




NTK Platform File Functions

 Apple Computer, Inc.
© 1997 Apple Computer, Inc.
All rights reserved.

No part of this publication or the software described in it may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except in the normal use of the software or to make a backup copy of the software and any documentation provided on CD-ROM. Printed in the United States of America.

The Apple logo is a registered trademark of Apple Computer, Inc. Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for licensed Newton platforms.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for printing or clerical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, AppleTalk, eMate, Espy, LaserWriter, the light bulb logo, Macintosh, MessagePad, Newton, Newton Connection Kit, and New York are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Geneva, NewtonScript, Newton Toolkit, and QuickDraw are trademarks of Apple Computer, Inc.

Acrobat, Adobe Illustrator, and PostScript are trademarks of Adobe Systems Incorporated, which may be registered in certain jurisdictions.

FrameMaker is a registered trademark of Frame Technology Corporation.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and/or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Microsoft is a registered trademark of Microsoft Corporation. Windows is a trademark of Microsoft Corporation. QuickView™ is licensed from Altura Software, Inc.

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, ADC will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to ADC.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD “AS IS,” AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction	1
Change History	1
Newton 2.1 Platform File	1
Newton 2.0 Platform File	1
Using Platform File Functions	2
Views	2
ClearSelectionHilites	2
ViewIsOpen	3
Extras Drawer	3
GetPartCursor	3
GetPartEntryData	4
LaunchPartEntry	4
SetExtrasInfo	5
AddExtraIcon	6
Miscellaneous	7
CardFileSafeRemoveLayout	8
GetAllFonts	8
GetCalendarMeetingTypes	8
GetCalendarMeetingTypeInfo	9
GetCommPortInfo	9
GetCorrectInfo	10
GetMaskedPixel	11
GetPrinterName	11
GetTransport	12
MissingImports	12
RegGlobalKeyboard	13
UnRegGlobalKeyboard	14
RegNamesRouteScript	14
UnRegNamesRouteScript	15
ShowBusyBox	15
StringToKeyCodes	16
SupplantSoupDef	16
Summary of Functions	17

Introduction

The platform files—stored in a folder named Platforms in the same folder as NTK—contain data tailored to different Newton products and a collection of Newton system definitions. The platform files also contain a number of utility functions and definitions for constants that reference them.

This document describes the additional functions provided by the NTK platform files that were not previously documented in *Newton Programmer's Reference* or *Newton Programmer's Guide: 2.1 OS Addendum*. However, those platform file functions that are included in these books are listed at the end of this document in the section "Summary of Functions," for completeness.

This document contains a change history, a brief description of how to call these platform file functions, and then the functions are described, grouped by topic. A summary of functions follows at the end of the document.

Change History

This section describes changes to the platform file functions since the last release of this document (November 1995).

Newton 2.1 Platform File

A new platform file, "Newton 2.1," (version 1.2b1) has been created to support the Newton 2.1 platform. This platform file contains the same functions as the 2.0 platform file, with these additional changes:

New functions:

```
GetMaskedPixel
StringToKeyCodes
```

Newton 2.0 Platform File

The "Newton 2.0" platform file (version 1.2b1) has been changed as follows:

New functions:

```
DragAndDropLtd
GetAllDialinNetworks
GetAllFonts
GetCommPortInfo
GetCorrectInfo
GetDialinNetwork
GetLocAccessNums
GetPartEntryData
GetTransport
```

RegDialInNetwork
UnregDialInNetwork

Obsolete function:

SimpleTextHeight

Using Platform File Functions

The constant that represents a function is the function name with the prefix `k` and the suffix `Func` (that is, `kfunctionnameFunc`).

The platform file functions are available at compile time; you can make them available at run time by incorporating them into your application in the following way. Call the function with the `NewtonScript call` syntax or the `Apply` function. This strategy saves space and time, because it does not require a slot in the base view and avoids inheritance lookup; it also works in code that doesn't have access to your base view, such as the application part `RemoveScript` function.

Here is an example of using the `call` syntax to call a platform file function:

```
call kNewInfoFunc with (arg1, arg2);
```

Here is an example of using the `Apply` function to call a platform file function:

```
Apply(kNewInfoFunc, [arg1, arg2]);
```

Views

These functions relate to views.

ClearSelectionHilites

```
ClearSelectionHilites(theView) // call kClearSelectionHilitesFunc
```

This function removes selection highlights from a view.

theView The view of class `clEditView` or `clParagraphView` from which you want to remove selection highlights.

return value Undefined; do not rely on it.

DISCUSSION

For views of the `clParagraphView` class, this function unhighlights any selected text. For views of the `clEditView` class, this function removes the thick gray rectangles used to indicate selections. For other view classes the behavior is undefined.

If you specify a view of the `clParagraphView` class that is a child of a view of the `clEditView` class, the behavior is also undefined.

Note

Do not confuse this function with the `Hilite` or `HiliteUnique` view methods. The `ClearSelectionHilites` function does not affect the highlight state of a view. ♦

ViewsOpen

```
ViewIsOpen(view) // call kViewIsOpenFunc
```

Returns `true` if the view is open.

<i>view</i>	The view you wish to check.
return value	True if the view is open; <code>nil</code> if it is not.

DISCUSSION

Note that a view can be open but not visible (if it is hidden).

This function is a better way to check if a view is open, rather than checking if the `viewController` slot is non-`nil`.

Extras Drawer

These functions operate on the Extras Drawer.

GetPartCursor

```
extrasDrawer:GetPartCursor(packageName, store, folderSym) // call  
kGetPartCursorFunc
```

Returns a cursor for entries corresponding to parts (icons) displayed in the Extras Drawer.

<i>packageName</i>	Specify a string naming a package, or <code>nil</code> . If you specify a package name, the cursor returns parts only from that package. To return parts from all packages, specify <code>nil</code> .
<i>store</i>	Specify a store object or <code>nil</code> . If you specify a store object, the cursor returns parts only from that store. To return parts from all stores, specify <code>nil</code> .
<i>folderSym</i>	Specify a symbol identifying a folder, or <code>nil</code> . If you specify a folder symbol, the cursor returns parts only filed within that Extras Drawer folder. To return parts from all folders, specify the symbol <code>'_all'</code> . To return parts from the unfiled folder, specify <code>nil</code> .
return value	A cursor for entries corresponding to parts (icons) displayed in the Extras Drawer.

DISCUSSION

The structure of the entries returned by the cursor is subject to change. Entries should be accessed only by using the functions `GetPartEntryData`, `LaunchPartEntry`, and `SetExtrasInfo`. Do not directly change the entries returned by `GetPartCursor`.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

GetPartEntryData

extrasDrawer:GetPartEntryData(*entry*) // call kGetPartEntryDataFunc

Returns a frame containing information about an Extras Drawer part entry.

entry An entry obtained from a part cursor (by using `GetPartCursor`).

return value A frame containing information about an Extras Drawer part entry.

DISCUSSION

The frame returned has the following slots.

Slot descriptions

<i>icon</i>	A bitmap frame (of the kind returned by <code>GetPictAsBits</code>) containing the bitmap for the part icon displayed in the Extras Drawer.
<i>iconPro</i>	A frame with two slots, <i>unhilited</i> and <i>hilited</i> , that contain <code>pix</code> families for normal and highlighted versions of the icon. The highlighted version of the icon is shown when the icon is selected.
<i>text</i>	A string that is the text shown under the part icon.
<i>labels</i>	A symbol identifying the Extras Drawer folder in which the part is filed. If this slot is <code>nil</code> , the part is unfiled.
<i>appSymbol</i>	A symbol identifying the application, if the part frame has an <code>app</code> slot.
<i>packageName</i>	A string that is the name of the package that contains the part.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

LaunchPartEntry

extrasDrawer:LaunchPartEntry(*entry*) // call kLaunchPartEntryFunc

Launches the specified part. The operation is the equivalent of the user tapping the part icon in the Extras Drawer.

entry An entry obtained from a part cursor (by using `GetPartCursor`).

return value Returns a non-`nil` value if the Extras Drawer would have closed itself after the icon was tapped. Returns `nil` if the Extras Drawer would have stayed open after the icon was tapped.

DISCUSSION

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

SetExtrasInfo

```
extrasDrawer:SetExtrasInfo(paramFrame, newInfo) // call kSetExtrasInfoFunc
```

Changes the extras drawer information for the specified Extras Drawer icon.

<i>paramFrame</i>	A frame identifying the icon whose Extras Drawer information you want to change. This frame can have the following slots:
<i>appSymbol</i>	Required. A symbol identifying the application that the icon represents.
<i>store</i>	Optional. A store object identifying the store on which the icon resides.
<i>packageName</i>	Optional. A string naming the package to which the icon belongs.
	Alternately, you can specify just an <i>appSymbol</i> for the <i>paramFrame</i> parameter, or you can specify an entry obtained from a part cursor (by using <i>GetPartCursor</i>). Note that specifying just an <i>appSymbol</i> is allowed for compatibility reasons; it may not be supported in future versions of the system software.
<i>newInfo</i>	A new information frame for the icon represented by <i>paramFrame</i> . The slots in this frame are described below. If you don't specify a particular slot (or specify <i>nil</i>), the value of that slot is not changed.
return value	The information frame that was in effect before this call. If the icon isn't found, this function returns <i>nil</i> .

DISCUSSION

You can read and modify the following slots in the *newInfo* frame.

Slot descriptions

<code>icon</code>	A bitmap frame (of the kind returned by <code>GetPictAsBits</code>) containing the bitmap for the part icon displayed in the Extras Drawer.
<code>iconPro</code>	A frame with two slots, <code>unhilited</code> and <code>hilited</code> , that contain <code>pix</code> families for normal and highlighted versions of the icon. The highlighted version of the icon is shown when the icon is selected.
<code>text</code>	A string that is the text shown under the part icon.
<code>labels</code>	A symbol identifying the Extras Drawer folder in which to file the icon. Do not specify <code>nil</code> .
<code>soupNames</code>	An array of strings that are the names of soups to be associated with this icon. This slot applies to soup icons only.
<code>ownerApp</code>	The <code>appSymbol</code> of the application that owns the soups. This slot applies to soup icons only.

Note

This function exists in both the Newton 2.x platform files and the MessagePad platform file, however it is implemented differently on each platform. This description applies only to the Newton 2.x platform. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

AddExtraIcon

```
extrasDrawer:AddExtraIcon(extraType, paramFrame, packageName, store) // call  
kAddExtraIconFunc
```

Adds an icon to the Extras Drawer.

<code>extraType</code>	Specify a symbol identifying the type of icon to add. You can specify <code>'soupEntry</code> to add a soup icon, or <code>'scriptEntry</code> to add a script icon.
<code>paramFrame</code>	Specify a frame containing information to be used in creating the icon. See the descriptions of slots below.
<code>packageName</code>	Specify a string naming a package with which this icon should be associated. If this package is removed from the Newton device, the icon you added will also be removed. For an icon of type <code>'soupEntry</code> , you should specify a package name different from your application. This prevents your soup icon from being removed if the application is on a card and the card is removed. Never pass <code>nil</code> for this argument.
<code>store</code>	Specify a store object on which the part entry should reside. A value of <code>nil</code> specifies the default store.
return value	Returns a frame that is the entry added to the Extras Drawer.

DISCUSSION

You can use this function to add an icon that represents several soups created by your application, for example. You can also add an icon that simply executes a function object when tapped.

This function does not check if your icon already exists before adding it. You must check to be sure it doesn't already exist.

The slots in the *paramFrame* frame vary depending on the value of *extraType*. The *paramFrame* frames for both types of icons share these slots:

Slot descriptions

<code>text</code>	Required. A string that is the text shown under the icon.
<code>app</code>	Recommended. A unique symbol used by <code>SetExtrasInfo</code> to find the icon.
<code>labels</code>	Optional. A symbol identifying the Extras Drawer folder in which to file the icon. Do not specify <code>nil</code> .

In addition, the *paramFrame* of soup icons should have these slots:

<code>ownerApp</code>	Optional. The <code>appSymbol</code> of the application that owns the soups. This slot is used for the soupervisor mechanism.
<code>soupNames</code>	Optional. An array of strings that are the names of soups to be associated with this icon.

The *paramFrame* of script icons should have these additional slots:

<code>icon</code>	Recommended. A bitmap frame (of the kind returned by <code>GetPictAsBits</code>) containing the bitmap for the icon displayed in the Extras Drawer. The bitmap should be 32 by 32 pixels.
<code>iconPro</code>	A frame with two slots, <code>unhilited</code> and <code>hilited</code> , that contain pix families for normal and highlighted versions of the icon. The highlighted version of the icon is shown when the icon is selected.
<code>tapAction</code>	Optional. A function object that is called if the user taps the icon. This function is passed no parameters. This function is stored in a soup, so you should keep it as small as possible.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

Miscellaneous

This section describes other functions.

CardFileSafeRemoveLayout

`CardFileSafeRemoveLayout(layout) // call kCardFileSafeRemoveLayoutFunc`

Safely removes a cardfile layout from the Names application. (Cardfile layouts appear on the Show menu in the Names application.)

layout A symbol identifying the cardfile layout you want to remove. This is the same symbol you passed to the cardfile method `AddLayout` to add the layout.

return value Undefined; do not rely on it.

DISCUSSION

You should use this function instead of the cardfile method `RemoveLayout` to remove a cardfile layout. Using this function avoids “Reinsert the card” warnings.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

GetAllFonts

`GetAllFonts() // call kGetAllFontsFunc`

Returns an array of fonts installed in the system.

return value An array of font description frames.

DISCUSSION**Note**

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

GetCalendarMeetingTypes

`GetCalendarMeetingTypes() // call kGetCalendarMeetingTypesFunc`

Returns an array of symbols that identify all of the meeting types registered with the Dates application (calendar).

return value The array includes both built-in meeting types and any new types registered through the use of `RegMeetingType`.

DISCUSSION**Note**

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

GetCalendarMeetingTypeInfo

```
GetCalendarMeetingTypeInfo(typeSymbol) // call
kGetCalendarMeetingTypeInfoFunc
```

Returns a frame of information about a particular meeting type.

<i>typeSymbol</i>	A symbol identifying a meeting type registered with the Dates application. Specify one of the symbols returned by <code>GetCalendarMeetingTypes</code> .
return value	A frame of information about a particular meeting type; see below. If the specified type is not found, then <code>nil</code> is returned.

DISCUSSION

The returned frame contains the following slots.

Slot descriptions

<code>label</code>	A string that is the text displayed in the New menu for this meeting type.
<code>icon</code>	A bitmap frame (of the kind returned by <code>GetPictAsBits</code>) containing the bitmap that is displayed in the New menu for this meeting type.
<code>smallIcon</code>	A bitmap frame containing the bitmap that is displayed in the meeting slip for this meeting type.
<code>shape</code>	A shape object containing the <code>icon</code> bitmap.
<code>memory</code>	A symbol under which the most recently used meeting title strings are stored. (These are stored and accessed using the functions <code>AddMemoryItem</code> and <code>GetMemoryItems</code> .)

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

GetCommPortInfo

```
GetCommPortInfo() // call kGetCommPortInfoFunc
```

Returns an array of communication port description frames.

return value	An array of communication port description frames.
--------------	--

DISCUSSION

Use this method to get a list of the available communication ports on a Newton device. `GetCommPortInfo` returns an array of frames with the following slots.

Slot descriptions

<code>item</code>	A string that is the name of the communications port; examples include: “Top PC Card,” “Infrared,” and “Serial.”
<code>hardwareLoc</code>	A four-character string identifying the serial port location. This string is suitable for use in the serial communication tool’s <code>kCMOSerialHWChipLoc</code> option.
<code>hardwareInfo</code>	A frame containing information about the device connected to this particular communications port. Currently, this slot is used only for PCMCIA communication cards. The slots in this frame are listed in Table 1-1. This frame may contain other slots, but you should not rely on their existence. The <code>hardwareInfo</code> slot is <code>nil</code> if a communications device is not connected to this particular port.

Table 1-1 `hardwareInfo` frame slots

Slot name	Description
<code>cisManufacturerName</code>	A string. Manufacturer name from the <code>TPLL1_INFO</code> field of the card’s <code>CISTPL_VERS_1</code> tuple.
<code>cisProductName</code>	A string. Product name from the <code>TPLL1_INFO</code> field of the card’s <code>CISTPL_VERS_1</code> tuple.
<code>cisProductInfo0</code>	A string. Additional product info from the <code>TPLL1_INFO</code> field of the card’s <code>CISTPL_VERS_1</code> tuple.
<code>cisProductInfo1</code>	A string. Additional product info from the <code>TPLL1_INFO</code> field of the card’s <code>CISTPL_VERS_1</code> tuple.
<code>cisManufacturerId</code>	An integer. Manufacturer information from the <code>TPLMID_MANF</code> field of the card’s <code>CISTPL_MANFID</code> tuple.
<code>cisManufacturerIdInfo</code>	An integer. Manufacturer information from the <code>TPLMID_CARD</code> field of the card’s <code>CISTPL_MANFID</code> tuple.

Note that this function returns an array that is suitable for use in a picker list, however, you might not want to include all the items in the picker. To remove items from the list, search on the `hardwareLoc` slot for the items you want to remove. The `hardwareLoc` slot is the one slot that is guaranteed to remain constant over time.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

GetCorrectInfo

```
GetCorrectInfo() // call kGetCorrectInfoFunc
```

Returns the system correction information frame, which contains correction information for recently-recognized words.

return value The system correction information frame.

DISCUSSION

For more information on the correction information frame, see the description of the `correctInfo` frame and `protoCorrectInfo` in the *Newton Programmer's Reference*, and see Chapter 10, "Recognition: Advanced Topics" in *Newton Programmers Guide*.

Note that the function `GetCorrectInfo` was incorrectly documented in *Newton Programmer's Reference*; it does not exist in the Newton 2.0 ROM. On Newton 2.0 systems, it is accessible only by using the `kGetCorrectInfoFunc` platform file function.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

GetMaskedPixel

```
GetMaskedPixel(x, y, pixFamily) // call kGetMaskedPixelFunc
```

Retrieves the value of a specific pixel within a pix family, taking into account its mask.

<i>x</i>	The x coordinate of the point to be tested, in local (view) coordinates.
<i>y</i>	The y coordinate of the point to be tested, in local (view) coordinates.
<i>pixFamily</i>	The pix family to test.
return value	An integer; -1 if the (x,y) pixel location lies outside the bounds of the pix family or if the mask is off at this position, otherwise the integer value of the specified pixel is returned (see below).

DISCUSSION

This function is similar to the existing `PtInPicture` function.

The value returned for a pixel that is actually within the pix family's bounds (and at an on position in the mask) depends on the bit depth of the pix family image. For images with a bit depth of 1, 2, 4, and 8, the pixel will be an index in the range (0, $2^{\text{bit depth}} - 1$). For example, if the image has a bit depth of 4, the value returned by the function would range from 0 to 15. If the image has a bit depth of 16 or 32, the pixels will have a direct format, and the function will return the direct RGB pixel value.

Note

This function exists only in the Newton 2.1 platform file and works only on Newton 2.1 units. ♦

GetPrinterName

```
GetPrinterName(printerFrame) // call kGetPrinterNameFunc
```

Retrieves the name of the printer, given a printer frame object.

<i>printerFrame</i>	A printer frame object. The only valid method for obtaining a printer frame object is to retrieve it from the system
---------------------	--

userConfiguration data with the `GetUserConfig` function. Do not try to construct the slots of this frame yourself because different types of printer drivers require different slots.

return value Returns a string representing the name of the printer associated with *printerFrame*.

DISCUSSION

Here is an example of some code that retrieves the name of the current printer:

```
printerFrame := GetUserConfig('currentPrinter');
thePrinterName := call kGetPrinterNameFunc with (printerFrame);
```

GetTransport

`GetTransport(transportSymbol)` // call `kGetTransportFunc`

Returns a transport object, given a transport symbol.

transportSymbol A symbol identifying a transport. This is the value of the transport's `appSymbol` slot.

return value Returns the transport object (a frame).

DISCUSSION

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

MissingImports

`MissingImports(pkgRef)` // call `kMissingImportsFunc`

Returns an array of frames describing units used by a package that are not currently available.

pkgRef A package reference identifying the package for which this function returns information. You can obtain a package reference by using the functions `ObjectPkgRef` or `GetPkgRef`.

return value An array of frames describing units used by a package that are not currently available; see below. If no units are missing, `nil` is returned.

DISCUSSION

In the array that is returned, each frame has the following slots.

Slot descriptions

<code>name</code>	A symbol identifying the unit.
<code>major</code>	The unit major version number.
<code>minor</code>	The unit minor version number.

Refer to the Newton DTS sample code for details on how to use this function and the unit import and export mechanism.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

RegGlobalKeyboard

`RegGlobalKeyboard(kbdSymbol, kbdTemplate) // call kRegGlobalKeyboardFunc`

Replaces the alphanumeric keyboard in the system with a custom keyboard.

<i>kbdSymbol</i>	A unique symbol identifying the keyboard view. You should append your developer signature to ensure that this symbol is unique.
<i>kbdTemplate</i>	A view template that is the new keyboard you are registering. This template must include a slot named <code>preAllocatedContext</code> whose value is set to the symbol <code>'alphaKeyboard</code> . This template must also include a slot named <code>userName</code> whose value is a string naming the keyboard. This is the keyboard name that the user sees in keyboard pickers.
return value	The keyboard view, if it was successfully instantiated and installed in the system; otherwise, this function returns <code>nil</code> .

DISCUSSION

The keyboard view will be created as a child of the root view. Anytime the alphanumeric keyboard would have been opened, the custom keyboard will be opened instead.

In your custom keyboard, you might want to include a button that opens the Personal Word List. That's the user's dictionary of personal words. The standard alphanumeric keyboard includes a button that opens the Personal Word List. To do this, include your own button in your keyboard template (don't use the bitmap from the ROM), and if the user taps it, execute code like this:

```
If GetRoot().reviewdict then
  begin
    GetRoot().reviewdict:Open();
    base:Close();
  end;
```

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

UnRegGlobalKeyboard

```
UnRegGlobalKeyboard(kbdSymbol) // call kUnRegGlobalKeyboardFunc
```

Unregisters a custom keyboard that you registered with `RegGlobalKeyboard`.

kbdSymbol A symbol identifying the keyboard to unregister.

return value Undefined; do not rely on it.

DISCUSSION

This function restores the original keyboard that you replaced in the system. It closes the custom keyboard, if it is open when this function is called, but this function does not open the original keyboard that was replaced.

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

RegNamesRouteScript

```
RegNamesRouteScript(symbol, routeScriptFrame) // call
kRegNamesRouteScriptFunc
```

Adds an application-defined action to the Action picker in the Names application.

symbol A unique symbol identifying the action you are adding. You should append your developer signature to ensure that this symbol is unique.

routeScriptFrame A frame describing the routing action, as described in the chapter “Routing Interface” in *Newton Programmer’s Guide*. A summary of this frame is described below.

return value Undefined; do not rely on it.

DISCUSSION

Here’s a summary of the *routeScriptFrame* parameter:

```
{
title: string,           // string name of picker item
icon: bitmap object,    // icon for picker item
RouteScript: symbol,    // func called if this action chosen
appSymbol: symbol,      // symbol for context of RouteScript
GetTitle: function      // supplied instead of title slot
...                       // other slots used by your app
}
```

Here’s an example of using the `RegNamesRouteScript` function:

```
call kRegNamesRouteScriptFunc with ('|EntryDumper:PIEDTS|,
  { GetTitle: func(target) begin
    if GetTargetCursor(target, nil):entry() then
      "Dump entry";
    else
      nil;// no selections, so don't show in list
    end,
  icon: nil,
```

```

RouteScript: func(target, targetView) begin
    local curs:=GetTargetCursor(target, nil);
    local e := curs:Entry();
    while e do begin
        print(e);
        e:=curs:Next();
    end;
end
});

```

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

UnRegNamesRouteScript

UnRegNamesRouteScript(*symbol*) // call kUnRegNamesRouteScriptFunc

Removes an application-defined action from the Action picker in the Names application.

symbol A symbol identifying the action you are removing.

return value Undefined; do not rely on it.

DISCUSSION

This function only removes actions added by RegNameRouteScript.

Here's an example of using the UnRegNamesRouteScript function:

```
call kUnRegNamesRouteScriptFunc with ('|EntryDumper:PIEDTS|);
```

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

ShowBusyBox

ShowBusyBox(*showIt*) // call kShowBusyBoxFunc

Shows or hides the system busy icon.

showIt A Boolean that specifies whether to show or hide the system busy icon. Specify *true* to show the busy icon until control returns to the system. Specify *nil* to hide the busy icon for the rest of the current iteration of the system event loop.

return value Undefined; do not rely on it.

DISCUSSION**Note**

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. Note that this function is implemented in ROM on Newton 2.1 units, so you can call it directly if your application runs only on the Newton 2.1 OS. ♦

StringToKeyCodes

```
StringToKeyCodes(string) // call kStringToKeyCodesFunc
```

Translates a string to its corresponding key codes.

<i>string</i>	The string to translate into key codes.
return value	An array of numeric key codes corresponding to the characters in <i>string</i> .

DISCUSSION

`StringToKeyCodes` returns an array of key codes suitable for passing to the function `HandleKeyEvents`.

Note

This function exists only in the Newton 2.1 platform file and works only on Newton 2.1 units. ♦

SupplantSoupDef

```
SupplantSoupDef(soup, soupDef) // call kSupplantSoupDefFunc
```

Installs the specified soup definition in the specified single soup.

<i>soup</i>	The soup on which this method operates. This value must be a soup, not a union soup.
<i>soupDef</i>	The soup definition frame this method installs.
return value	Undefined; do not rely on it.

DISCUSSION**IMPORTANT**

Changing a soup definition frame is not recommended. Use this function only if you know that what you are attempting to do will not cause errors or undesirable side effects. ♦

The `SupplantSoupDef` function works on single soups only, not on union soups. You can use the union soup method `GetSoupList` to retrieve a list of the member soups that are currently available to a specified union.

You can use the `SupplantSoupDef` function to

- Change the user-visible information for a specified soup. For example, you could use this function to change the string that the Extras drawer displays as the soup's name.
- Add a soup definition frame to a soup that lacks one. For example, soups created by system software prior to version 2.0 do not have soup definition frames.
- Replace the soup definition frame in a soup that already has one. Note that this may cause inconsistencies with other soups in the union that can lead to unstable behavior.

Note

This function does not change the soup definition currently registered with the system—it changes only the local copy of the definition that is held by a soup created from it. To change a soup definition registered with the system, you must replace it completely. To do so, first call the `UnRegUnionSoup` function to unregister the current soup definition, and then call the `RegUnionSoup` function to register the new soup definition. ♦

Because most of the information in a soup definition frame is used only when the system creates a new soup, the appropriate usage of the `SupplantSoupDef` function is very limited. For example, although you can use this method to change the indexes that a soup definition specifies for new soups, the actual indexes in existing soups are not updated by this method. Soups created subsequently from this definition may not have the same complement of indexes as other soups in their union, which may cause operations on the union soup to fail. Exercise extreme caution when using this method for any purpose.

The following code fragment provides an example of the proper use of this function.

```
// unregister old definition
UnRegUnionSoup("mySoup:mySig", '|MyApp:MySig|');
// register new version of soup definition
// assume myNewSoupDef is valid
local uSoup := RegUnionSoup('|MyApp:MySig|, myNewSoupDef);
// update existing soups
foreach member in uSoup:GetSoupList() do
    begin
        call kSupplantSoupDefFunc with (member, newDef);
        // perform other housekeeping like adding or removing indexes
    end;
```

Note

This function exists only in the Newton 2.x platform files and works only on Newton 2.x units. ♦

Summary of Functions

This section contains a summary of all the functions provided by the platform files, including those functions documented previously in *Newton Programmer's Reference* and in the latest *Newton Programmer's Guide: 2.1 OS Addendum*.

Views

`ClearSelectionHilites(theView)`
`ViewIsOpen(view)`

Extras Drawer

`GetPartCursor(packageName, store, folderSym)` // in 2.1 ROM
`LaunchPartEntry(entry)` // in 2.1 ROM
`SetExtrasInfo(paramFrame, newInfo)` // in 2.1 ROM
`AddExtraIcon(extraType, paramFrame, packageName, store)` // in 2.1 ROM

Miscellaneous

`CardFileSafeRemoveLayout(layout)`
`GetAllFonts()` // in 2.1 ROM

```

GetCalendarMeetingTypes()
GetCalendarMeetingTypeInfo(typeSymbol)
GetCommPortInfo()
GetCorrectInfo() // in 2.1 ROM
GetMaskedPixel(x, y, pixFamily)
GetPrinterName(printerFrame)
GetTransport(transportSymbol)
MissingImports(pkgRef)
RegGlobalKeyboard(kbdSymbol, kbdTemplate)
UnRegGlobalKeyboard(kbdSymbol)
RegNamesRouteScript(symbol, routeScriptFrame)
UnRegNamesRouteScript(symbol)
ShowBusyBox(showIt) // in 2.1 ROM
StringToKeyCodes(string)
SupplantSoupDef(soup, soupDef)

```

Functions Documented in *Newton Programmer's Reference*

```

AddLocale(theLocaleBundle) // in 2.1 ROM
FindLocale(titleString) // in 2.1 ROM
GetLanguageEnvironment()
QuietSendAll(transportSym)
RegEmailSystem(classSymbol, name, internet)
RegPagerType(classSymbol, name)
RegPhoneType(classSymbol, name, char)
RemoveLocale(locSymbol) // in 2.1 ROM
UnregEmailSystem(classSymbol)
UnregPagerType(classSymbol)
UnregPhoneType(classSymbol)

```

Functions Documented in *Newton Programmer's Guide: 2.1 OS Addendum*

```

DragAndDropLtd(unit, dragBounds, limitBounds, copy, dragInfo)
GetAllDialInNetworks()
GetDialInNetwork(networkSym)
GetLocAccessNums(entry, which)
GetPartEntryData(entry) // in 2.1 ROM
RegDialInNetwork(networkSym, networkFrame)
UnregDialInNetwork(networkSym)

```

Obsolete Functions

The following functions are supplied in the MessagePad platform file, but are obsolete in the Newton 2.x platform files. They are still supported in Newton 2.x under different names (“Deprecated” is appended), but it is recommended that you do not use them. In most cases, these platform file functions have been replaced by ROM functions in Newton 2.x, or in some cases, they no longer apply to Newton 2.x.

MessagePad name	Newton 2.x name (obsolete)
AddAlarm	AddAlarmDeprecated
CloseRemoteControl	CloseRemoteControlDeprecated
FlushUserConfig	FlushUserConfigDeprecated
GetAlarm	GetAlarmDeprecated
GetAppAlarmKeys	GetAppAlarmKeysDeprecated
GetDefaultStore	GetDefaultStoreDeprecated
GetUserConfig	GetUserConfigDeprecated

MessagePad name

OpenRemoteControl
 PtInBitMap
 RegFindApps
 RegFormulas
 RegisterCardSoup
 RegPrefs
 RemoveAlarm
 RemoveAppAlarms
 Send
 SendRemoteControlCode
 SetDefaultStore
 SetExtrasInfo
 SetUserConfig
 SimpleTextHeight
 UnionSoupIsNull
 UnRegFindApps
 UnRegFormulas
 UnRegisterCardSoup
 UnRegPrefs

Newton 2.x name (obsolete)

OpenRemoteControlDeprecated
 PtInBitMapDeprecated
 RegFindAppsDeprecated
 RegFormulasDeprecated
 RegisterCardSoupDeprecated
 RegPrefsDeprecated
 RemoveAlarmDeprecated
 RemoveAppAlarmsDeprecated
 SendDeprecated
 SendRemoteControlCodeDeprecated
 SetDefaultStoreDeprecated
 SetExtrasInfoDeprecated
 SetUserConfigDeprecated
 SimpleTextHeightDeprecated
 UnionSoupIsNullDeprecated
 UnRegFindAppsDeprecated
 UnRegFormulasDeprecated
 UnRegisterCardSoupDeprecated
 UnRegPrefsDeprecated

