WRIST PDA®
with PALM OS®

Software Development Kit

For Fossil and Abacus Brands

Version 1.3 12/20/2004

Table Of Contents

1 PART 1 – DEVELOPMENT GUIDE 1
1.1 Overview 1
1.2 Development Environment 1
1.2.1 Requirements 1
1.2.2 Installation of the Software Developer's Kit 2
1.2.3 Testing on the Wrist PDA Device 2
1.2.4 Testing on the Wrist PDA Emulator 2
1.2.5 Testing on Palm's Emulator or a Palm OS Device 3
1.3 Creating Wrist PDA Resources and Code 3
1.3.1 Using Large Fonts 3
1.3.2 Using the Back Key and Rocker Switch 4
1.3.3 Example Programs 5
2 PART 2 – CREATING A WATCH APPLICATION 9
2.1 Overview 9
2.2 Requirements To Be Recognized as a Watch App 9
2.3 Required Launch Codes 10
2.4 Optional Launch Code 10
2.5 Managing Global Data 10
2.6 Display and System State Must Be Saved/Restored 12
2.7 Alarm and Low Battery Notifications 12
3 PART 3 – WRIST PDA API REFERENCE 13
3.1 Fossil fonts 13
3.1.1 Font ID Table 13
3.1.2 FossilIsFossilFont() 13
3.1.3 FossilIsSystemFont() 14
3.1.4 FossilLargeFontID() 14
3.1.5 FossilNormalFontID() 14
3.2 Forms 15

3.2.1 FossilLargerFont Gadget 15
3.2.2 FossilFrmIsLargeFont() 16
3.2.3 FossilFrmEnlargeObject() 16
3.3 Menus & Command Bar 17
3.3.1 FossilMenuLargeFontGet() 17
3.3.2 FossilMenuLargeFontSet() 17
3.3.3 FossilMenuCmdBarIconsSet() 17
3.3.4 FossilMenuCmdBarIconsGet() 18
3.3.5 FossilMenuSetActiveMenuRscID() 18
3.4 Keys & Menu Navigation 19
3.4.1 Key Definitions 19
3.4.2 Menu Navigation 19
3.4.3 FossilBackKeyModeSet() 20
3.4.4 FossilBackKeyModeGet() 20
3.5 Extended Fonts 21
3.5.1 FossilExtendedFontSelectGet() 21
3.5.2 FossilExtendedFontSelectSet() 22
3.6 Miscellaneous Functions 22
3.6.1 WRISTPDA Macro 22
3.6.2 FossilIsWristPDADevice() 22
3.6.3 WPdaGetVersion( ) 22
3.6.4 FossilDisplayRefreshRateSet() 23
3.6.5 FossilDisplayRefreshRateGet() 23
3.6.6 FossilGetSystemStatus() 24
3.6.7 FossilSetWatchApp() 24
3.6.8 FossilUpdateDisplay() 24
3.6.9 FossilResetStat() 25
4 PART 4 – JOT API REFERENCE 26
4.1 General 26
4.2 The API 26
4.3 JotAPI.h 27

4.4 Brief Example 27
4.5 Detailed Example 27

5 PART 5 – WRISTPDA.H HEADER FILE 28

1 Part 1 – Development Guide

1.1 Overview

The purpose of this document is to provide software developers familiar with Palm OS programming direction on how to create applications that are "Fossil Wrist PDA friendly". That is, how to create applications that take full advantage of the Wrist PDA and run appropriately on it.

The Fossil Wrist PDA has a 66 MHz Motorola Dragonball Super VZ processor, 4 Mbytes of Flash ROM memory, 8 Mbytes of RAM memory, and runs Palm OS 4.1.2.

The Wrist PDA is capable of running most Palm OS 4.1/4.0 and earlier applications.

There are some differences between the Wrist PDA and other Palm OS devices:

- Display resolution is the same as a standard Palm OS device (160x160), but the display is physically smaller (25mm square).
- No direct-access app keys (vchrHard1 through vchrHard4).
- No Graffiti area – Write directly on the screen instead.
- Large fonts added to make it easier to view text on the small display.
- Added Rocker switch for Up, Down, and Enter key inputs.
- Added Back key.
- Added APIs to utilize the features unique to the Wrist PDA.

This document describes the project setup and development techniques to take advantage of new features of the Fossil Wrist PDA.

Check the developer sections of the Fossil and Abacus websites for SDK updates.

http://www.fossil.com/tech/
http://www.abacuswatches.com/

1.2 Development Environment

1.2.1 Requirements

A compiler capable of compiling and linking Palm OS 4.1/4.0 programs is required. CodeWarrior 8.0 with the 8.3 update was used to build the Wrist PDA applications.

Any SDK or development toolkit that is compatible with Palm OS 4.1/4.0 may be used to build applications for the Fossil Wrist PDA. Palm OS SDK 4.0 was used for the development of the Wrist PDA applications.

The included Wrist PDA emulator, the Wrist PDA device, or another Palm OS 4.1 device can be used for debugging.

1.2.2 Installation of the Software Developer's Kit

No special installation procedure is required for the Wrist PDA SDK. Just copy the folders from the CD or zip archive to a directory of your choosing on your development system.

1.2.3 Testing on the Wrist PDA Device

All Wrist PDA applications should be tested on an actual Wrist PDA device. While the Wrist PDA emulator or other Palm OS device emulators (with the Wrist PDA system extensions loaded) can be used to test basic functionality of Wrist PDA applications, the execution environments provided by the emulators are not 100% identical to the Wrist PDA device. The only way to be sure that a Wrist PDA application will execute correctly on a Wrist PDA device is to thoroughly test it on a Wrist PDA device.

1.2.4 Testing on the Wrist PDA Emulator

The Wrist PDA emulator can be used to test the basic functionality of a Wrist PDA application. For example, the Wrist PDA emulator is an effective environment in which to test form layout with the large Fossil fonts and Rocker and Back key event processing.

The Wrist PDA emulator is provided in the POSE directory of the SDK distribution. See the ReadMe.txt file in that directory for additional details about the Wrist PDA emulator.

The Wrist PDA emulator uses a custom version of the Emulator.exe executable and a custom version of the Wrist PDA rom image, WristPDA_Emulator.rom.

Other versions of Emulator.exe do not support the Wrist PDA emulator rom image.

The rom image provided with the emulator release is identical to the rom image on the actual Wrist PDA device, with one exception.

The WristPDA_Emulator.rom rom image does not use the Wrist PDA HAL (Hardware Abstraction Layer). Due to this difference, functionality that depends upon the Wrist PDA HAL is not available on the emulator. Specifically, Watch

Mode is not available. Thus the Wrist PDA emulator is ***not*** a suitable test environment in which to test Watch apps running in Watch Mode.

For most application development, this is a minor functional limitation. Most of the Wrist PDA application and system extension software development and basic functionality testing was done with a similar emulator.

1.2.5 Testing on Palm's Emulator or a Palm OS Device

Testing on a device other than the Wrist PDA or the Wrist PDA emulator can be done by first loading the Wrist PDA system extensions (*WPdaDialogs.prc*, *WPdaSysExt.prc*, and *WPdaSysExt2.prc*) and performing a soft reset of the device. The system extension binaries are found in the POSE\Bin directory of the SDK distribution.

The four new keys on the Wrist PDA are accessible on a standard emulator or other Palm OS device due to a mapping performed by the *WPdaSysExt.prc* system extension. The following table gives the mapping of the keys:

Emulator Key Mapping

STANDARD Key	WRIST PDA Key
vchrHard1 (Date Book)	vchThumbWheelUp (Rocker Up)
vchrHard2 (Address)	vchThumbWheelDown (Rocker Down)
vchrHard3 (To Do)	vchThumbWheelPush (Rocker Enter)
vchrHard4 (Notepad)	vchThumbWheelBack (Back)

1.3 CREATING WRIST PDA RESOURCES AND CODE

The following sections provide a brief description of creating resources and code for building an application specifically for the Fossil Wrist PDA, and some programming examples. Refer to Part 3 – Wrist PDA API Reference for details on the usage of each new API.

1.3.1 Using Large Fonts

Fonts are selected using *FntSetFont*(). The Palm OS standard fonts are available as well as the new large Fossil Fonts. The Wrist PDA API provides several macros to assist in changing the font on a conditional basis for applications that are written to run on the Wrist PDA and on other platforms. Refer to the API documentation and example source code for details on the usage of these macros.

Objects on the form (lists, triggers, etc.) need to be larger to accommodate the large fonts. For objects drawn by the Form Manager this is accomplished by using a special gadget, the *FossilLargerFont* gadget. Once this gadget is added

to the form, the Form Manager will automatically enlarge all objects of the form using the corresponding Fossil font range. This gadget is defined as:

- An object ID terminating with "70" decimal. (i.e. form 1500 would use 1570)
- A rectangle with coordinates 0, 1, 2, 3 (X, Y, Width, Height)
- The usable bit turned OFF

The height of lists is calculated at compile time by the resource compiler by multiplying the font height with the number of visible lines (and adding borders to it). Both Constructor and PilRC ignore the width of the checkbox object specified by the developer. As a result, lists and checkboxes are not dimensioned correctly when using the Fossil fonts. For lists and checkboxes, as well as any objects not drawn by the Form Manager (i.e. drawn at runtime) other APIs are available to enlarge objects. *FossilFrmEnlargeObject()* may be called to enlarge a single object or all objects on the form. *FossilMenuLargeFontSet()* is used to enlarge menus that are not directly associated with a form.

When using Constructor to create forms the preview that Constructor displays is not representative of what will be displayed on the actual device, since Constructor is not aware of the large fonts. That is, the view shown by Constructor is not "what you see is what you get." Programmers must allow extra horizontal and vertical space between objects to accommodate the large fonts. After enlarging the font, some iteration of positioning form objects, compiling, and viewing the results on a Wrist PDA device or emulator is necessary to get the objects on a form correctly positioned.

1.3.2 Using the Back Key and Rocker Switch

A new feature on the Wrist PDA is the addition of a Rocker Switch that provides Up, Down, and Enter key input, and a Back key. These keys are mapped to normal key down events and may be processed by an application's event handler. This allows the event handling routines to perform "Rocker Selection" by using Fossil's specific keys for various UI elements like Menu, Lists, Pop-Ups, etc. Under this concept, one item may be kept highlighted. The highlight can be moved to other items using Rocker Up and Rocker Down, or selected using the Rocker Enter key.

If an application does not process the events, the default behavior provided by the OS is:

Rocker Up:	Page up
Rocker Down:	Page down
Rocker Enter:	Nothing
Back:	Go to Launcher

If a menu is active, and the application does not process the events, the default behavior is:

Rocker Up:

If no menu is selected, the first menu first item is selected
If a menu is selected but no item, then the first item is selected
If the first item is selected, go to previous menu, last item
If the menu is already to first menu, no change, no refresh

Rocker Down:

If no menu is selected, the first menu first item is selected
If a menu is selected but no item, then the first item is selected
If the last item is selected, go to next menu, first item
If the menu is already to last menu first item, no change no refresh

Rocker Enter:

If Menu item selected, then initiate the item
If no menu item is selected, then get out of menu mode

Back:

Get out of menu mode

An application may choose to handle the Back key itself instead of automatically exiting to the Launcher by using *FossilBackKeyModeSet().* This might be the case if the app has multiple forms and chose to return to the previous form instead of exiting.

For details of the event IDs and Enabling/Disabling the Back key, refer to the API documentation.

1.3.3 Example Programs

The four standard Palm OS PIM apps (Address, DateBook, Memo, and ToDo) were modified for the Wrist PDA to use the large Fossil font set, Back key, and Rocker Switch selection. These apps are included with the Wrist PDA SDK in the Examples directory to provide examples of how to port a non-trivial existing app to the Wrist PDA platform.

Also included in the Examples directory are Watch, a fully functional Wrist PDA watch application, and JotCtrl, a program that controls Jot settings using the Jot API.

The following sections describe:

- Correct project settings for Wrist PDA development.
- Determining whether the application is running on the Wrist PDA platform.
- Writing an application that can run correctly on either the Wrist PDA or another Palm OS device.

- Using the Fossil fonts.
- Using the Rocker switch and the Back key.

Most of the changes made in the example code specifically for the Wrist PDA are marked by '#if WRISTPDA' or contain the word Fossil. You can search the source for WRISTPDA and Fossil to find the changes that were made to support the Wrist PDA.

1.3.3.1 Project Settings

The example programs were developed using CodeWarrior 8.0 with the 8.3 update. In CodeWarrior, the following settings must be changed from their default values:

Target / Target Settings / Linker	Macintosh 68K
Target / Target Settings / Post Linker	PalmRez Post Linker
Target / 68K Target / Project Type	Palm OS Application
Target / 68K Target / File Name	<app name>.tmp
Language Settings / C/C++ Language / Prefix File	WristPDA.h
Linker / 68K Linker / PalmRez Post Linker / Mac Resource	<app name>.tmp
Linker / 68K Linker / PalmRez Post Linker / Output file	<app name>.prc
Linker / 68K Linker / PalmRez Post Linker / Creator	<app's creator code>
Linker / 68K Linker / PalmRez Post Linker / Database Name	<app name>
Linker / 68K Linker / PalmRez Post Linker/ Transliteration	Palm OS 3.1 and later

The path to WristPDA.h must be included in the "Target Settings / Access Path". Refer to the .mcp project files included with the example programs.

1.3.3.2 Fossil Version of Palm OS

An application may determine at runtime whether the platform is the Fossil version of Palm OS by calling *FossilIsWristPDADevice*() or *WpdaGetVersion()*. See WristPDA.h for examples of their use.

In the example programs, the targetted OS is a compile time option and is determined using the WRISTPDA macro defined in WristPDA.h. Conditional compilation clauses using this macro are used to selectively include or exclude code as appropriate using this macro.

1.3.3.3 Fossil Fonts

The Fossil fonts are used by the Palm OS Form Manager for any form that includes the *FossilLargerFont gadget.* Refer to the .rsrc files for examples. For text objects drawn at runtime the *FntSetFont*() API is used. An example can be found in AddrDialList.c.

1.3.3.4 Rocker Selection

On the Wrist PDA, objects are highlighted to allow selection using the Rocker switch. Typically Rocker Up/Down move the selection from item to item and Enter invokes or opens the selected item.

The effort required to implement rocker selection can range from trivial to extensive. Selecting from a fixed set of options tends to be simple, while selecting items from a database tends to be complex. If the items selected from the database can be filtered in various ways, it can be very complex.

A trivial example is the Address app's Preferences dialog, see PrefsHandleEvent() in AddrPrefs.c. In this example there are only two fixed options available for selection.

ListViewMoveSelection() in MemoMain.c provides a moderate example. In this example the items are selected from a database, but there are no complex filters applied to the database records that determine which records are visible and selectable.

ListViewMoveSelection() in ToDo.c provides a complex example. In this example the items are selected from a database, and there are a number of complex filters applied to the database records that determine which records are visible and selectable.

In the *ListViewMoveSelection()* in MemoMain.c and ToDo.c examples there is a significant amount of fanout into other code within these source files. That is, the entire rocker selection implementation does not reside in the ListViewMoveSelection() routine; a significant portion resides outside this routine. Be sure to study the fanout to other code to get a complete idea of the level of effort required to implement rocker selection in these examples.

1.3.3.5 Scroll Buttons

Scroll buttons should be used in place of scrollbars on the Wrist PDA. See the resource files for examples.

Scrollbars are still functional on the Wrist PDA, but they tend to be difficult to use because they are narrow and at the edge of the display.

1.3.3.6 Display Refresh Rate

In order to reduce power consumption and extend battery life, the Wrist PDA display is updated at a slower refresh rate compared to traditional Palm OS handhelds.

The display refresh rate may be changed using the *FossilDisplayRefreshRateSet()* API. The refresh rate is reset to its default value when the running application quits.

The default display refresh rate is 5 updates per second. Most applications work fine with this default setting. Some applications that need to perform high-speed display updates, such as games with animated displays, may benefit from a higher refresh rate. Applications that set a higher refresh rate should restore the original rate as quickly as possible after their display updates are completed in order to avoid excessive power consumption and the corresponding shortened battery life.

In addition to the periodic display refresh mode that updates the display a specified number of times per second, an *automatic* mode is also provided. The automatic mode only updates the display when the contents of the DragonBall framebuffer have changed since the last update, up to a maximum of 10 updates a second. This is the preferred mode to run in for apps that do not perform extensive display updates, as it minimizes power consumption compared to updating the display a fixed number of times per second. The automatic mode is selected by calling *FossilDisplayRefreshRateSet* with a parameter value of kFossilRefreshAuto.

If an application needs an immediate display update it may call the FossilUpdateDisplay() API.

2 Part 2 – Creating a Watch Application

2.1 Overview

A Wrist PDA Watch application must be capable of running in two modes, normal mode and Watch mode. In normal mode the Watch app is launched from the system Launcher and it runs in a normal application execution environment in its own context. In Watch mode the Watch app is sublaunched by the Wrist PDA HAL once a minute and it runs in a restricted execution environment in the context of whatever app was active at the time it was sublaunched.

In general, when running in Watch mode, a Watch app should update the displayed time and date on the watch face and then exit as quickly as possible to keep power consumption to a minimum.

When running in normal mode a Watch app could implement additional functionality that it does not provide in Watch mode. For example, in normal mode a Watch app could display seconds. Seconds should not be displayed in Watch mode since the Watch app is only invoked by the HAL once a minute.

While the intent of the Watch mode implemented on the Wrist PDA is to display time and date information, a developer is free to display additional information and provide additional functionality, within the constraints imposed by Watch mode.

The source code to the reference Watch app included in the Wrist PDA ROM is included in the SDK distribution in the Examples\Watch directory.

2.2 Requirements To Be Recognized as a Watch App

To be recognized as a Watch app, the following requirements must be met by the app's prc database:

- The application type must be 'appl'.
- The application creator must be 'Foss'.
- The application resources must include the signature string with resource object id 1 and value "Wrist_PDA_Watch_App".

Applications meeting these requirements are enumerated by the Wrist PDA preference panel and they are available for selection as the active Watch app. Only applications that operate correctly as a Wrist PDA Watch app should prevent themselves as such.

2.3 REQUIRED LAUNCH CODES

A Wrist PDA Watch app must support the following launch codes:

sysAppLaunchCmdNormalLaunch :

This launch code is used when starting the Watch app in normal mode as a normal app, i.e. from the Launcher. In this mode the app does not need to perform a quick display update and then exit. The app can execute until the user chooses to exit it by whatever interface is provided to do so.

wpdaAppLaunchWatchDrawTime :

While the Wrist PDA is in Watch mode, once a minute in response to a real-time clock interrupt, the HAL invokes the Watch app with this launch code to have the Watch app draw the time with the current watch face and immediately return.

The HAL also invokes the Watch app with this launch code to provide the temporary time display that is available in Auto-Off display modes Current and Off by pressing the PageUp/Down keys.

wpdaAppLaunchWatchFaceNext :

While the Wrist PDA is in Watch mode, in response to a Rocker Down key press, the HAL invokes the Watch app with this launch code to have the Watch app switch to the next watch face, draw the time, and immediately return.

wpdaAppLaunchWatchFacePrev :

While the Wrist PDA is in Watch mode, in response to a Rocker Up key press, the HAL invokes the Watch app with this launch code to have the Watch app switch to the previous watch face, draw the time, and immediately return.

2.4 OPTIONAL LAUNCH CODE

sysAppLaunchCmdNotify / fossilNotifyWatchModeWakeup :

The HAL broadcasts this notification when the user wakes up the Wrist PDA from Watch mode by pressing the Enter key. A Watch app or other Wrist PDA aware app must register to receive this notification by calling FossilSetWatchApp and SysNotifyRegister. A Watch app may respond to this notification (or not respond to it) as it sees fit. The reference Watch app executing in normal mode exits when it receives this notification.

2.5 MANAGING GLOBAL DATA

In the Palm OS system architecture an application's A5 global data is generally not available when that application is sublaunched in another application's

context. This is the case with a Wrist PDA Watch app, which is sublaunched in the context of the app that was active at the time the Wrist PDA device went to sleep.

This does not mean that a Watch app cannot have global data. It means that it cannot have global data that the C compiler is directly aware of. It means that the developer has to go to some extra effort to create, initialize, maintain, and access global data instead of relying on the C compiler to do it.

This is a standard problem in the Palm OS application development environment that has been solved in a number of different ways. Once of the most common solutions is to store global data in a database and use the Data Manager (Dm) APIs to access it. This approach was deemed to be too high impact from a development perspective and too inefficient from a performance perspective to use in the reference Wrist PDA Watch app.

The reference Wrist PDA Watch app stores its global data in an allocated chunk of memory that is system-owned so that it persists from one invocation of the Watch app to another. A pointer to this memory chunk is stored in a Palm OS feature so that it can be retrieved by subsequent invocations of the Watch app. The memory chunk contains a structure that contains the individual global variables used by the Watch app. The PilotMain entry point into the Watch app creates and initializes the global memory structure if it doesn't already exist or retrieves the pointer to it if it does already exist. After that, the pointer to the global memory structure is passed as a parameter through the routines that comprise the Watch app.

The only critical detail with this global memory implementation is that different Watch apps ***must*** use different feature creator and feature ids to uniquely identify the feature that contains the pointer to their Watch app's global memory. The Wrist PDA supports multiple Watch apps being resident at the same time, with the user selecting which one is used for Watch mode from the Wrist PDA preference panel. The other Watch apps can still be invoked in normal mode from the Launcher and they should function correctly. For this to work, Watch apps using this global memory implementation must use unique feature creator and feature ids.

The reference Wrist PDA Watch app uses feature creator values 'Foss' and 'Abac' and feature ids 'WF' and 'WA' to identify its global memory pointer features when built as the Fossil and Abacus Watch apps, respectively. Developers that use the reference Watch app as a starting point or that use the same global memory implementation ***must*** be sure to change these values. Creator code values should be registered with PalmSource to insure global uniqueness.

2.6 DISPLAY AND SYSTEM STATE MUST BE SAVED/RESTORED

Since it is sublaunched in the context of other applications, a Watch app must completely save and restore the display state and any other system state that it modifies. See the reference watch app source for examples of accomplishing this. Failure to do this completely and correctly will result in problems that may be obscure and difficult to reproduce.

2.7 ALARM AND LOW BATTERY NOTIFICATIONS

The Wrist PDA may be in Watch mode for extended periods of time between the times it is in PDA mode. If alarms trigger (say for DateBook appointments) or if the battery is low, the user should be notified of these events while the Wrist PDA is in Watch mode. Since the active Watch app provides the only visible user interface while the Wrist PDA is in Watch mode, it is the responsibility of the Watch app to provide notification that these events have occurred.

For alarms, when the user wakes up the Wrist PDA into PDA mode via the Enter key, the HAL calls the AlmAlarmCallback() system entry point that initiates the standard system and application response to the triggered alarm. This typically results in an Attention Manager or application form being displayed to allow the user to respond to the alarm.

A Watch app may obtain the current triggered alarm and battery state by calling the FossilGetSystemStatus API. This API informs the caller if an alarm has triggered or if the battery is ok, low, very low, or empty.

The reference Watch app uses this information to provide indication to the user of this status. If an alarm has triggered then an alarm icon is drawn in the upper left corner of the screen. During processing of the wpdaAppLaunchWatchDrawTime launch code only, if an alarm has triggered then the alarm sound is played in addition to displaying the icon. If the battery status is not ok then one of three low battery icons is drawn in the upper right corner of the screen.

It is recommended that third-party Watch app developers handle alarm and battery status notification in the same manner as the reference Watch app, in order to provide a consistent user experience. The alarm and low battery icons are included in the reference Watch app project and third-party developers are encouraged to use them.

3 Part 3 – Wrist PDA API Reference

3.1 FOSSIL FONTS

3.1.1 Font ID Table

Fonts are selected using *FntSetFont*(). The Palm OS standard fonts are available as well as the large Fossil Fonts:

Palm OS standard fonts	Additional Fossil large fonts
StdFont	FossilStdFont
BoldFont	FossilBoldFont
LargeFont	FossilLargeFont
SymbolFont	FossilSymbolFont
Symbol11Font	FossilSymbol11Font
Symbol7Font	FossilSymbol7Font
LedFont	FossilLedFont
LargeBoldFont	FossilLargeBoldFont
	FossilLarge8Font *
	FossilLargeBold8Font *

*Note: While FossilLarge8Font and FossilLargeBold8Font are 'Fossil' fonts, they are NOT part of the Fossil font set! Apps that use these two fonts are directly responsible for managing their use, with no assistance from the Fossil font macros below.

Four macros are defined to simplify writing applications intended for both the Wrist PDA and standard PDA device. These are described in the following sections.

3.1.2 FossilIsFossilFont()

DESCRIPTION:

This macro determines whether the parameter *fontId* is a Fossil font.

PROTOTYPE:

```
Boolean FossilIsFossilFont(fontId); // Font ID (see sec 3.1.1)
```

RETURNS:

Returns *true* if the specified fontId is a Fossil Font, otherwise *false*.

NOTES:

This macro returns *false* for *FossilLarge8Font* and *FossilLargeBold8Font.*

3.1.3 FossilIsSystemFont()

DESCRIPTION:

This macro determines whether the parameter *fontId* is a standard Palm OS font (not a Fossil font).

PROTOTYPE:

```
Boolean FossilIsSystemFont(fontId); // font ID (see sec 3.1.1)
```

RETURNS:

Returns *true* if the specified fontId is not a Fossil Font, otherwise *false*.

NOTES:

This macro returns *false* for *FossilLarge8Font* and *FossilLargeBold8Font.*

3.1.4 FossilLargeFontID()

DESCRIPTION:

This macro is used to conditionally convert a standard Palm OS font ID to Fossil font.

If *doIt* is true and if *fontId* is a normal Palm OS font, the macro returns the Fossil font corresponding to the *fontId* parameter. Otherwise the *fontId* parameter is returned unchanged.

PROTOTYPE:

```
FontID FossilLargeFontID(Boolean doIt,   // if true, convert font
                         FontID fontId); // new font ID
```

RETURNS:

fontId or the Fossil font ID corresponding to *fontId.*

NOTES:

This macro returns the original fontId when called with *FossilLarge8Font* and *FossilLargeBold8Font.*

EXAMPLE:

The following line of code changes the system font to *FossilStdFont if* WRISTPDA is *true*, or to *stdFont* if WRISTPDA is *false*.

```
curFont = FntSetFont(FossilLargeFontID(WRISTPDA,stdFont));
```

3.1.5 FossilNormalFontID()

DESCRIPTION:

This macro is used to conditionally convert a Fossil font ID to a standard Palm OS font.

If *doIt* is true, and *fontId* is a Fossil font, the font is changed to the standard Palm OS font corresponding to the *fontId* parameter. . Otherwise the *fontId* parameter is returned unchanged.

PROTOTYPE:

```
FontID FossilNormalFontID(Boolean doIt,   // if true, convert
                          FontID fontId); // new font ID
```

RETURNS:

fontId or the standard Palm OS font ID corresponding to *fontId*.

NOTES:

This macro returns the original fontId when called with *FossilLarge8Font* and *FossilLargeBold8Font.*

3.2 FORMS

3.2.1 FossilLargerFont Gadget

All form objects capable of displaying text (except the form title) have a fontID associated with the object.

The PilRC resource compiler allows developers to select a numerical value for the form objects allowing them to directly specify a Fossil font if they wish to.

Constructor allows developers to select the font for each object containing text, but it only allows one of the Palm OS standard fonts to be selected.

A special gadget object may be added to a form to allow developers to use Constructor or PilRC to automatically enlarge all text objects on the form using a standard Palm OS font id to use the corresponding Fossil font id instead. The gadget is detected by the Form Manager at runtime and the Form Manager performs the mapping from standard Palm OS font ids to Fossil font ids.

This gadget ***must*** have the following characteristics:

- An object ID terminating with “70” decimal.
- A rectangle with coordinates 0, 1, 2, 3 (X, Y, Width, Height).
- The usable bit turned OFF.

The following macros defined in WristPDA.h describe the gadget attributes:

FossilFrmGadgetLargeFontIDMod100	70
FossilFrmGadgetLargeFontRectX	0
FossilFrmGadgetLargeFontRectY	1

FossilFrmGadgetLargeFontRectW	2
FossilFrmGadgetLargeFontRectH	3

When the Form Manager detects the gadget it does the following:

- During the call to FrmInitForm(), the form is loaded and if the gadget is detected, the font ids of all objects containing text on the form (if they are in the Palm OS range) are mapped to the corresponding Fossil font id.

- The height of the list boxes are calculated to be correct for the Fossil font that is used for that list box.

The Form Manager also uses this gadget to know that the form title must be drawn using FossilBoldFont instead of boldFont

3.2.2 FossilFrmIsLargeFont()

DESCRIPTION:

This functions determines whether the specified form is using Fossil Fonts (the Fossil gadget is present).

PROTOTYPE:

```
Boolean FossilFrmIsLargeFont( const FormType* formP );
```

RETURNS:

Returns *true* if the form is using a Fossil Font, *false* otherwise.

NOTES:

3.2.3 FossilFrmEnlargeObject()

The height of lists is calculated at compile time by the resource compiler by multiplying the font height with the number of visible lines (and adding borders to it). Also, both Constructor and PilRC ignore the width of the checkbox object specified by the developer. As a result, lists and checkboxes are not dimensioned correctly when using the Fossil fonts.

DESCRIPTION:

This function is used to enlarge one or all objects in a form.

PROTOTYPE:

```
Void FossilFrmEnlargeObject( FormType* formP,   // Form Pointer
                             UInt16 objectIdx)  // Object Index
```

RETURNS:

Nothing.

NOTES:

Must be called before a form is first displayed.

The argument *kFossilFrmEnlargeObjectAll* may be used as a special value of objectIdx to enlarge all objects in the form.

This function is automatically called by *FrmInitForm()* if the form contains the FossilLargerFont gadget.

3.3 MENUS & COMMAND BAR

Menus don't have to be associated with a form to be displayed. The following functions are provided to control the font used in such menus.

3.3.1 FossilMenuLargeFontGet()

DESCRIPTION:

This function determines whether the specified menu is displayed using a Fossil Font.

PROTOTYPE:

```
Boolean FossilMenuLargeFontGet( MenuBarType* menuP );
```

RETURNS:

Return *true* if the menu is displayed in large fonts.

NOTES:

3.3.2 FossilMenuLargeFontSet()

DESCRIPTION:

Set if a menu is to be displayed in Large font.

PROTOTYPE:

```
Boolean FossilMenuLargeFontSet( MenuBarType* menuP,
                                Boolean setLargeFont );
```

RETURNS:

Boolean - The previous menu large font state.

NOTES:

This command MUST NOT BE CALLED when the menu is already being displayed.

3.3.3 FossilMenuCmdBarIconsSet()

DESCRIPTION :

Sets the Menu Command Bar Fossil mode.

The *newIconFlags* parameter is any combination of:

kFossilMenuCmdNone
kFossilMenuCmdMenu
kFossilMenuCmdFind
kFossilMenuCmdKeyboard
kFossilMenuCmdAll

PROTOTYPE:

```
UInt16  FossilMenuCmdBarIconsSet( UInt16 newIconFlags );
```

RETURNS:

Previous Menu Command Bar Mode as described above.

NOTES:

3.3.4 FossilMenuCmdBarIconsGet()

DESCRIPTION:

Gets the Menu Command Bar Fossil mode.

PROTOTYPE:

```
UInt16 FossilMenuCmdBarIconsGet(void);
```

RETURNS:

Valid values are any combination of :

kFossilMenuCmdNone
kFossilMenuCmdMenu
kFossilMenuCmdFind
kFossilMenuCmdKeyboard
kFossilMenuCmdAll

NOTES:

3.3.5 FossilMenuSetActiveMenuRscID()

DESCRIPTION:

Set the active menu from a Resource ID and indicates if this menu if to be enlarged.

PROTOTYPE:

```
void FossilMenuSetActiveMenuRscID(UInt16 resourceId,
                                  Boolean largeMenu );
```

RETURNS:

Nothing.

NOTES:

If largeMenu is *true*, the menu is displayed using Fossil Fonts.

3.4 KEYS & MENU NAVIGATION

3.4.1 Key Definitions

In order to provide compatibility with Palm OS 5 handhelds and future versions of Palm OS, the Fossil Wrist PDA keys use the same keyDownEvent virtual character codes as the Palm OS vchrThumbWheel keys:

Rocker In (Enter)	vchrThumbWheelPush
Rocker Up	vchrThumbWheelUp
Rocker Down	vchrThumbWheelDown
Back key	vchrThumbWheelBack

If the running application does not process these events (returns *handled* == *false* from it's event handler), then the processing of those events on return from *FrmDispatchEvent*() is as follows*:*

*vchrThumbWheelUp***:** Replaced by a *vchrPageUp*
*vchrThumbWheelDown***:** Replaced by a *vchrPageDown*
*vchrThumbWheelPush***:** No action
*vchrThumbWheelBack***:** Replaced by an appStopEvent

This allows the form handling routines to process Fossil's specific keys for various UI elements like Menu, Lists, Pop-Up, etc. Moreover, if the UI Element is not being processed by the application, this allows having a default behavior that uses standard Palm OS events.

Details of the default behavior while a menu is open is given in section 3.4.2. An application may override these default behaviors by processing these events itself, using the functions described in sections 3.4.3 and 3.4.4.

3.4.2 Menu Navigation

By default, once a menu is open the following keys will have the following behavior:

For **vchrPageUp** and **vchrThumbWheelUp**:
If no menu is selected, the first menu first item is selected
If a menu is selected but no item, then the first item is selected
If the first item is selected, go to previous menu, last item
If the menu is already to first menu, no change, no refresh

For **vchrPageDown** and **vchrThumbWheelDown**
If no menu is selected, the first menu first item is selected
If a menu is selected but no item, then the first item is selected
If the last item is selected, go to next menu, first item
If the menu is already to last menu last item, no change no refresh

For **vchrThumbWheelPush**

If Menu item selected, then initiate this menu item
If no menu item is selected, then get out of menu mode.

For **vchrThumbWheelBack**

Get out of menu mode

Hidden menu items and separation lines are skipped over.

Note that the menu can be open in many different ways including using the Command bar icon or tapping on the form's title.

An application may override the default behavior by processing these events itself, using the functions described in sections 3.4.3 and 3.4.4.

3.4.3 FossilBackKeyModeSet()

DESCRIPTION:

Sets the Back key interpretation mode:

```
kFossilBackKeyNoAction   // Ignore this key
kFossilBackKeyLauncher   // Puts this key in the event
                         // queue, if not handled, insert
                         // an appStopEvent
kFossilBackKeyStopEvent  // Insert an appStopEvent (without
                         // putting any key in the event
                         // queue)
```

PROTOTYPE:

```
void    FossilBackKeyModeSet( UInt16 newBackKeyMode );
```

RETURNS:

Nothing.

NOTES:

3.4.4 FossilBackKeyModeGet()

DESCRIPTION:

Gets the Back key interpretation mode.

PROTOTYPE:

```
void FossilBackKeyModeGet(void);
```

RETURNS:

Current Back Key mode:

```
kFossilBackKeyNoAction   // Ignore this key
kFossilBackKeyLauncher   // Puts this key in the event
```

```
                                    // queue, if not handled, insert
                                    // an appStopEvent
            kFossilBackKeyStopEvent // Insert an appStopEvent (without
                                    // putting any key in the event
                                    // queue)
```

NOTES:

3.5 EXTENDED FONTS

The *FossilExtendedFontSelectSet* API allows the developer to enable or disable an extended implementation of the system *FontSelect* API. By default, the extended *FontSelect* dialog is disabled. The developer must make a call to *FossilExtendedFontSelectSet* to enable it.

The extended FontSelect dialog displays six fonts for selection instead of the usual three. The six fonts are displayed in this order from left to right:

FossilStdFont
FossilBoldFont
FossilLarge8Font
FossilLargeBold8Font
FossilLargeFont
FossilLargeBoldFont

While the extended *FontSelect* dialog is enabled the FossilLarge8Font and FossilLargeBold8Font fonts are loaded into memory and available for use via calls to *FntSetFont*.

The extended *FontSelect* state is not restored to the default state when an application exits, so the application should save the state on entry and restore it on exit.

3.5.1 FossilExtendedFontSelectGet()

DESCRIPTION:
This function returns the FontSelect extended dialog state.

PROTOTYPE:

```
void   FossilExtendedFontSelectGet(Boolean * extendedP );
```

RETURNS:
Nothing.

NOTES:

3.5.2 FossilExtendedFontSelectSet()

DESCRIPTION:

This function controls use of the extended FontSelect dialog.

PROTOTYPE:

```
void   FossilExtendedFontSelectSet(Boolean extended);
```

RETURNS:

Nothing.

NOTES:

3.6 MISCELLANEOUS FUNCTIONS

3.6.1 WRISTPDA Macro

The WRISTPDA macro is defined in WristPDA.h for use in applications that run on both the Fossil WristPDA platform and the standard Palm OS. Developers can use this macro in conditional compilation clauses to include or exclude source code as appropriate for the current build target.

3.6.2 FossilIsWristPDADevice()

DESCRIPTION:

Determine if this is a Wrist PDA device.

This provides a simpler alternative to WpdaGetVersion for code to determine if it is running on a Wrist PDA, specifically, if the Wrist PDA system extensions are running.

PROTOTYPE:

```
Boolean FossilIsWristPDADevice(void)
```

PARAMETERS: None

RETURNS:

Boolean - *true* if this is a Wrist PDA device, undefined if not a Wrist PDA device.

NOTES:

The return values is *undefined* if this call is made on a non Wrist PDA device, so the return value must be explicitly compared only to *true*.

3.6.3 WPdaGetVersion( )

DESCRIPTION:

This macro retrieves the Wrist PDA version number.

PROTOTYPE:

```
Err WPdaGetVersion(Uint32 *verP)          // pointer to Uint32 to
                                          //  place version #
```

RETURNS:

errNone if the Fossil OS is running, otherwise an error code.

NOTES:

3.6.4 FossilDisplayRefreshRateSet()

DESCRIPTION:

Set refresh rate in number of system ticks.

In order to reduce power consumption, the Wrist PDA display is updated at a slower refresh rate compared to traditional Palm OS handhelds. An application using this function should restore the previous refresh rate as soon as possible to prevent battery drainage and other possible system implications.

The refresh rate is reset to its default value when the running app quits (when Launcher or any other app is executed).

PROTOTYPE:

```
void FossilDisplayRefreshRateSet(UInt16 newRefreshRate);

    // newRefreshRate = number of system ticks per second.
```

RETURNS:

Nothing.

NOTES:

Two special constants are available for parameter *newRefreshRate*:

kFossilRefreshAuto to set rate automatically
kFossilRefreshDefault to set the default rate

3.6.5 FossilDisplayRefreshRateGet()

DESCRIPTION:

Gets the display refresh rate in number of system ticks

PROTOTYPE:

```
UInt16  FossilDisplayRefreshRateGet(void)
```

RETURNS:

One refresh per rate/*SysTicksPerSecond*() seconds, or
kFossilRefreshAuto indicating that Auto refresh mode is active

NOTES:

3.6.6 FossilGetSystemStatus()

DESCRIPTION:

Direct the HAL to return system status (alarm triggered, low battery state) that is needed by Watch applications.

PROTOTYPE:

```
void FossilGetSystemStatus(void);
```

RETURNS:

UInt16 - The requested system status:
kFossilSystemStatusAlarmNotTriggered
kFossilSystemStatusAlarmTriggered
kFossilSystemStatusBatteryOk
kFossilSystemStatusBatteryLow
kFossilSystemStatusBatteryVeryLow
kFossilSystemStatusBatteryEmpty

3.6.7 FossilSetWatchApp()

DESCRIPTION:

Inform the HAL if the current app is a Watch app.

Only Watch apps should make this call, other apps should not make this call.

When the HAL has been informed that the current app is a Watch app, it will broadcast the fossilNotifyWatchModeWakeup notification upon wakeup from Watch mode. A Watch app may respond to this notification (or ignore it) as it sees fit.

PROTOTYPE:

```
void FossilSetWatchApp(Boolean WatchAppFlag) // true if watch app
```

RETURNS:

Nothing

3.6.8 FossilUpdateDisplay()

DESCRIPTION:

Direct the HAL to update the display immediately by copying the contents of the DragonBall framebuffer to the UltraChip framebuffer.

PROTOTYPE:

```
void FossilUpdateDisplay(void);
```

RETURNS:

Nothing

3.6.9 FossilResetStat()

DESCRIPTION:

Resets the Wrist PDA to compatibility mode.

PROTOTYPE:

```
void FossilResetStat(void);
```

RETURNS:

Nothing.

NOTES:

4 Part 4 – Jot API Reference

4.1 General

Jot is controlled by sublaunching it with specific launch codes that control its behavior (features). The turning on and off of features is only temporary since they are not written back into the preferences. A soft-reset will always restore the user selected preferences.

In general, if an application would like to turn off, for example, the full screen inking, it should do so on start (StartApplication) and reset the feature when it terminates (StopApplication). The application can check the return value in order to determine if a feature was actually turned on/off. In our example, if the application wants to turn off full screen inking but full screen inking was turned off already in the preferences, the application should not turn on full screen inking when it quits.

4.2 The API

The include file JotAPI.h contains the constant definitions used in the Jot API.

The Palm OS function to call to sublaunch Jot with a specified launch code is:

SysAppLaunch(cardNo, dbID, 0, 32801, (char *) & fFeature, NULL);

fFeature is defined as a UInt32, and can be set to any of the following bit masks:

DISABLE_JOT
ENABLE_JOT

These flags turn Jot off and on. If the call is successful, this flag will be cleared from fFeature when SysAppLaunch returns.

DISABLE_WRITING
ENABLE_WRITING

These flags disable or enable writing directly on the screen. Note that this does not determine if ink is actually shown. If the call is successful, this flag will be cleared from fFeature when SysAppLaunch returns.

DISABLE_INKING
ENABLE_INKING

These flags turn off/on the ink shown on the screen. If the call is successful, this flag will be cleared from fFeature when SysAppLaunch returns.

DISABLE_MODEMARK
ENABLE_MODEMARK

These flags turn off/on the mode mark if full screen writing is enabled. If the call is successful, this flag will be cleared from fFeature when SysAppLaunch returns.

4.3 JOTAPI.H

```
// LaunchCode Features

#define ENABLE_JOT         0x0001
#define DISABLE_JOT        0x0002
#define ENABLE_WRITING     0x0004
#define DISABLE_WRITING    0x0008
#define ENABLE_INKING      0x0010
#define DISABLE_INKING     0x0020
#define ENABLE_MODEMARK    0x0040
#define DISABLE_MODEMARK   0x0080

#define JOTCREATOR         'JWJT'
```

4.4 BRIEF EXAMPLE

```
/*
 *---------------------------------------------------------------------------
 * Example to turn off inking and writing on the screen
 *---------------------------------------------------------------------------
 */

UInt16              cardNo;
LocalID             dbID = NULL;
DmSearchStateType   searchState;
UInt32              fFeature;

fFeature = DISABLE_WRITING | DISABLE_INKING;
DmGetNextDatabaseByTypeCreator(true, &searchState, sysFileTApplication,
JOTCREATOR, true, &cardNo, &dbID);
SysAppLaunch(cardNo, dbID, 0, 32801, (char *)&fFeature, NULL);
```

4.5 DETAILED EXAMPLE

A detailed Jot API example is presented in the JotCtrl program in the Examples directory of the Wrist PDA SDK.

5 Part 5 – WristPDA.h Header File

```
/***********************************************************************
 *
 * Copyright (c) 2003 PalmSource, Inc.  All rights reserved.
 * Copyright (c) 2003 Fossil, Inc.  All rights reserved.
 *
 * File:        WristPDA.h
 *
 * Description: Include file for the Fossil Wrist PDA.
 *
 ***********************************************************************/

#ifndef __WRISTPDA_H__
#define __WRISTPDA_H__

#include <PalmOS.h>

#include "HAL.h"

//////////////////////////////////////////////////////////////////////////
// Use the WRISTPDA macro to conditionally compile Wrist PDA code
//
#define WRISTPDA                    1

//////////////////////////////////////////////////////////////////////////
// Fossil creator codes
//
#define WPdaCreator                 'Foss'  // WPdaSysExt.prc & other use
#define WPdaDialogsCreator          'FosD'  // WPdaDialogs.prc
#define WPdaExt2Creator             'FosX'  // WPdaSysExt2.prc
#define sysFileCWristPDAPnl         'wpda'  // WristPDAPnl.prc

//////////////////////////////////////////////////////////////////////////
// Use the WPdaGetVersion macro to detect if running on a Fossil OS or not,
// or use the FossilIsWristPDADevice API defined later in this file
//
#define WPdaFtrNumVersion           0
#define WPdaGetVersion( verP )  \
            (Err) FtrGet( WPdaCreator, WPdaFtrNumVersion, verP )

//////////////////////////////////////////////////////////////////////////
// Fossil Key definitions
//
// Key bit masks
#define keyBitRockerUp              0x1000      // Rocker switch up
#define keyBitRockerDown            0x2000      // Rocker switch down
#define keyBitEnter                 0x4000      // Rocker switch in
#define keyBitBack                  0x8000      // Back key

//////////////////////////////////////////////////////////////////////////
// Fossil virtual key character codes
//

// Palm OS 5.0 R2 Thumb Characters (also known as Jog)
// The following values exist to allow all hardware that has these
// (optional) control clusters to emit the same key codes.

#define vchrThumbWheelUp            0x012E      // Thumb-wheel up
#define vchrThumbWheelDown          0x012F      // Thumb-wheel down
```

```
#define vchrThumbWheelPush          0x0130      // Thumb-wheel press/center
#define vchrThumbWheelBack          0x0131      // Thumb-wheel cluster back

// Alternate Fossil names for the standard Thumb vchr codes

#define vchrHardRockerEnter         vchrThumbWheelPush  // Rocker switch in
#define vchrHardRockerUp            vchrThumbWheelUp    // Rocker switch up
#define vchrHardRockerDown          vchrThumbWheelDown  // Rocker switch down
#define vchrHardBack                vchrThumbWheelBack  // Back key

//////////////////////////////////////////////////////////////////////////////
// Fossil defines for UI
//
#define WPDA_BUTTON_HEIGHT          16  // Default button height on forms

//////////////////////////////////////////////////////////////////////////////
// Fossil Font IDs (map one-to-one onto original PalmOS font set)
//
#define FossilFontIDStart           0x10
#define FossilStdFont               (FossilFontIDStart +0)
#define FossilBoldFont              (FossilFontIDStart +1)
#define FossilLargeFont             (FossilFontIDStart +2)
#define FossilSymbolFont            (FossilFontIDStart +3)
#define FossilSymbol11Font          (FossilFontIDStart +4)
#define FossilSymbol7Font           (FossilFontIDStart +5)
#define FossilLedFont               (FossilFontIDStart +6)
#define FossilLargeBoldFont         (FossilFontIDStart +7)
#define FossilFontIDEnd             (FossilFontIDStart +7)

//////////////////////////////////////////////////////////////////////////////
// Fossil Extended Font IDs (no mapping onto original PalmOS font set)
//
#define FossilExtendedFontIDStart   0x20
#define FossilLarge8Font            (FossilExtendedFontIDStart +0)
#define FossilLargeBold8Font        (FossilExtendedFontIDStart +1)
#define FossilExtendedFontIDEnd     (FossilExtendedFontIDStart +1)

// Note: While FossilLarge8Font and FossilLargeBold8Font are 'Fossil' fonts,
//       they are NOT part of the Fossil font set with a one-to-one mapping
//       onto the standard font set.  Apps that use these two fonts are
//       directly responsible for managing their use, with no assistance from
//       the Fossil font macros below.

//////////////////////////////////////////////////////////////////////////////
// Fossil Font ID Macros
//
#define FossilIsFossilFont( _fontId ) \
            ((_fontId >= FossilFontIDStart) && (_fontId <= FossilFontIDEnd))

#define FossilIsSystemFont( _fontId ) \
            ((_fontId >= stdFont) && (_fontId <= largeBoldFont))

#define FossilLargeFontID( _doIt, _fontId ) \
            (_fontId + (_doIt && FossilIsSystemFont(_fontId) ? \
                FossilFontIDStart : 0))

#define FossilNormalFontID( _doIt, _fontId ) \
            (_fontId - (_doIt && FossilIsFossilFont(_fontId) ? \
                FossilFontIDStart : 0))

//////////////////////////////////////////////////////////////////////////////
// Fossil notifications
//
```

```
// The fossilNotifyWatchModeWakeup notification is broadcast upon
// wakeup from Watch mode when a Watch app is the current app
// Watch apps respond to this notification as they see fit
#define fossilNotifyWatchModeWakeup     'wkup'

//////////////////////////////////////////////////////////////////////////
// Fossil Watch app custom launch codes
//

// Draw the time with the current face and immediately return
#define wpdaAppLaunchWatchDrawTime  ( sysAppLaunchCmdCustomBase + 0 )

// Switch to the next face, draw the time, and immediately return
#define wpdaAppLaunchWatchFaceNext  ( sysAppLaunchCmdCustomBase + 1 )

// Switch to the previous face, draw the time, and immediately return
#define wpdaAppLaunchWatchFacePrev  ( sysAppLaunchCmdCustomBase + 2 )

//////////////////////////////////////////////////////////////////////////
// Fossil Main API
//
#define FossilAPITrap       sysTrapOEMDispatch

#define FOSSIL_API(sel) \
        _SYSTEM_API(_CALL_WITH_SELECTOR)(_SYSTEM_TABLE,FossilAPITrap,sel)

#define FossilAPIUnused1                    0
#define FossilAPIUnused2                    1
#define FossilAPIBackKeyModeGet             2
#define FossilAPIBackKeyModeSet             3
#define FossilAPIMenuCmdBarIconsGet         4
#define FossilAPIMenuCmdBarIconsSet         5
#define FossilAPIDisplayRefreshRateGet      6
#define FossilAPIDisplayRefreshRateSet      7
#define FossilAPIResetStat                  8
#define FossilAPIFrmIsFossil                9
#define FossilAPIMenuSetActiveMenuRscID     10
#define FossilAPIMenuLargeFontGet           11
#define FossilAPIMenuLargeFontSet           12
#define FossilAPIFrmEnlargeObject           13
#define FossilAPIMaxSelector                14

#define FossilIsWristPDADeviceOpCode        ( WPdaExt2Creator + 0 )
#define FossilIsWristPDAHALOpCode           ( WPdaExt2Creator + 1 )
#define FossilExtendedFontSelectGetOpCode   ( WPdaExt2Creator + 2 )
#define FossilExtendedFontSelectSetOpCode   ( WPdaExt2Creator + 3 )

/*****************************************************************
 *
 *  FUNCTION:       FossilIsWristPDADevice
 *
 *  DESCRIPTION:    Determine if this is a Wrist PDA device.
 *
 *                  This provides a simpler alternative to WPdaGetVersion
 *                  for code to determine if it is running on a Wrist PDA,
 *                  i.e. if the Wrist PDA system extensions are available.
 *
 *  PARAMETERS:     None
 *
 *  RETURNS:        Boolean - true if this is a Wrist PDA device,
 *                            undefined if not a Wrist PDA device.
 *
```

```
 *                  Note: The return values is *undefined* if this call
 *                        is made on a non Wrist PDA device, so the return
 *                        value must be explicitly compared only to true.
 *
 *****************************************************************/
#define FossilIsWristPDADevice() \
    HwrCustom( WPdaExt2Creator, \
               FossilIsWristPDADeviceOpCode, \
               NULL, \
               NULL )

/*****************************************************************
 *
 *  FUNCTION:       FossilIsWristPDAHAL
 *
 *  DESCRIPTION:    Determine if this is a Wrist PDA HAL.
 *
 *  PARAMETERS:     None
 *
 *  RETURNS:        Boolean - true if this is a Wrist PDA HAL,
 *                            undefined if not a Wrist PDA HAL.
 *
 *                  Note: The return values is *undefined* if this call
 *                        is made on a non Wrist PDA device, so the return
 *                        value must be explicitly compared only to true.
 *
 *****************************************************************/
#define FossilIsWristPDAHAL() \
    HwrCustom( WPdaExt2Creator, \
               FossilIsWristPDAHALOpCode, \
               NULL, \
               NULL )

/*****************************************************************
 *
 *  FUNCTION:       FossilBackKeyModeGet
 *
 *  DESCRIPTION:    Gets the Back key interpretation mode.
 *
 *  PARAMETERS:     None
 *
 *  RETURNS:        See FossilBackKeyModeSet() for return values.
 *
 *****************************************************************/
UInt16  _FossilBackKeyModeGet(void)
            FOSSIL_API(FossilAPIBackKeyModeGet);

#define FossilBackKeyModeGet() \
        ( FossilIsWristPDADevice() == true ) ? \
             _FossilBackKeyModeGet() : 0

#define kFossilBackKeyNoAction  0x0000
#define kFossilBackKeyLauncher  0x0001
#define kFossilBackKeyStopEvent 0x0002  // Default set by FossilResetStat()
                                        // and by Launching a new app.

/*****************************************************************
 *
 *  FUNCTION:       FossilBackKeyModeSet
 *
 *  DESCRIPTION:    Sets the Back key interpretation mode.
 *
 *  PARAMETERS:     newBackKeyMode - The new Back key mode.
```

```
 *
 *                  Valid values are:
 *
 *                      kFossilBackKeyNoAction  - Ignore this key.
 *
 *                      kFossilBackKeyLauncher  - Puts this key in the event
 *                                                queue. If not handled,
 *                                                insert an appStopEvent.
 *
 *                      kFossilBackKeyStopEvent - Insert an appStopEvent
 *                                                (without putting any key
 *                                                in the event queue).
 *  RETURNS:        Void
 *
 ***********************************************************************/
void    _FossilBackKeyModeSet( UInt16 newBackKeyMode ) \
            FOSSIL_API(FossilAPIBackKeyModeSet);

#define FossilBackKeyModeSet( newBackKeyMode ) \
        if ( FossilIsWristPDADevice() == true ) \
             _FossilBackKeyModeSet( newBackKeyMode )

/***********************************************************************
 *
 *  FUNCTION:       FossilMenuCmdBarIconsGet
 *
 *  DESCRIPTION:    Gets the Menu Cmd Bar Fossil mode.
 *
 *  PARAMETERS:     None
 *
 *  RETURNS:        UInt16 - Bitmask indicating which icons are present.
 *
 *                  Valid values are any combination of:
 *
 *                          kFossilMenuCmdNone
 *                          kFossilMenuCmdMenu
 *                          kFossilMenuCmdFind
 *                          kFossilMenuCmdFindMenu
 *
 ***********************************************************************/
UInt16  _FossilMenuCmdBarIconsGet(void)
            FOSSIL_API(FossilAPIMenuCmdBarIconsGet);

#define FossilMenuCmdBarIconsGet() \
        ( FossilIsWristPDADevice() == true ) ? \
            _FossilMenuCmdBarIconsGet() : 0

#define kFossilMenuCmdNone      0x0000
#define kFossilMenuCmdMenu      0x0001
#define kFossilMenuCmdFind      0x0002
#define kFossilMenuCmdKeyboard  0x0004
#define kFossilMenuCmdAll       \
            (kFossilMenuCmdMenu | kFossilMenuCmdFind | kFossilMenuCmdKeyboard)

/***********************************************************************
 *
 *  FUNCTION:       FossilMenuCmdBarIconsSet
 *
 *  DESCRIPTION:    Sets the Menu Cmd Bar Fossil mode.
 *
 *  PARAMETERS:     newIconFlags - Bitmask indicating which icons are present
 *
 *                  Valid values are any combination of:
```

```
 *
 *                          kFossilMenuCmdNone
 *                          kFossilMenuCmdMenu
 *                          kFossilMenuCmdFind
 *                          kFossilMenuCmdFindMenu
 *
 *  RETURNS:        UInt16 previous menu cmd bar icon bitmask.
 *
 ***********************************************************************/
UInt16  _FossilMenuCmdBarIconsSet( UInt16 newIconFlags )
            FOSSIL_API(FossilAPIMenuCmdBarIconsSet);

#define FossilMenuCmdBarIconsSet( newIconFlags ) \
        ( FossilIsWristPDADevice() == true ) ? \
            _FossilMenuCmdBarIconsSet( newIconFlags ) : 0

/***********************************************************************
 *
 *  FUNCTION:       FossilMenuSetActiveMenuRscID
 *
 *  DESCRIPTION:    Set the active menu from a Resouirce ID
 *                  and indicate if this menu if to be enlarged.
 *
 *  PARAMETERS:     resourceId - Id of menu resource to make active
 *
 *                  largeMenu  - true if menu to be displayed w/ large font
 *
 *  RETURNS:        void
 *
 ***********************************************************************/
void    _FossilMenuSetActiveMenuRscID(UInt16 resourceId, Boolean largeMenu )
            FOSSIL_API(FossilAPIMenuSetActiveMenuRscID);

#define FossilMenuSetActiveMenuRscID( resourceId, largeMenu ) \
        if ( FossilIsWristPDADevice() == true ) \
            _FossilMenuSetActiveMenuRscID( resourceId, largeMenu )

/***********************************************************************
 *
 *  FUNCTION:       FossilMenuLargeFontGet
 *
 *  DESCRIPTION:    Get the large font state of a specified menu.
 *
 *  PARAMETERS:     menuP - Ptr to the menu to query.
 *
 *  RETURNS:        Boolean - true if the menu pointed to by menuP is
 *                            displayed with large font, false otherwise.
 *
 ***********************************************************************/
Boolean _FossilMenuLargeFontGet( MenuBarType* menuP )
            FOSSIL_API(FossilAPIMenuLargeFontGet);

#define FossilMenuLargeFontGet( menuP ) \
        ( FossilIsWristPDADevice() == true ) ? \
            _FossilMenuLargeFontGet( menuP ) : false

/***********************************************************************
 *
 *  FUNCTION:       FossilMenuLargeFontSet
 *
 *  DESCRIPTION:    Set if a menu is to be displayed in large font.
 *
 *  PARAMETERS:     menuP - Ptr to the menu to set.
```

```
 *
 *                  setLargeFont - true to display menu w/ large font.
 *
 *  RETURNS:        Boolean - The previous menu large font state.
 *
 ***********************************************************************/
Boolean _FossilMenuLargeFontSet( MenuBarType* menuP, Boolean setLargeFont )
            FOSSIL_API(FossilAPIMenuLargeFontSet);

#define FossilMenuLargeFontSet( menuP, setLargeFont ) \
        ( FossilIsWristPDADevice() == true ) ? \
            _FossilMenuLargeFontSet( menuP, setLargeFont ) : false

/***********************************************************************
 *
 *  FUNCTION:       FossilDisplayRefreshRateGet
 *
 *  DESCRIPTION:    Gets the display refresh rate in number of system ticks.
 *
 *  PARAMETERS:     None
 *
 *  RETURNS:        UInt16 - The current display refresh rate in ticks.
 *                          (1 refresh per rate/SysTicksPerSecond() seconds).
 *
 *                  One special return value is defined:
 *
 *                      kFossilRefreshAuto - Auto refresh mode is active.
 *
 ***********************************************************************/
UInt16  _FossilDisplayRefreshRateGet( void )
            FOSSIL_API(FossilAPIDisplayRefreshRateGet);

#define FossilDisplayRefreshRateGet() \
        ( FossilIsWristPDADevice() == true ) ? \
            _FossilDisplayRefreshRateGet() : 1

/***********************************************************************
 *
 *  FUNCTION:       FossilDisplayRefreshRateSet
 *
 *  DESCRIPTION:    Set the display refresh rate in number of system ticks.
 *
 *  PARAMETERS:     newRefreshRate - The new display refresh rate in ticks.
 *                      (1 refresh per rate/SysTicksPerSecond() seconds).
 *
 *                  Two special parameter values are defined:
 *
 *                      kFossilRefreshAuto    - Use auto refresh mode.
 *                      kFossilRefreshDefault - Use default refresh rate.
 *
 ***********************************************************************/
void    _FossilDisplayRefreshRateSet( UInt16 newRefreshRate )
            FOSSIL_API(FossilAPIDisplayRefreshRateSet);

#define FossilDisplayRefreshRateSet( newRefreshRate ) \
        if ( FossilIsWristPDADevice() == true ) \
            _FossilDisplayRefreshRateSet( newRefreshRate )

#define kFossilRefreshAuto      0xFFFE
#define kFossilRefreshDefault   0xFFFD

/***********************************************************************
 *
```

```
 *  FUNCTION:       FossilResetStat
 *
 *  DESCRIPTION:    Resets the Wrist PDA to compatibility mode.
 *
 *                  All Fossil modes are reset to the most compatible state.
 *
 *  PARAMETERS:     None
 *
 *  RETURNS:        Void
 *
 ***********************************************************************/
void    _FossilResetStat( void )
            FOSSIL_API(FossilAPIResetStat);

#define FossilResetStat() \
        if ( FossilIsWristPDADevice() == true ) \
            _FossilResetStat()

// Fossil Form gadget spec

#define FossilFrmGadgetLargeFontIDMod100        70

#define FossilFrmGadgetLargeFontRectX           0
#define FossilFrmGadgetLargeFontRectY           1
#define FossilFrmGadgetLargeFontRectW           2
#define FossilFrmGadgetLargeFontRectH           3

#define FossilFrmGadgetLargeFontRect            \
            { FossilFrmGadgetLargeFontRectX,    \
              FossilFrmGadgetLargeFontRectY,    \
              FossilFrmGadgetLargeFontRectW,    \
              FossilFrmGadgetLargeFontRectH }

/***********************************************************************
 *
 *  FUNCTION:       FossilFrmIsLargeFont
 *
 *  DESCRIPTION:    Determine if a speficied form is a Fossil friendly form.
 *                  (That is, if the Fossil gadget is present).
 *
 *  PARAMETERS:     formP - Ptr to form to query.
 *
 *  RETURNS:        Boolean - true if form has Fossil gadget,
 *                            false otherwise.
 *
 ***********************************************************************/
Boolean _FossilFrmIsLargeFont( const FormType* formP )
            FOSSIL_API(FossilAPIFrmIsFossil);

#define FossilFrmIsLargeFont( formP ) \
        ( FossilIsWristPDADevice() == true ) ? \
            _FossilFrmIsLargeFont( formP ) : false

/***********************************************************************
 *
 *  FUNCTION:       FossilFrmEnlargeObject
 *
 *  DESCRIPTION:    Enlarge one or all objects in a form.
 *
 *                  Must be called before a form is first displayed.
 *
 *  PARAMETERS:     formP - Ptr to form to operate on.
 *
```

```
 *                  objectIdx - Index of object to enlarge, or
 *                              kFossilFrmEnlargeObjectAll to enlarge all
 *                              objects on the form.
 *
 *  RETURNS:        void
 *
 ***********************************************************************/
void    _FossilFrmEnlargeObject( FormType* formP, UInt16 objectIdx )
            FOSSIL_API(FossilAPIFrmEnlargeObject);

#define FossilFrmEnlargeObject( formP, objectIdx ) \
        if ( FossilIsWristPDADevice() == true ) \
            _FossilFrmEnlargeObject( formP, objectIdx )

#define kFossilFrmEnlargeObjectAll      0xFFFF

/***********************************************************************
 *
 *  FUNCTION:       FossilExtendedFontSelectGet
 *
 *  DESCRIPTION:    Return FontSelect extended dialog state.
 *
 *                  Boolean * extendedP - Ptr to Boolean to receive state
 *
 *  PARAMETERS:     extendedP - Ptr to variable to receive the extended
 *                              dialog state; true if in extended state.
 *
 *  RETURNS:        void
 *
 ***********************************************************************/
#define FossilExtendedFontSelectGet( extendedP ) \
        if ( FossilIsWristPDADevice() == true ) \
            HwrCustom( WPdaExt2Creator, \
                       FossilExtendedFontSelectGetOpCode, \
                       (void *) extendedP, \
                       NULL ); \
        else \
            * extendedP = false

/***********************************************************************
 *
 *  FUNCTION:       FossilExtendedFontSelectSet
 *
 *  DESCRIPTION:    Control use of extended FontSelect dialog.
 *
 *  PARAMETERS:     extended - Indicates whether to use extended
 *                             dialog or not.
 *
 *                             true  -> Use extended dialog
 *                             false -> Use original dialog
 *
 *  RETURNS:        void
 *
 ***********************************************************************/
#define FossilExtendedFontSelectSet( extended ) \
        if ( FossilIsWristPDADevice() == true ) \
            HwrCustom( WPdaExt2Creator, \
                       FossilExtendedFontSelectSetOpCode, \
                       (void *) extended, \
                       NULL )

//////////////////////////////////////////////////////////////////////////
// Fossil HAL API
```

```
//

#define hwrWristPDAUpdateDisplayOpCode      1
#define hwrWristPDAGetSystemStatusOpCode    2
#define hwrWristPDASetWatchAppOpCode        3

/*****************************************************************
 *
 *  FUNCTION:       FossilUpdateDisplay
 *
 *  DESCRIPTION:    Direct the HAL to update the display by copying the
 *                  contents of the DragonBall framebuffer to the
 *                  UltraChip framebuffer.
 *
 *  PARAMETERS:     None
 *
 *  RETURNS:        void
 *
 *****************************************************************/
#define FossilUpdateDisplay() \
    if ( FossilIsWristPDAHAL() == true ) \
        HwrCustom( WPdaCreator, \
                   hwrWristPDAUpdateDisplayOpCode, \
                   NULL, \
                   NULL )

/*****************************************************************
 *
 *  FUNCTION:       FossilGetSystemStatus
 *
 *  DESCRIPTION:    Direct the HAL to return system status (alarm triggered,
 *                  low battery state) that is needed by Watch applications.
 *
 *  PARAMETERS:     RequestType - The type of system status requested.
 *
 *                      Must be one of these values:
 *
 *                          kFossilGetAlarmState
 *                          kFossilGetBatteryState
 *
 *  RETURNS:        UInt16 - The requested system status.
 *
 *                      Returns one of these values:
 *
 *                          kFossilSystemStatusAlarmNotTriggered
 *                          kFossilSystemStatusAlarmTriggered
 *
 *                          kFossilSystemStatusBatteryOk
 *                          kFossilSystemStatusBatteryLow
 *                          kFossilSystemStatusBatteryVeryLow
 *                          kFossilSystemStatusBatteryEmpty
 *
 *****************************************************************/
#define FossilGetSystemStatus( RequestType ) \
    ( FossilIsWristPDAHAL() == true ) ? \
        HwrCustom( WPdaCreator, \
                   hwrWristPDAGetSystemStatusOpCode, \
                   (void *) RequestType, \
                   NULL ) : 0

// Values for FossilGetSystemStatus RequestType parameter

#define kFossilGetAlarmState    0
```

```
#define kFossilGetBatteryState  1

// Return values for FossilGetSystemStatus

#define kFossilSystemStatusAlarmNotTriggered    0
#define kFossilSystemStatusAlarmTriggered       1

#define kFossilSystemStatusBatteryOk            0
#define kFossilSystemStatusBatteryLow           1
#define kFossilSystemStatusBatteryVeryLow       2
#define kFossilSystemStatusBatteryEmpty         3

/*****************************************************************
 *
 *  FUNCTION:       FossilSetWatchApp
 *
 *  DESCRIPTION:    Inform the HAL if the current app is a Watch app.
 *
 *                  Only Watch apps should make this call, other apps
 *                  should not make this call.
 *
 *                  When the HAL has been informed that the current app
 *                  is a Watch app, it will broadcast the
 *                  fossilNotifyWatchModeWakeup notification upon wakeup
 *                  from Watch mode.  A Watch app may respond to this
 *                  notification (or ignore it) as it sees fit.
 *
 *  PARAMETERS:     WatchAppFlag - Boolean, true for a Watch app,
 *                                 false otherwise.
 *
 *  RETURNS:        void
 *
 *****************************************************************/

#define FossilSetWatchApp( WatchAppFlag ) \
    ( FossilIsWristPDAHAL() == true ) ? \
        HwrCustom( WPdaCreator, \
                   hwrWristPDASetWatchAppOpCode, \
                   (void *) WatchAppFlag, \
                   NULL ) : 0

#endif  //  __WRISTPDA_H__
```