and view an on-line version of the Health Care Solutions Guide with links to the developers home pages. Stay tuned on how to participate in this opportunity.

We're Here For You

Aside from our commitment to home health care, we are determined to make the Newton OS the standard in all aspects of health care delivery and education. Our Bookmaker and our Press technologies have already started a revolution in electronic medical references. Several of our developers are achieving significant first or second round financing and are growing at a tremendous rate. If there is a segment of the health care market that you feel we are neglecting, or if you have additional ideas in how we can co-market solutions, please send email to <huang@newton.apple.com>.

NTJ

Newton Technology Conference 1996 Technical Overview

This article summarizes the technical information that was presented at the Newton Technology Conference 1996. Additional information on each of these topics is provided elsewhere in this issue.

Text Engine

The Newton "edit view" view class has been part of the standard Newton user interface since Newton 1.0. It was extended in Newton 1.3 to handle deferred recognition, and then extended in Newton 2.0 to handle pictures. However, it is limited as simply a container for unrelated view children, not easily extensible as a complex word processor.

In Newton 2.1, we introduce a new text-editing engine called protoTXView. This text engine allows embedded graphics within text, paragraph formatting features found in desktop word processors, and increases the maximum size of documents (stored in Virtual Binary Objects).

The new text engine allows paragraph- by-paragraph style information like left and right indents, complex tab stops (left, right, and decimal), and lineSpacing. An optional "ruler" user interface allows modification of these settings. This paragraph-by-paragraph information is called "ruler information."

Methods are provided for getting and setting information about styles (bold, italic, underline), ruler information, and text. These APIs are based on the concepts of ranges and runs. Ranges represent places within the text, represented by character offsets, like {first: 0, last:5}. Runs represent style or ruler information with arrays of values representing pairs: numbers of characters and the value. For instance, [3, fontSpec1, 5, fontSpec2] represents eight characters of style information, with three characters of fontSpec1 and 5 characters of fontSpec2. Run arrays are used to get and set style and ruler information.

To use a basic protoTXView, just add the required view slots (viewFlags, viewJustify, and viewBounds) and use the SetStore and SetGeometry methods in your viewSetupFormScript. To save/retrieve data, you use the Internalize and Externalize messages. Note that the externalized version of the data that you can save in soups is not in a documented format.

There is an important decision to make for your own protoTXView: paged or non-paged data. Paged text indicates that there is a defined "page height" and horizontal lines appear at the end of each page to indicate the pagination. Also, if a "page break" is inserted with the InsertPageBreak message, white space will appear for the rest of that page (possibly advantageous and possibly confusing, depending on your application and your target audience.)

Whether your view is Paged (the name of the argument to SetGeometry) affects other parts of your protoTXView development. For instance, functions like GetTotalHeight work differently depending on whether the view isPaged. If your view is NOT isPaged, you'll probably want to set the height of your view with SetGeometry to be arbitrarily large (see the sample for details). That will allow scrolling and clipping to work fine. In order to find the "real" height of the text (for scroll arrows or scroll bars), see the sample on how to use GetTotalHeight or a custom (sample code) method GetTextHeight to do the right thing.

There are also hooks for doing in-line pictures using graphics shapes. To do that, add a graphic of the form {class: 'graphics, shape: aShape} as if it were text.

If you want to do "hypertext" processing, you can use methods like viewClickScript, PointToChar, GetWordRange, GetRangeData to do it.

If you want to do "find," you have two choices: use an open protoTXView-based view, or use the Externalized data with the protoTXViewFinder object. There are methods like Find and ReplaceAll in protoTXView, and you can do simple text searching with protoTXViewFinder (useful for implementing global 'find,' since your application may not be open.)

The Apple eMate™ 300 and the Apple MessagePad™ 2000 have a new, larger screen. The new screen is 320x480, one half VGA. The new screen provides 16 levels of grayscale.

There are, however, limitations. First, graphics are always visually "in line" in the text. You cannot intelligently wrap text around large pictures. Also, no recognition capabilities are built into protoTXView. You can provide limited recognition with viewWordScript, but you will not be able to use "overwriting" or the corrector views. And finally, the data format that is exported from protoTXView for saving to a soup is in an undocumented format. You can, however, do simple text searches in saved data without opening a protoTXView -— see the documentation for protoTXViewFinder.

For more information, see the documentation and the DTS Sample "TXWord" to see how to use the APIs for font/style menus (MakeFontMenu), scrolling (Scroll, viewUpdateScrollersScript, GetScrollValues), and inserting graphics.

Grayscale Graphics Abstract

The Apple eMate™ 300 and the Apple MessagePad™ 2000 have a new,

larger screen. The new screen is 320x480, one half VGA. The new screen provides 16 levels of grayscale. Shapes and text can be rendered using gray fills and gray patterns.

Color support has also been added. Colors are specified as RGB values that get converted to gray tones when rendered. Pixel maps can be created using multiple bit depths. Each depth can specify its own color map. The color map will be converted to grays when the pixel values are displayed.

PICT 2 is supported now as well. Most opcodes for PICT 2 as defined in Inside Macintosh are supported with the exception of a few drawing modes related to color. NTK 1.6.4 will provide support for directly importing color PICTs into NTK projects.

Shape handling has been improved, with more flexible manipulations now supported. Selection handles can be displayed at the bounding corners of a shape, and FindShape will consider the selection handles when determining which shape has been hit.

Bitmaps can be resized and anti-aliased. The anti-aliasing results in a pixel map that uses gray values.

The User Interface protos have not been changed for gray scale with a single exception: protoFloatNGo now uses a gray outline. Any proto that supports displaying of text of shapes can display gray images.

Keyboard

Extensive enhancements have been made to improve support for keyboards in Newton 2.1 OS. These enhancements include the following.

- New APIs to capture keyboard input.
- Notification of key view focus changes.
- Support for key commands.
- Command key support in pickers.
- Default buttons and close boxes.

Newton 2.0 OS laid the groundwork for keyboard support and included a limited API to intercept keyboard input. Newton 2.1 OS extends this API so that any type if view can accept keyboard input. The OS also provides new methods to intercept key down, key up, and key repeat events. In addition, views are now notified of when they gain key view focus and when they lose key view focus. The Newton user interface has also been extended to include visual cues that a view is the key view.

One major addition to the OS is an API to handle key command combinations. You can now provide an event handler which will be called when a particular key combination has been pressed. Key commands are contextual which means that you can have one view in your application which has a different set of key commands than another view. For application-wide key commands you could register the commands with your base view. You can additionally have system-wide key commands by registering key commands with the root view.

A help slip is available which will give you a list of key command possibilities for the currently active key view. This help slip is always available by pressing and holding the command key for approximately two seconds.

There is also support for showing key equivalents in your application's pickers. When the user has a picker open and types the key combination, you can specify whether the OS will directly call your key command event handler or whether the picker's pickActionScript will be called. The OS handles displaying what the key combination is for each item in the picker.

Lastly, you can create a default button which will respond to the return key. This button will be drawn with a darker border so the user knows it is a default button. Additionally, you can specify that a closebox respond to command-w or command-period. A closebox could be either a button or the standard Newton closebox.

Sound

The new version of the Newton OS adds the ability to record sound, as well as providing more support for playing back sounds. There are UI elements which make implementing sound capture very easy, as well as the ability to record sounds with control over many options.

The MessagePad 2000 has an internal microphone, whereas the eMate 300 has a headphone jack. Both devices support sound input and output on the interconnect plug.

There are two built-in user interfaces for recording and playing sound. One is designed to be embedded inside your application's views, and the other is in a "floating" slip with its own close box. Both provide standard record, stop and play buttons, with an indicator to show the length of the sound.

If more control over the recording process is needed, your application can use the methods of protoSoundChannel. To record, first you initialize the sound channel and set its parameters, then you queue one or more sound buffers, then start recording. You can queue more sound buffers during the recording session. Eventually you stop the process and clean up the objects.

There are a number of sound compressors and decompressors you can use when playing and recording sounds: MuLaw, IMA and GSM. An FM synthesizer supporting up to 4 sine waves is a built-in "decompressor," and a Macintalk speech synthesizer, while not built in to the ROM, does exist as a package.

Newton Works

Works is a new document framework which comes pre-installed on the eMate 300 and will be included on a disk for the MessagePad 2000. This framework is intended to provide a simple yet powerful shell for productivity applications, much like the desktop products with similar names.

Four applications will be initially available in the Works framework: a word processor, a drawing tool, a spreadsheet, and a calculator with graphing capabilities.

Works uses a document metaphor that clearly separates individual projects. Unlike desktop productivity applications, Works on the Newton platform is extensible through the stationery mechanism. To add a new type of document, simply register a view definition and a data definition using the existing stationery API. The superSymbol slot of your data definition should be 'newtWorks. New stationery should try as much as possible to maintain easy-to-understand interfaces and use established patterns for user interfaces such as command key equivalents and menus.

Works defines a variety of required and optional APIs that you add to your stationery's viewDef and dataDef. These APIs cover preferences, scrolling, finding, registering tools, providing help, and manipulating the status bar.

For more information on using Works, see the Works article and the DTS sample code about Works.

Multi-User/Simple Mode Abstract

The Apple eMate™ 300 is intended for an environment where several students may be sharing a single device. A teacher can easily set up an eMate™ for the students in a class. Simple mode also allows for a simplified user experience with the eMate™.

The teacher may also want to limit the availability of applications in the Extras Drawer. When setting up the eMate™ 300, the teacher can select which applications will appear in the Extras Drawer and which will not. For example, the teacher may decide that having access to the Calculator would not be a good idea during an arithmetic test, so the Calculator can be removed from the Simple Mode Extras Drawer.

Each student's data is kept separate. eMate™ 300 aware applications can determine who the current user is and use that to filter the information available to the user. Apple strongly recommends that each user have a separate soup for storing information. When in Multi-User mode, the eMate backs up data specific to the user, helping to keep different users' information separate.

New students can be easily added to the unit. Creating a new account is simply a matter of assigning a name and a password. At the teacher's option, a password need not be required. Optionally, students can add themselves to the unit when they first sign on.

The CDIL provides you with a cross-platform, media-independent API for a virtual pipe through which you send and receive data.

Classroom Connection

When students want to import and export data from their Newton eMate 300s, they will connect over AppleTalk or a serial cable to the "Classroom Connection" server. This server makes backups of the user's data (both NewtonWorks data and third-party application soups) and allows selective import of the backed up files.

For data stored by NewtonWorks stationery, a single file is created on the desktop for each soup entry, containing the soup entry frame as well as a small amount of header information. Non-NewtonWorks packages have their entire soup or soups backed up to a single file.

These desktop files are readable by desktop applications which will probably use the DILs to convert to and from the Newton frame format. If you have control over the desktop application, you could have it read and write the Newton format automatically.

Alternatively, writing an XTND translator or a drag-and-drop converter application will be a good way of converting the data from the Newton format into one suitable for an existing desktop application. XTND translators are being developed to allow applications to read and write the NewtonWorks drawing and word processing documents.

In order to provide custom icons, creator codes and filetypes for the desktop files, the Classroom Connection server will work with developer-created extension files. The provided icons and other resources will be matched with the Newton data based either on the class of the NewtonWorks soup entry, or (for non-NewtonWorks data) the soup name.

DILs

The DILs (Desktop Integration Libraries) are free libraries that you use in your desktop applications (MacOS and Windows) to communicate with a Newton device. The DILs make it much easier both to move the data as well as translate from the Newton format into a non-NewtonScript application.

There are three varieties of DILs: the CDIL (for basic communications), the FDIL (for interpreting frames and other NewtonScript objects) and the PDIL (to speak the Connection application's protocol, which enables AutoDocking). The CDIL and FDIL were released in January 1996; the PDIL will be in beta form soon. An update to the Windows DILs was released in November.

The CDIL provides you with a cross-platform, media-independent API for a virtual pipe through which you send and receive data. It optionally performs byte-swapping and Unicode conversion for you, and is the core upon which the FDIL and PDIL are built. The CDIL API is at a higher level than traditional desktop communications APIs are, and insulates the developer from the complexity of communications protocols.

The FDIL provides a binding API to translate NewtonScript frames, arrays and their contents into C-style structures or other data buffers used in desktop applications. Since frames are highly flexible and dynamic objects, the application provides a template to extract their data into the C structures, and then can access the "unbound" or freeform data as necessary.

The PDIL provides higher-level access to the Newton device, and since it communicates with the built-in Connection application (now renamed Dock), writing NewtonScript code is no longer necessary to synchronize with or transfer data to and from a Newton device.

The PDIL libraries are what enable AutoDocking on new devices, and also can communicate with Newton 2.0 devices. PDIL functions provide full access to Newton soups (including queries), let developers download code and packages, and allow the desktop to remotely call functions and methods on the Newton device.

IrDA

IrDA (Infrared Data Association) is a consortium founded in 1993 as a non-profit organization. Its goal is to create and promote interoperable, low cost infrared data interconnection standards that support a walk-up, point-to-point user model. The standards support a broad range of appliances, computing, and communications devices.

IrDA has defined a set of industry standards which provide robust, low cost, low power, high performance point-to-point infrared communications. The new version of the Newton OS adds support for three layers of the IrDA specifications. Specifically, these are the asynchronous serial ("SIR" or "physical") communications layer, the link access protocol ("LAP") layer, and the link management protocol ("LMP") layer.

The SIR layer is implemented on the Newton using a UART and is capable of speeds from 2400 bps to 115.2 Kbps. The LAP layer provides the error correction and reliable datagram delivery mechanism. The LMP layer implements a simple name server and link multiplexer and is the layer upon which endpoints are based.

IrDA communications are physically half-duplex, yet the protocol "simulates" a full duplex serial communications link. Applications which work over a standard asynchronous serial connection can be quickly and easily modified to use IrDA.

IrDA is a networking protocol of sorts. Devices have types and names, and connection parameters such as speed and window size are negotiated between devices automatically. The standard is continually evolving and improving.

IrDA is not compatible with the IR beaming protocol used in the MessagePad 130 or any of the previous MessagePad models. In order to provide backward compatibility in beaming, the beaming transport in the Newton OS has been modified to detect which protocol is in use by the sender/receiver, and to use either Sharp ASK, Newton IR, or IrDA as appropriate. Full IrDA specifications are available on the web at <http://www.irda.org/>.

Mail Enabler

With the release of Newton 2.0, developers were pleased to find supported APIs for developing transports that can integrate into built-in and third-party applications. However, these APIs are not specifically directed at mail-specific transports. Developers of text and text-and-attachment transports still must write code and design dialogs to achieve a consistent user interface with standard transports like the Compuserve© client.

For Newton 2.1, Apple designed the Mail Enabler protos. Apple will release this suite of protos in a future NTK "platform file" for use with third-party transports. They are NOT in the ROM, even though they are part of the Newton 2.1 project, and will be released when the Newton 2.1 ROM is finalized. They will enable developers to simplify the development process for mail transports for both Newton 2.0 and Newton 2.1.

The features include mail-like routing slips, connection slips, automatic text export, mail header, text dataDefs and stationery, and automatic toggling between text and attachments for transports that support both dataTypes.

To use the Mail Enabler protos, you still must write your own comms code. However, much of the user interface code and transport code is simpler. As before the Mail Protos, most of your transport development time will probably be specific to your endpoint and protocol code.

Who should use Mail Enabler? Any text-only or text-and-frame (attachment) transport that has a mail-like routing slip. There are many optional features, like cc/bcc, attachment handling (the 'frame dataType), and optional preference items you can add to your preference slips for changing the mail font, etc.

The mail transport proto also handles automatic export to text appropriately, and knows how to switch between viewing "text" and viewing the "frame" within the Out box (via a GetTransportScripts menu item).

For routing slips, you get to/cc/bcc pickers for free, as well as a "show message" text viewer. You can add additional views to the routing slip.

To use the Mail Enabler connection slip, you must provide a method to return an array of phone numbers and baud information, given the current worksite and city information (and also a view to open if the user selects "Other phone number" from a phone number picker.)

If you are preparing to write a mail transport now, learning about transports and communications endpoints before starting development is recommended. See the DTS transport samples "MinMail" and "ArchiveTransport" for examples of writing text mail transports *without* Mail Enabler. Also, see the NTJ article "FSM Article," and the samples "Altered States" and "CommsFSM."

Newton Internet Enabler

NIE (Newton Internet Enabler) is a software package which extends the communications capabilities of the Newton OS. Specifically, NIE provides TCP and UDP endpoints over PPP and SLIP physical links. For security, version 1.1 adds PAP, CHAP, and Interactive Authentication support. NIE also provides a non-recursive client interface to a DNS, as well as automatic ICMP responses.

NIE is composed of three main components: Link Controller, Domain Name Resolver, and Internet Communications Tool.

The Link Controller is responsible for managing the physical connection to an internet service provider. Multiple applications can share the same physical link to the service provider. Only one physical link may be active at a time.

Using the Internet Setup application, the user defines the IP information necessary for communication (for example, the protocol to use, the local IP address, the IP address of a domain name server, the phone number, et cetera) as well as the login script the link controller uses to establish the

If you have an idea for an article
you'd like to write
for Newton Technology Journal,
send it via Internet to: NEWTONDEV@applelink.apple.com
or AppleLink: NEWTONDEV

connection. Once defined, the user simply refers to this connection information by name. Multiple connection setups may be stored, and a particular setup recalled when establishing the connection.

The Domain Name Resolver is a simple non-recursive "stub" interface to a Domain Name Server. The DNR can resolve host names to IP addresses and IP addresses to fully qualified domain names for both standard and mail server name entries. The DNR requires authoritative responses from the DNS.

The Internet Communications Tool is a standard communications service which implements UDP and TCP endpoints. Note that although UDP is not a connection-based or stream-oriented protocol, the endpoint API is used with packetization flags to send and receive individual UDP datagrams. For TCP, the full protoBasicEndpoint API (sans end-of-packet flags) for reliable stream-based connections is supported. Multiple endpoints may be simultaneously active and sharing the same physical connection to the service provider.

Programming for NIE is essentially no different than for any other protoBasicEndpoint service. Only a few additional steps are required, specifically, choosing a physical connection setup and establishing the physical link, and tearing down the physical link when finished. For as long as the physical link is established, the application simply uses endpoints like any other communications application on the Newton. For more information, please refer to the NIE web page at <http://devworld.apple.com/dev/newton/tools/nie.html>.

Building A Better State Machine

What is a Finite State Machine or FSM? Quite simply, it's an ordered system of states, state transitions, input events, and output results. An FSM is an implementation of a logical model. Conversely, state diagrams, state tables, and state outlines are logical models of the implementation.

A state diagram (sort of a cross between a Venn diagram and a Karnaugh map) is usually the first step in defining an FSM. From here, a state table or state outline is usually generated. Creating a state diagram is an inductive process, utilizing the power of the human mind to garner the often abstract relationships between components of the system. FSMs are not a panacea of program development and problem solving, but they are extremely useful tools in many kinds of software development.

The use of Finite State Machines (FSMs) is but one method of defining a sequential process, finding its weaknesses, and preparing for its implementation in an obvious, deterministic fashion. FSMs can be applied to just about any sequential process, from language parsing to process control. And for computer communications applications in particular, they are often a very powerful tool.

There are essentially two "types" of Finite State Machine: Deterministic and Non-deterministic. In a deterministic FSM, for any given input event there is exactly one unique state transition. In a non-deterministic FSM, for any given input event there are multiple possible state transitions. A non-deterministic FSM may become deterministic in some future state, but the goal is generally to make the FSM as deterministic as possible, as early as possible.

Many issues arise when dealing with FSMs. One of the most common is the problem of synchronization; organizing the parallel activities of two or more disparate FSMs. Among the many ways to enforce synchronization behavior is by the use of "Do" and "Done" events, and "Wait" states. A "SyncFSM" coordinator is used as a higher-level interface.

Another, perhaps more obvious, issue is the fact that the number of states in an FSM can easily "explode" as new required state transitions are added to the system. This "factorial explosion" behavior is often addressed through the use of state reduction techniques. In essence, flow-control intelligence is added to the system in order to reduce the sheer number of states and volume of code required for implementation. Incorrect state reduction can quickly lead to unstable or non-deterministic finite state machines, however, so the developer is cautioned to proceed in this area with great care.

There are other issues to be addressed when dealing with finite state machines, far too many to list here. For more information and some very useful examples of FSM implementations, please refer to the Newton DTS sample code projects "protoFSM", "Altered States", "ArchiveTransport", "Comms FSM", "Mini-MetaData", and "Thumb," to name a few. NTJ

Updating Your Application Checklist

Phase I: Preparation

- ❑ Run your application and see what breaks
- ❑ Fix the items that prevent the application from running

Phase 2: Further Investigation

- ❑ Check speed issues
  - ❑ *repeating buttons*
  - ❑ *scrolling*
  - ❑ *look for calls to* `BusyBox` *and test*
  - ❑ *redo areas that use loops for timing (use* `Ticks` *instead)*
  - ❑ *disable status display boxes and check speed*

- ❑ Screen size issues
  - ❑ *Look at all your layouts*
    - ❑ Check in both rotations (portrait and landscape)
    - ❑ Check on MessagePad 2000 and eMate 300
    - ❑ Check with Button Bar on the left (see Views below)
    - ❑ Make sure dynamic children are correct (too few/many)
    - ❑ Templates with complex justification

- ❑ Views
  - ❑ *Make sure the* `keyView` *is what you expect*
  - ❑ `GlobalBox` *can have negative values (e.g. button bar left in landscape)*
  - ❑ *Conversion between global and local coordinates (usually in viewClickScripts when converting units)*

- ❑ Multiple PC Cards
  - ❑ *Use 2 cards and change default store, check things get saved correctly*
  - ❑ *Check code for following bad code*
    - ❑ `GetStores()[1]` // or any value except for 0
    - ❑ `Length(GetStores())` // bad
  - ❑ *Make sure you are doing filing the right way*
    - ❑ Using filing button to do card routing
    - ❑ Filtering appropriately on store if applicable
  - ❑ *Endpoints to PC card do the right thing* (`MakeModemOption`)

Phase 3: Important Updates

- ❑ Screen Size Updates
  - ❑ *See if you can benefit from larger screen area*
  - ❑ *Check if your detail view and overview should be different sizes*
  - ❑ *Size: think Watch and Whiteboard! (min. and max. sizes)*
  - ❑ *Make base application view dragable* (`protoDragger` with `protoLargeCloseBox`)

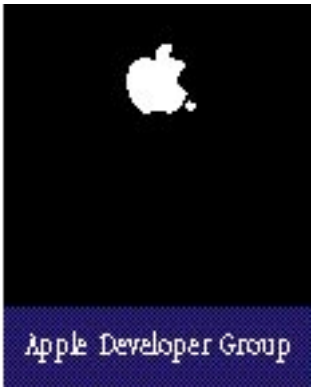
- ❑ Keyboard Support
  - ❑ *Add required key commands to your application*
    - ❑ some examples are filing, routing
  - ❑ *Support key navigation in overview (see DTS sample)*
  - ❑ *Decide on application specific key commands and implement*
  - ❑ *Conditionally display embedded keyboard* (`KeyboardConnected`)

- ❑ Grayscale I
  - ❑ *Add a gray Extras icon (need NTK 1.6.4)*
  - ❑ *Change obvious pictures like splash screens*
  - ❑ *Use* `FindShape` *instead of* `HitShape` *where appropriate*
  - ❑ *Change dithered items to use RGB patterns*

- ❑ Simple/Multi-user mode (eMate 300 only)
  - ❑ *Store and display user specific data* (`GenSoupName`)
  - ❑ *Hide filing features*
  - ❑ *Deal with user logins for changing user (UserConfig change)*

- ❑ Update old IR endpoints to use IrDA

NTJ



Newton Developer Programs

Apple offers three programs for Newton developers – the Newton Associates Program, the Newton Associates Plus Program and the Newton Partners Program. The Newton Associates Program is a low cost, self-help development program. The Newton Associates Plus Program provides for developers who need a limited amount of code-level support and options. The Newton Partners Program is designed for developers who need ujnlimited expert-level development. All programs provide focused Newton development information and discounts on development hardware, software, and tools – all of which can reduce your organization's development time and costs.

Newton Associates Program

This program is specially designed to provide low-cost, self-help development resources to Newton developers. Participants gain access to online technical information and receive monthly mailings of essential Newton development information. With the discounts that participants receive on everything from development hardware to training, many find that their annual fee is recouped in the first few months of membership.

Self-Help Technical Support
- Online technical information and developer forums.
- Access to Apple's technical Q&A reference library.
- Use of Apple's Third-Party Compatibility Test Lab.

Newton Developer Mailing
- *Newton Technology Journal – six issues per year.*
- *Newton Developer CD – four releases per year* which may include:
  - Newton Sample Code.
  - Newton Q & A's.
  - Newton System Software updates.
  - Marketing and business information.
- *Apple Directions – The Developer Business Report.*
- *Newton Platform News & Information.*

Savings on Hardware, Tools, and Training
- Discounts on development-related Apple hardware.
- Apple Newton development tool updates.
- Discounted rates on Apple's online service.
- US $100 Newton development training discount.

Other
- Developer Support Center Services.
- Developer conference invitations.
- *Apple Developer University Catalog.*
- *APDA Tools Catalog.*

Annual fees are $250.

Newton Partners Program

This expert-level development support program helps developers create products and services compatible with Newton products. Newton Partners receive all Newton Associates Program features, as well as unlimited programming-level development support via electronic mail, discounts on five additional Newton development units, and participation in select marketing opportunities.

With this program's focused approach to the delivery of Newton-specific information, the Newton Partners Program, more than ever, can help keep your projects on the fast track and reduce development costs.

Unlimited Expert Newton Programming-level Support
- One-to-one technical support via e-mail.

Apple Newton Hardware
- Discounts on five additional Newton development units.

Pre-release Hardware and Software
- Consideration as a test site for pre-release Newton products.

Marketing Activities
- Participation in select Apple-sponsored marketing and PR activities.

All Newton Associates Program Features:
- Developer Support Center Services.
- Self-help technical support.
- Newton Developer mailing.
- Savings on hardware, tools, and training.

Annual fees are $1500.

New: Newton Associates Plus Program

This new program now offers a new option to developers who need more than self-help information, but less than unlimited technical support. Developers receive all of the same self-help features of the Newton Associates Program, plus the option of submitting up to 10 development code-level questions to the Newton Systems Group DTS team via e-mail.

Newton Associates Plus Program Features:
- All of the features of the Newton Associates Program.
- Up to 10 code-level questions via e-mail.

Annual fees are $500.

For Information on All Apple Developer Programs
Call the Developer Support Center for information or an application. Developers outside the United States and Canada should contact their local Apple office for information about local programs.

Developer Support Center at (408) 974-4897
Apple Computer, Inc.
1 Infinite Loop, M/S 303-1P
Cupertino, CA 95014-6299

AppleLink: DEVSUPPORT

Internet: devsupport@applelink.apple.com