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 Shay 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 Shay 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 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 Shay platform, so it's always readily available. A common classroom scenario might have a student opening the Shay 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 Shay 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 Shay 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 Shay, 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 unregistering 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 stationery techniques in adding additional functionality to the Names app.