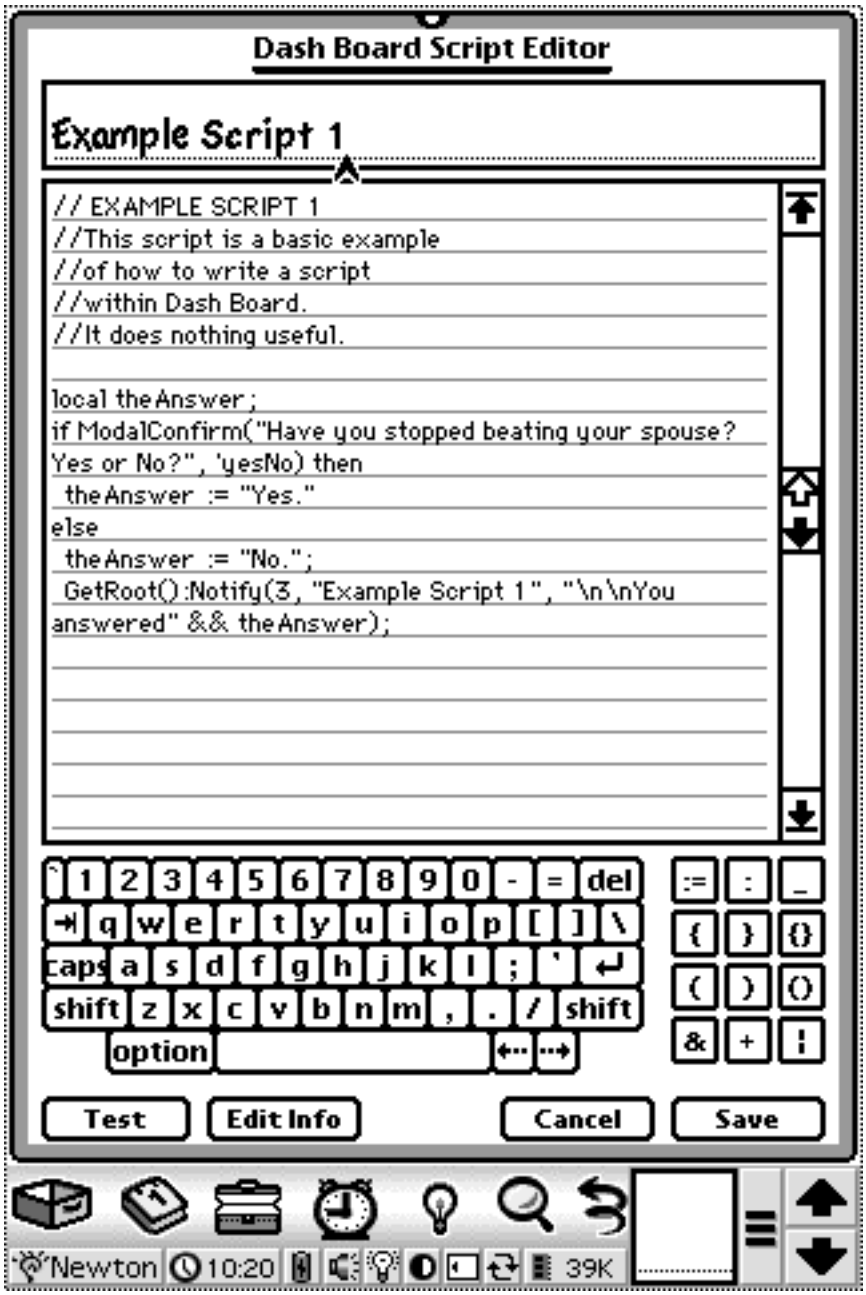
DASH BOARD SCRIPTER'S GUIDE

Version 1.5, March 1, 1999
Dash Board version at publication time: 1.5



ABOUT THIS BOOK

The ***Dash Board Scripter's Guide*** is intended to help you get the most out of Dash Board's scripting capabilities. In the sections that follow, you'll read about topics that will help you create Dash Board scripts. This book covers how Dash Board loads and executes scripts, what special methods are available to make your scripting tasks easier, and some limitations of Dash Board's "Import" function.

This book assumes you have a working knowledge of NewtonScript, the programming language used to create most Newton applications and the one that is used to create Dash Board scripts. If you are unfamiliar with NewtonScript, we recommend checking out the book ***Programming for the Newton Using Macintosh, 2nd Edition,*** by Julie McKeehan and Neil Rhodes (ISBN 0-12-484832-X). A volume for Windows users may also be available.

It is also assumed that you have a working knowledge of Dash Board and the Dash Board script editor. The mechanics of how to use the Dash Board script editor are covered in the ***Dash Board User's Guide***, available from the Five Speed Software web site (http://www.fivespeed.com).

Even if you are new to NewtonScript, you may find helpful information at the Dash Board scripting page (http://www.fivespeed.com/dashboard/scripting.html).

INTRO TO SCRIPTING WITH DASH BOARD

The Script Frame

A Dash Board **script** is simply a bit of NewtonScript source code. When a script is chosen from the Newton Menu or Letter Launcher, Dash Board looks up the soup entry, reads the source code, and compiles it into a function object.

Assuming there are no errors in compilation, Dash Board then creates a **script frame**. A script frame is a frame containing various slots, as described below. One of those slots contains the function object that resulted when the script code was compiled.

Dash Board then sends a message to the frame, to execute that function object.

Let's look at the script frame (items between the "<>" are English representations of the slot's contents):

a Dash Board script frame

```
scriptFrame :=
   {
     _proto: <reference to Dash Board itelf>,
     _parent: <reference to root (GetRoot())>,
     |scriptFunc:FiveSpeed|: <function - the script>
     |scriptName:FiveSpeed|: <string - the script name>,
     icon: NIL,
     stroke: NIL,
     parm: NIL,
     config: {text: "Dash Board does not support this
GestureLaunch feature"},
     scriptclass: NIL,
     word: NIL,
  };
```

In the above script frame, note that the **_proto** slot contains a reference to Dash Board itself. This allows scripts to access inherited methods and data from the Dash Board application itself, through proto inheritance. The methods you can use this way in your scripts are covered later in this book.

Next note that the **_parent** slot of the script frame contains a reference to the root view. This allows the script to access inherited methods and data from the root view. For example, scripts can call the Notify() root method like this:

using parent inheritance to access a root method

```
:Notify(3, "A Notify Alert", "I am inheriting the Notify
method from the root view.");
```

The next two slots are also significant. The **|scriptFunc:FiveSpeed|** slot contains the actual function object that Dash Board created when it compiled the script source code. When Dash Board runs a script, it creates the script frame depicted in Figure 1, then sends this frame the **|scriptFunc:FiveSpeed|()** message.

The |scriptName:FiveSpeed| slot contains a string which is the text name of the script. This is the name that appears in the Newton Menu, and in the list box in the Dash Board Prefs app's Scripts panel.

The **icon, stroke, parm, config, scriptclass, word** slots are all NIL. These slots are not used in Dash Board. They are provided solely for compatibility with imported GestureLaunch 3 scripts, which may expect to find these slots in the script frame.

While your script is executing, the value of **self** is a reference to the script frame. However, please note that when the script has finished executing, the script frame is disposed of--it is not kept in memory.

The Dash Board Scripts Soup

Dash Board stores its scripts in a soup. The soup entries are frames with the following format:

Dash Board script soup entry format

```
{
   name: <string - the text name of the script>,
   scriptText: <string - the script source code>,
   id: <symbol - the script's unique ID>,
   date: <integer - time in seconds when script was first
saved>,
   info: <string - informational description of script>
}
```

The name slot contains the text name of the script. The actual code is stored in the scriptText slot.

The date slot is filled automatically by Dash Board when you create a new script.

The `info` slot contains a string that should contain a brief informative description of the script, for the user.

The `id` slot contains a symbol that should be unique, so it should contain a variant of your unique developer's signature. For example, Five Speed Software's unique developer signature is "FiveSpeed", so a script written by Five Speed Software might have the id `'|SomeScript:FiveSpeed|`. The standard way to make a unique symbol is to append a colon (":") and your unique signature to the end of it.

If you do not have a unique developer's signature, and you think you might want to distribute your Dash Board scripts at some point, we encourage you to register with Apple Computer, Inc. for a registered developer signature.

We encourage you to do this as soon as possible, as Apple has been very lackluster in supporting the Newton after its discontinuation. We think that there is a danger that they will stop maintaining the Newton developer signature registry at some point. Although all Newton users and developers should express suitable outrage if that actually occurs, and insist that they continue maintaining the registry, it would seem prudent to register for a unique developer signature as soon as possible. (Also, be kind to the actual folks who process your registration--they don't have anything to do with Apple's poor treatment of Newton users and developers.)

As of this writing, the proper way to apply for a unique developer's signature is to send email to `NewtonDev@apple.com` and request one. You should include your name, company name, address, and top three choices for a unique signature (in case your first choice is already taken).

Note that you should keep your signature short; 8 characters or less is probably a good idea.

Apple can take a very long time to respond; please be patient.

NOTE: As of this writing, it’s possible that Apple may never respond, as they have almost completely stopped supporting the Newton platform in any way. If you are unable to get a unique signature, choose something that is probably unique. The name-signature combination only needs to be unique among Dash Board scripts, so problems are not likely.

USING INHERITED DASH BOARD METHODS

Dash Board 1.5 provides five methods which you can use in your scripts:

`DashBoardDoDialog()` - prompts the user for input and returns a value

`DashBoardGetScriptResult()` - executes another script, and returns the resulting value

`DashBoardDoSpecialItem()` - executes a Dash Board Special Item, and returns **`NIL`**

`DashBoardInsertText()` - inserts text at the caret, in a standard text entry view such as the Notes app, or in an editable TXview, such as those used by the Works app

`DashBoardFakePNButton()` – This command is for users of PowerNames, the Names enhancement from SilverWARE (http://www.silverware.com). It simulates a tap on the special PowerNames button bar icon.

As shown in the preceding chapter, your scripts can access these built in Dash Board methods through proto inheritance. You should call these methods simply by sending a message to the script frame, so all that is required is a trailing colon (i.e., sending the message to self, which is the script frame). Thus, the following code is sufficient:

Figure 3. - calling a method inherited from Dash Board

```
:DashBoardDoSpecialItem('cmdNewChecklist);
```

NOTE: You may "discover" other Dash Board methods which can be called from your scripts. Please do not do this. All of the methods designed to be called from Dash Board scripts begin with "DashBoard". Calling other internal Dash Board methods from scripts can crash Dash Board or have other undesirable consequences.

The DashBoardDoDialog() Method

This method displays a modal dialog, with a prompt, an input line for the user to enter an answer, a cancel button, and an enter button. The dialog can be customized depending on the arguments your script supplies. The arguments to this method are described below.

aScriptFrame`:DashBoardDoDialog(`*theValType*`,` *theMessage*`,` *theTitle*`,` *theAnswer*`,` *theCancelBtnTxt*`,` *theOkBtnText*`)`

aScriptFrame - the script frame. You can just use `:DashBoardDoDialog` without explicitly specifying this.

theValType - the value type you want returned. Supported values for this argument are `'string`, `'integer`, `'real`, and `'symbol`, which causes Dash Board to return the user input as a string, integer, real number, or symbol, respectively. If *theValType* is `NIL`, then the default return value type is string. If the text entered by the user cannot be coerced into the proper value, the dialog prompts the user to change the entered text.

theMessage - A string, which is the message that should be displayed in the dialog. If this argument is `NIL`, the dialog uses the default text, "Enter a value in the entry field provided."

theTitle - A string, which is the title displayed at the top of the dialog slip. If this argument is `NIL`, the dialog uses the default text, "Script Input".

theAnswer - A string, which is the default answer provided in the input line of the dialog when it is opened. If this argument is `NIL`, the dialog's input line is empty when it is first opened.

theCancelBtnTxt - A string, which is the text shown on the cancel button. If this argument is `NIL`, the default text is "Cancel". Note that if you provide a string that it too long, it will be truncated to fit. The maximum width of the button is slightly less than half the width of the dialog.

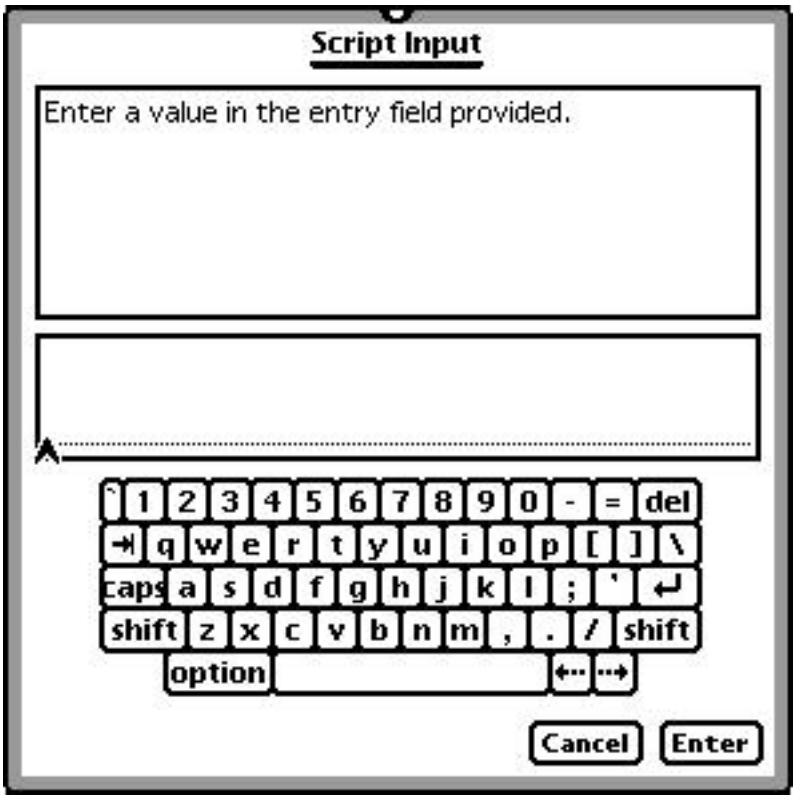
theOkBtnText - A string, which is the text shown on the enter button. If this argument is `NIL`, the default text is "Enter". Note that if you provide a string that it too long, it will be truncated to fit. The maximum width of the button is slightly less than half the width of the dialog.

If the user taps the enter button, `DashBoardDoDialog()` returns the value entered by the user, coerced to the type specified by the *theValType* argument. Otherwise, if the user taps the cancel button, this method returns `NIL`.

The following code produces the dialog shown below:

Figure 4. - a generic use of the DashBoardDoDialog() method

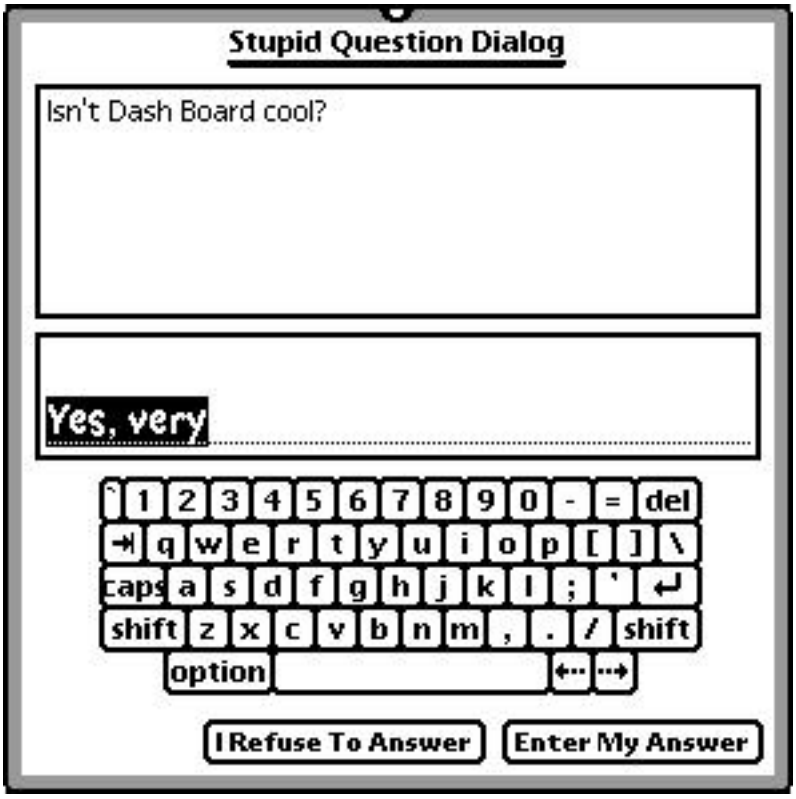
```
:DashBoardDoDialog(NIL, NIL, NIL, NIL, NIL, NIL);
```



Taking advantage of the customization options of this method, we can create a more interesting dialog:

Figure 5. - using DashBoardDoDialog() with parameters specified

```
:DashBoardDoDialog('string, "Isn't Dash Board cool?",
"Stupid Question Dialog", "Yes, very", "I Refuse To
Answer", "Enter My Answer");
```



The DashBoardDoDialog() method should be useful when you want to write a script that requires some user input; for example, a Reminder script might prompt the user for the number of minutes later that they want to be reminded, and then set a system alarm for that number of minutes in the future.

The DashBoardGetScriptResult() Method

This is a potentially very powerful method, that allows one script to have Dash Board execute another script and return the result to the first script, which can then use that data for whatever it needs to do.

This could be used to create a library of scripting tools; you could create simple component scripts to perform complex tasks you often use in your scripts, and then call these scripts from other scripts with a single line of code.

The DashBoardGetScriptResult() method can lookup a script by name, ID, or both. The method and its arguments are described below.

aScriptFrame`:DashBoardGetScriptResult(`*scriptName*`,` *scriptID*`)`

aScriptFrame - the script frame. You can just use `:DashBoardGetScriptResult` without explicitly specifying this.

scriptName - a string, that is the name of the script to be executed.

scriptID - A symbol, which is the ID of the script to be executed.

Either of the arguments to this method can be `NIL`, but not both. If only *scriptName* is specified, then Dash Board will execute the first script it finds that matches that name (which would generally be the one with the earliest creation date). If only *scriptID* is specified, then Dash Board will execute the first script it finds with the matching symbol in its `id` slot. Hopefully, that would ensure that it is the right script, as script IDs should be unique. However, just in case, you can supply both a name and an ID, which provides the most assurance that the script executed will be the right one.

This method returns the value returned by execution of the script requested. If the script cannot be found, this method returns the symbol `'|scriptNotFound:FiveSpeed|`.

example of calling a script from another script:

```
local theResult := :DashBoardGetScriptResult("Days Since
Last Backup", '|LastBackup:FiveSpeed|);
if IsInteger(theResult) and theResult > 7 then
   :Notify(3, "Backup Reminder", "Hey, you really should
back up soon!");
```

The DashBoardDoSpecialItem() Method

The DashBoardDoSpecialItem() method allows you to programmatically execute a Dash Board special item.

aScriptFrame`:DashBoardDoSpecialItem(`*specialItemSymbol*`)`

aScriptFrame - the script frame. You can just use `:DashBoardGetScriptResult` without explicitly specifying this.

specialItemSymbol - a symbol, which specifies the special item to execute. This symbol must be one from the list below.

This method always returns `NIL`.

example of the DashBoardDoSpecialItem() method:

```
:DashBoardDoSpecialItem('cmdNewChecklist);
```

Valid Special Item Symbols

NOTE: This list contains symbols for all valid symbols you can use with this method, for completeness; however, in some cases it is easier to just write the code yourself. For example to reboot the Newton, you could use the code `:DashBoardDoSpecialItem('cmdReboot);`, but it would be easier to simply use `Reboot().`

An asterisk (*) indicates that this command can be used with this method, but has no corresponding Special Item

symbol -	**corresponding Dash Board Special Item**
`'cmd61`	- -10061 Fix
`'cmdAbout`	- About Dash Board
`'cmdAlignPen`	- Align Pen
`'cmdBLOff`	- Backlight Off
`'cmdBLOn`	- Backlight On
`'cmdBLToggle`	- * toggles backlight state
`'cmdBtnsL`	- * same as choosing "Buttons Left" from Rotate menu
`'cmdBtnsR`	- * same as choosing "Buttons Right" from Rotate menu
`'cmdClearRecent`	- * same as choosing "Clear Recent Menu"
`'cmdConnect`	- Connect
`'cmdCtrlsL`	- * same as choosing "Controls Left" from Rotate menu
`'cmdCtrlsR`	- * same as choosing "Controls Right" from Rotate menu

'cmdGuestModeOff - HWR Guest Mode Off
'cmdGuestModeOn - HWR Guest Mode On
'cmdKbd1 - Keyboard
'cmdKbd2 - Keyboard (Numeric)
'cmdKbd3 - Keyboard (Phone)
'cmdKbd4 - Keyboard (Date)
'cmdLLHelp - Show LL Shortcuts
'cmdMIAAG - MI At A Glance
'cmdMICall - MI New Call
'cmdMIEvent - MI New Event
'cmdMIMtg - MI New Meeting
'cmdMITodo - MI New Todo
'cmdMute - Volume Mute
'cmdNewCall - New Call
'cmdNewChecklist - New Checklist
'cmdNewEvt - New Event
'cmdNewMtg - New Meeting
'cmdNewNote - New Note
'cmdNewOutline - New Outline
'cmdNewRec - New Recording
'cmdNewTodo - New Todo
'cmdPrefs - * opens Dash Board Prefs app
'cmdQuit - * quits Dash Board app
'cmdReboot - Reset Newton
'cmdRegister - Register Dash Board
'cmdRotate180 - Rotate 180°
'cmdRotate90L - Rotate 90° Left
'cmdRotate90R - Rotate 90° Right
'cmdSDS0 - Default Internal
'cmdSDS1 - Default Card 1
'cmdSDS2 - Default Card 2
'cmdSleep - Sleep
'cmdUmnute - Volume Unmute
'cmdWhoWhere - Set Owner/Worksite

The DashBoardInsertText() Method

This method inserts a text string at the caret. It works in any standard text entry view, such as the Notes app, and also works with views based on the protoTXView, used in the Works app.

aScriptFrame`:DashBoardInsertText(`*theString*`)`

aScriptFrame - the script frame. You can just use `:DashBoardInsertText` without explicitly specifying this.

theString - a text string. This is the text that will be inserted at the caret.

The return value of this method is undefined.

example of the DashBoardInsertText() method:

```
local myText :=
"John Doe
123 Main Street
New York, NY 12345";
:DashBoardInsertText(myText);
```

The example will insert John Doe's address in the current caret view.

The DashBoardFakePNButton() Method

This method simulates a tap on the special button bar icon that ships with PowerNames, from SilverWARE (http://www.silverware.com).

aScriptFrame`:DashBoardFakePNButton(`*whereTapped, howTapped*`)`

aScriptFrame - the script frame. You can just use `:DashBoardInsertText` without explicitly specifying this.

whereTapped – which part of the icon to simulate a tap on. Use `'icon` to simulate tapping on the icon, or `'text` to simulate tapping on the text beneath the icon.

howTapped – specifies whether the simulated tap is just a normal tap, or a tap-and-hold. Use `'tap` to simulate a normal tap, or `'hold` to simulate tapping and holding the pen down.

The return value of this method is undefined.

example of the DashBoardFakePNButton() method:

```
:DashBoardFakePNButton('icon, 'hold);
```

The example will simulate tapping and holding the pen down on the icon of the PowerNames button.

ABOUT IMPORTING SCRIPTS

Dash Board can import scripts written for GestureLaunch 3, from Innovative Computer Solutions (http://www.newts.com). Future versions of Dash Board may be able to import other types of scripts.

There are some important issues involving importing scripts that all Dash Board Scripters should be aware of.

Scripts, like any other software, may be copyrighted and their use may be subject to licensing limitations. While many scripts are publicly available and free of charge, some scripts are commercial software. You should ensure that you are licensed to use any scripts that you import into Dash Board. Copyrighted scripts which accompany other software packages may not be used without properly purchasing and obtaining a license from the vendor. Importing scripts which you are not legally entitled to use constitutes software piracy.

In particular, you need to be a registered user of GestureLaunch in order to use scripts written by ICS for use with that program. GestureLaunch for Dash Board is a Dash Board-compatible version of the program, that not only allows you to use all ICS scripts, but also provides powerful new scripting features that allow a great deal more flexibility than Dash Board alone. As of this writing, it is available from ICS for only $10.00. For more information, see the ICS web site at:

http://www.newts.com

Secondly, not all imported scripts will work in Dash Board. For example, some GestureLaunch 3 scripts rely on special features of GestureLaunch 3 which are not supported by Dash Board.

Dash Board tries to check imported scripts and determine if it uses methods or features of GestureLaunch 3 which Dash Board does not support. If it does, Dash Board will insert appropriate warnings in the comments preceding the script code, to let you know of possible problems.

While this is helpful, the process is not foolproof. Scripts may need to be modified to work with Dash Board. Some trial and error experimentation may be necessary.

ABOUT EXPORTING SCRIPTS

Dash Board can now export scripts, as well as importing them. There are two types of Export.

Notes (Text) Export
To export a script or scripts to the notes program, go to the Scripts panel of the Dash Board Prefs app. Select the script(s) you want to export, and tap the Export button. Choose "Export to Notes" from the popup menu.

This will create a new note in the Notes app for each exported script. The contents of the note will be the script code.

Package Export
If you have the PackMan utility installed, you can export your scripts to a Newton package. This allows you to easily and conveniently share your scripts with others. You can create Newton packages which will install your scripts on another user's Newton device, and easily distribute them on the internet via the web, FTP, etc. If you are serious about Dash Board scripting, you need PackMan.

PackMan is produced by Innovative Computer Solutions (http://www.newts.com).

Export Tips
It is a good idea to double check that your scripts have a suitable unique ID and accurate description in the Info field, if you wish to share them with others. To check, tap the Edit Info button in the Dash Board script editor.

FEEDBACK

We're always happy to hear suggestions for what we can do to improve your Dash Board scripting experience. Feel free to email us your ideas and suggestions at support@fivespeed.com.

Also, please check out the Dash Board Scripting page (http://www.fivespeed.com/dashboard/scripting.html) for more information about scripting with Dash Board. We will be periodically updating the site with new tips and information for Dash Board scripters.