HandEra 330

Programmer's Companion

Revision 1.04
Revision Date: July 2001

CONTACT INFORMATION

HandEra, Inc.

2857 104th St.
Des Moines, IA 50322
U.S.A.
515-252-7525
www.handera.com

Palm Computing World Wide Web www.palm.com

Metrowerks World Wide Web www.metrowerks.com

Table of Contents

Chapter 1: About This Document

The *HandEra 330 Programmer's Companion* is part of the HandEra Software Development Kit (SDK). This document is intended as a guide for developers interested in creating applications for the HandEra 330 handheld computer.

For the most recent version of the SDK, check at http://www.handera.com.

This development kit does not contain information regarding the design and implementation of standard Palm OS applications. For this information, please refer to the Palm OS SDK documentation provided by Palm at http://www.palmos.com. Relevant documents include:

- *Palm OS SDK Reference*
- *Palm OS Programmer's Companion*
- *Constructor for Palm OS*

This document assumes the reader is familiar with basic Palm OS concepts detailed in *the Palm OS Programmer's Companion*. This document also uses the basic typographical conventions found in Palm documentation.

- Code elements, such as functions and structures, will use a fixed width font.
- **Bold type will be used for emphasis.**
- Document names, such as *Palm OS Programmer's Companion*, are italicized.

Application Development

Basic Palm OS Applications

This document does not contain information regarding the design and implementation of standard Palm OS applications. For this information, please refer to the Palm OS SDK documentation provided by Palm at http://www.palmos.com.

Tools

The recommended software development system for HandEra 330s is CodeWarrior for Palm OS from Metrowerks (http://www.metrowerks.com). CodeWarrior for Palm OS is an integrated Palm OS development system that includes a compiler, linker, assembly-level debugger, and a Palm emulator. The sample programs for the SDK are included as CodeWarrior projects, developed under CodeWarrior for Palm OS, Release 7. It is important to use the Constructor for Palm OS version 1.5 or newer.

The header files provided by HandEra work with both the GNU compiler and CodeWarrior. If GNU or some other development platform is being used for development, the examples in this SDK can be used as a reference.

SDK

The SDK zip file contains the following:

- **\Include** – HandEra 330 specific header files.
- **\Example** – CodeWarrior example projects.
- **\Docs** – Programmer's Companion and Reference Manual.

The example projects in this Software Development Kit (SDK) use the following tools for development:

- CodeWarrior R7
- Constructor v1.6
- PalmOS SDK 3.5

Palm OS Emulator (POSE)

The Palm OS® Emulator (POSE) is software that runs on desktop PCs and emulates the hardware of the various models of Palm OS platform devices. For a more complete description of running POSE or installing ROMs, refer to the Developers' Section at www.palmos.com.

Chapter 2: Features and Definitions

This chapter presents background information necessary for developers interested in writing applications that take advantage of the features found on the HandEra 330. This information includes an overview of the HandEra 330's feature set, as well as a detailed look at some of its more advanced concepts.

HandEra 330 Features

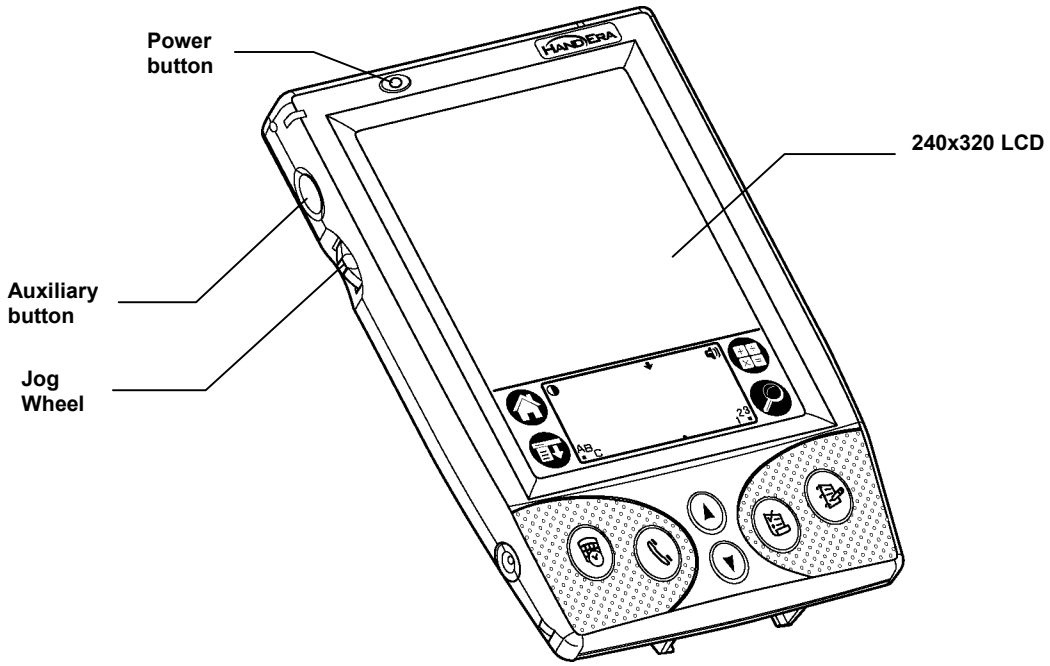

Figure 2-1. HandEra 330

Physical Characteristics

As shown on Figure 2-1 and Figure 2-2, the mechanical design of the HandEra 330 is compatible with the Palm III form factor. This provides backward compatibility with existing 3rd party hardware add-on devices, such as portable keyboards. The design has been modified to include the expansion capabilities of the HandEra 330.

Power

The HandEra 330 is the first Palm Powered device to support both alkaline and rechargeable Lithium Ion Batteries. External power can be supplied to the unit either through the cradle or directly into the unit via the side power jack.

Memory and CPU

The HandEra 330 comes standard with 8 MB of DRAM and 2 MB of flash. It includes the Motorola Dragonball Vz processor running at 33 MHz.

Audio

Typically, Palm Powered platform devices have limited sound generation, with the ability to play simple tones. The HandEra 330 includes support for playing and recording .WAV files through an enhanced onboard speaker and microphone.

Buttons

As a rule, Palm Powered devices include a single row of hardware buttons near the base of the unit – the Date Book, Address, To Do List, Memo Pad, and arrow buttons. In addition to these buttons, the HandEra provides two new buttons – a jog wheel and an auxiliary button located on the left-hand side of the unit. Refer to Figure 2-1. Support for these buttons is further described .

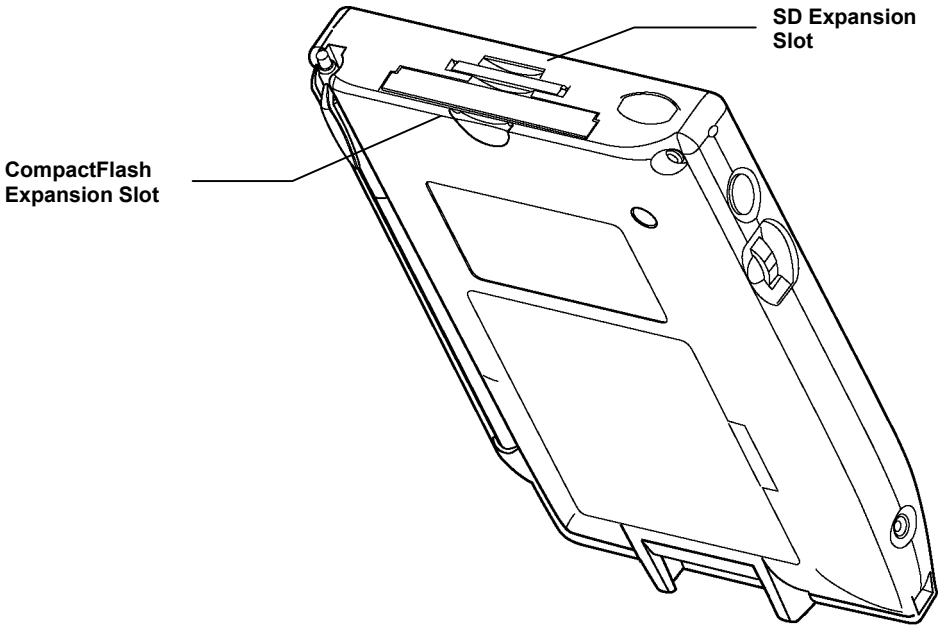

Figure 2-2. Expansion Slots

Expansion

The HandEra 330 comes with two industry standard expansion slots – CompactFlash (CF) and Secure Digital (SD). CF supports both storage and I/O devices while SD is limited to storage devices at this time. The HandEra 330's CF and SD support is compatible with the Expansion and File Manager found in the Palm 4.0 SDK. Applications which take advantage of the expansion media on the Sony Clie and Palm m505 should also work on the HandEra 330.

Screen

To date, most Palm Powered devices have featured a screen resolution of 160x160 pixels. The HandEra 330 features an LCD with a resolution of 240x320 pixels. The increased pixel density allows for more detailed images to be displayed. Figure 2-3 shows two versions of the application launcher. On the left is an example of the launcher running on a 160x160 screen. On the right is an example of the launcher running on the higher resolution HandEra 330 screen.

Figure 2-3. Application Launcher

The screen resolution of 240 x 320 pixels is referred to as Quarter VGA, or QVGA for short. VGA is the Video Graphics Array computer graphics standard developed by IBM for its Personal Computer. 240 x 320 pixels is one-fourth the size of the VGA standard 640 x 480.

Definitions

Screen Modes

The concept of a screen mode is new to the Palm OS Computing platform. The screen mode is a setting that determines how an application's forms are displayed. The HandEra 330 provides support for two screen modes:

- **Scale To Fit** – To provide compatibility with the thousands of 3rd party applications developed for the Palm Powered platform, the HandEra 330 includes support for expanding 160x160 forms to fit the 240x240 pixels above the Silkscreen. This is the default screen mode for applications.
- **1 to 1 Mapping** – This screen mode does not expand the program's forms. This mode is used for form sizes greater than 160x160 developed for the higher resolution LCD screen.

❖ **Note:** The concept of a screen mode applies to a single form, not to an application. An application may contain forms which run in *Scale To Fit* mode and forms which run in *1 to 1 Mapping* mode.

Screen Rotation

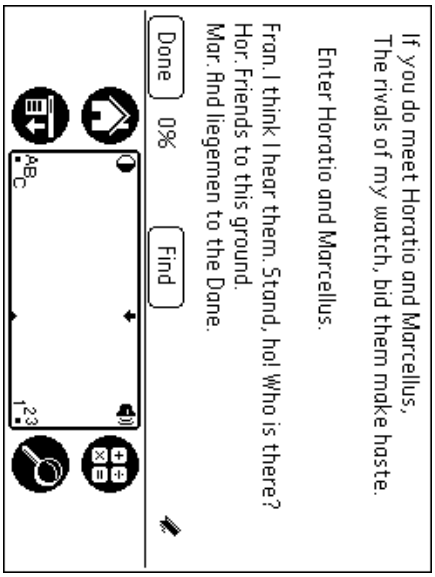

Figure 2-4. Screen Rotation

The HandEra 330 also supports screen rotation. Applications can be developed which display forms at 90°, 180°, 270° angles from the origin. For example, spreadsheet apps have been developed which can be rotated 90°, allowing more columns to be displayed.

- Warning: Due to limitations of the Dragonball Vz processor, screen rotations other than 0° are not recommended for applications using animation.

Silkscreen

Silkscreening is a method of printmaking that impresses a design over a framed screen of silk. Since the introduction of the Pilot 1000, all Palm Powered devices have featured a silkscreened area immediately beneath the LCD. Traditionally, this area has included:

- Two Graffiti® writing areas, one for alphabetical character entry and one for numeric character entry. The system event manager converts strokes in the Graffiti area into the appropriate key events.
- The Applications, Menu, Calculator, and Find icons. Taps on the icons are also converted into key events.

Some devices now include a Contrast icon as well. Japanese-localized devices have a silkscreened area customized for Kanji input.

This area is referred to as the **Silkscreen** for the purposes of this document. The Silkscreen for the HandEra 300 is not actually silkscreened, as there is no printmaking process involved. Instead, the area has been *virtualized*, so that the Graffiti areas and the icons are rendered on the bottom 240x80 pixels of the LCD.

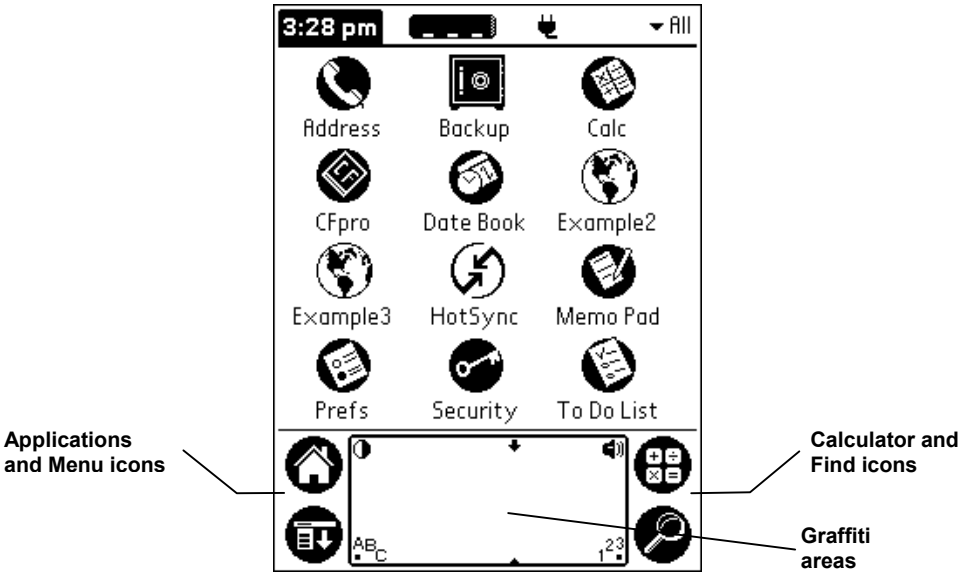

Figure 2-5. Application Launcher plus Silkscreen

The Silkscreen for the HandEra 330 also contains a contrast and volume icon. Tapping on these buttons pops up a dialog to set the contrast and volume respectively.

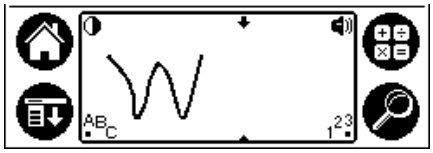

Figure 2-6. Graffiti

It is important to note that the functionality of the virtualized Silkscreen is very similar to that of its printed equivalent. The system event manager converts strokes in the Graffiti area and taps on the icons into the appropriate key events.

A virtualized Silkscreen has the following advantages:

- The Graffiti area can show *inking*, which is the echo of the user's stylus stroke. Refer to Figure 2-6. This can be beneficial for learning the Graffiti system.
- The Graffiti area is backlit to aid in pen input under low light conditions.
- Applications can make use of the area when the Silkscreen is minimized.
- The area can be customized on the fly for localization. For example, the area could be programmatically altered to emulate Japanese devices.

Because the Silkscreen on the HandEra 330 is virtualized, it can be run in two different modes – Maximized or Minimized.

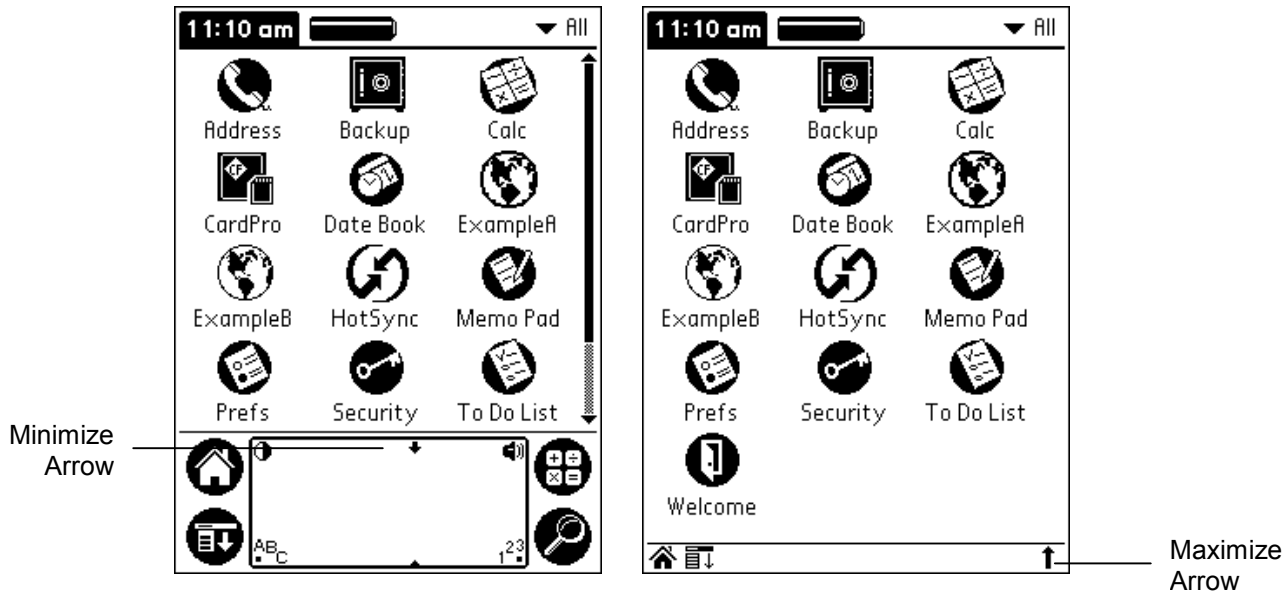

Figure 2-7. Maximized / Minimized

- **Maximized** – In Maximized mode, the Silkscreen is drawn at full height. The area contains the full-sized Applications, Menu, Calculator, and Find icons, as well as the Graffiti area.
- **Minimized** – In Minimized mode, the Silkscreen is drawn at a reduced height. The area contains smaller icons and no Graffiti area.

The user can switch between these modes by tapping on the Minimize and Maximize arrows in the Silkscreen.

Chapter 3: Programming Overview

Steps For Building HandEra 330 Apps

The HandEra 330 supports most Palm Powered apps. However, apps fine-tuned to work with the QVGA screen will offer the best look and performance. To modify an existing application for the QVGA screen and maintain compatibility with existing Palm Powered devices, perform the following steps:

1. **Create high resolution icons and bitmaps**

   High-resolution icons and bitmaps can be added to an application as additional resources. No code changes are required. See Chapter 4: High-Resolution Icons and Bitmaps for further details.

2. **Create high-resolution forms**

   Support for high-resolution forms can be achieved by either creating distinct forms at compile-time or by resizing existing forms at run-time with API calls provided by the HandEra 330. Resizing a form at run-time is preferable since it eliminates the need for duplicate resources and code. See Chapter 5: for further details

3. **Add a QVGA identifier resource**

   All QVGA-aware apps need to include a special ‘sKst’ resource. This resource is used to differentiate QVGA apps from legacy apps. For example, the Applications Preferences panel displays all such apps as “High-Res”. See Chapter 5: for details on adding this resource to an application.

4. **Add display extent event-handling**

   The application should handle the events associated with changes to the display extent. Typically, this would be triggered when a user maximizes or minimizes the Silkscreen. When such an event is received, the application should resize the current form. See Chapter 6: for further details.

5. **Add one-hand operation support (optional)**

   One-hand applications allow a user to hold and operate the Palm Powered device with one hand. The user is able to navigate between forms and dialogs by using the hardware buttons. Although not required, it is possible to convert an existing app to support one-hand operation by processing the jog wheel and auxiliary button events. See Chapter 7: for more information.

Chapter 4: High-Resolution Icons and Bitmaps

This chapter describes how to add high-resolution icons and bitmaps to applications. This is accomplished by simply including additional resources to the application – no code change is required. The application can then be run on any Palm Powered device, since the additional resources will simply be ignored on non-QVGA handhelds.

Families

Since the introduction of Palm OS 1.0, the graphics standard for the Palm Powered platform has experienced a number of significant changes. Palm OS 1.0 originally could only support B/W 1 bit/pixel images. As of Palm OS 3.5, there is now support for 1, 2, 4, or 8 bits/pixel for grayscale and color images. To accommodate these changes while maintaining backward-compatibility with existing applications, the Palm Powered platform introduced the idea of *families*. A family is a grouping of a particular resource. Only one member of a family is selected in any given context.

For example: Recent versions of the OS now support bitmap families, a group of bitmaps that contain the same basic image but each with a different pixel depth. When the application issues a command to draw a bitmap family, the OS selects the version of the bitmap with the pixel depth equal to the current mode.

The family approach allows applications to support new graphics hardware by changing their resources only – no programming changes are required.

The HandEra 330 provides support for new bitmap and icon families to complement the existing Palm Powered platform standards. These will be defined later in this chapter.

Application Icons

The Application Launcher displays a window from which users can run applications installed on the Palm Powered device.

On non-HandEra units, the Launcher presents the Icon View by default, in which installed applications are represented graphically as 22x22 pixel icons. Refer to Figure 4-1. The launcher also supports List View, which displays 15x9 icons to allow you to see more applications on the screen at once. Refer to Figure 4-2.

Figure 4-1. 22x22 Icons

Figure 4-2. 15x9 Icons

Under current Palm Powered standards, each application contains two application icon family resources, one for each View of the launcher.

- The 22x22 icon family used with the Icon View has a resource ID of 1000.
- The 15x9 icon family used with the List View has a resource ID of 1001.

Application icon family resources, 'tAIF', groups a number of identically-dimensioned icon images of different color depths (1-bit, 2-bit, 4-bit, or 8-bit) into a single resource. The resource element 'tAIF' is converted into a single 'TAIB' resource before it is linked into the resulting .prc file.

- ❖ Note: GNU tools do not support the 'tAIF' intermediate resource.

The HandEra 330 builds upon this idea of icon views by introducing additional higher-resolution icons. The following additional icon families are supported:

- The 32x32 icon family with a resource ID of 2000.
- The 22x14 icon family used a resource ID of 2001.
- The 32x32 inverted icon family with a resource ID of 2002.

- ❖ Note: the inverted icon family (2002) is used as part of the virtualized Silkscreen. The launcher simply inverts the 1000 and 1001 family icons when tapped by the user.

As before, each icon within a given family has the same resolution, but a different color depth (1-bit, 2-bit, 4-bit, or 8-bit).

No special code is necessary in an application to support the higher-resolution icons. As a result, the application can run on any Palm Powered device. Other devices ignore the higher-resolution icons.

The HandEra 330 Application Launcher includes support for four views:

- Large Icon View
- Small Icon View
- Large List view
- Small List Viewon

Large Icon View – This is the default view used by the launcher. This view is comparable to the Icon View on other Palm Powered devices. In this view, the launcher attempts to display a 32x32 icon resource for each application. If this resource is not found, the 22x22 icon is scaled to fit. Refer to Figure 4-3. Note the icon for Example1 is a 22x22 icon scaled to fit, while the icon for Example2 is a 32x32 icon.

Figure 4-3. Large Icon View

Small Icon View – In this view, the launcher displays a 22x22 icon resource for each application. Refer to Figure 4-4.

Figure 4-4. Small Icon View

Large List View – This view is comparable to the List View on other Palm devices. In this view, the launcher attempts to display a 22x14 icon resource for each application. The 22x14 icon is a new standard designed for higher resolution displays. If this resource is not found, the 15x9 icon is scaled to fit. Refer to Figure 4-5.

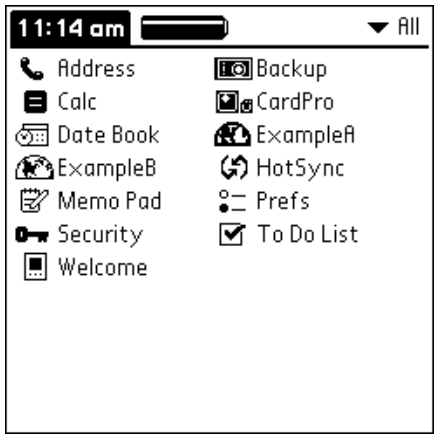

Figure 4-5. Large List View

Small List View – In this view, the launcher displays a 15x9 icon resource for each application. Refer to Figure 4-6.

Figure 4-6. Small List View

Bitmaps

Under the current Palm Powered standards, the bitmap family resource, 'tBMF', groups a number of identically-dimensioned bitmap images of different bit depths into a single resource. The resource element 'tBMF' is converted into a single 'Tbmp' resource before it is linked into the resulting `.prc` file. When the application makes a function call to render the 'Tbmp' resource, the OS selects the version of the bitmap with the pixel depth equal to the current mode. If such a bitmap is not found, the bitmap with the pixel depth closest to but less than the display depth is chosen.

Bitmap Family Tree

The HandEra 330 builds upon the bitmap family concept by introducing the idea of a **bitmap family tree**. A bitmap family tree consists of an original bitmap family and its associated higher-resolution bitmap family. Developers are responsible for ensuring that the dimensions of the higher-resolution bitmap family are 1.5 times greater than the original family's dimensions. When an application renders a 'Tbmp' resource on the screen, the screen extension searches the bitmap family tree for a higher-resolution bitmap family for that resource. If the higher-resolution family is not found, the extension scales the existing bitmap family by 1.5.

The figure below shows two example 160x160 legacy applications. ExampleA does not contain the higher-resolution bitmap family – its bitmap is expanded 1.5 times by the HandEra 330. ExampleB does contain a higher-resolution bitmap family, and as result, the HandEra 330 displays the higher-resolution bitmap instead. Note: These apps will look identical on non-HandEra Palm Powered devices.

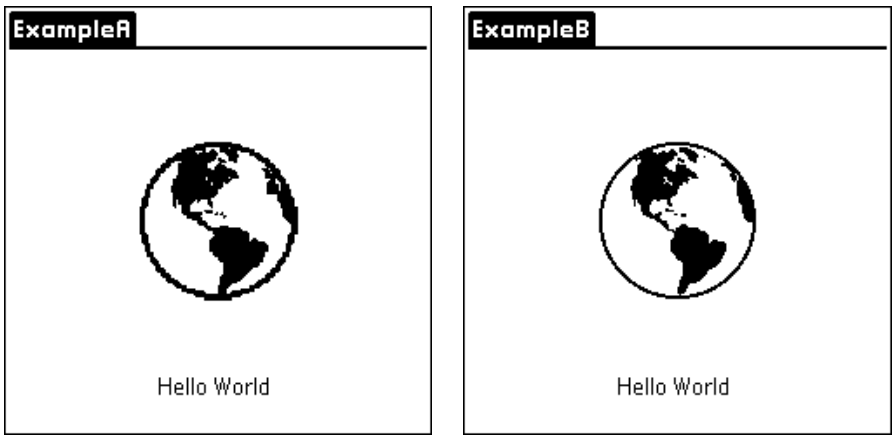

Figure 4-7.

Forms which contain bitmap family trees will work automatically. No special handling is required.

If the application contains code that draws the bitmaps manually, the standard Palm functions `DmGetResource`, `DmGet1Resource`, `WinDrawBitmap` and `WinPaintBitmap` can be used. The extension will use the higher-resolution bitmap family if found, otherwise the original bitmap family is expanded by 1.5. However, there is one restriction. After calling either `DmGetResource` or `DmGet1Resource`, neither of these functions should be called again until the bitmap has been drawn. Back-to-back calls to get resources will produce unpredictable results.

This HandEra 330 implementation does not effect any of the previous bitmap standards for the Palm Powered platform. All of the following are still supported:

- Color tables.
- Versions 0-2 of bitmap encoding and compression
- Naming conventions introduced in Palm OS 3.0 are still supported.

Creating Bitmap Family Trees

Developers can use either of the following methods for creating bitmap family trees:

- 'Sbmp' Resource
- 'Tbmp' Offset Resource

'Sbmp' Method

The 'Sbmp' resource is new to the Palm Powered platform. 'Sbmp' is structurally equivalent to a 'Tbmp' resource. 'Sbmp' is implemented to define an associative relationship between an existing 'Tbmp' resource and its higher-resolution equivalent. Developers need to create an 'Sbmp' resource that is 1.5 times larger and with the identical resource ID to create this relationship.

Unfortunately, Constructor cannot create 'Sbmp' resources. To create 'Sbmp' resources, developers need to use third party tools, such as PilRC. Alternatively, developers can use the 'Tbmp' Offset method.

'Tbmp' Offset Method

The 'Tbmp' Offset method involves creating a second 'Tbmp' resource which is a higher-resolution equivalent resource. Developers need to ensure that the resource ID value of this higher-resolution version is exactly 4096 (1000 hexadecimal) greater than the existing 'Tbmp' resource. The advantage to this approach is that these resources can be created with existing Constructor (v1.6). This method is not as "clean" as the 'Sbmp' method, but is necessary until Constructor supports 'Sbmp' resources.

Creating a Sibling Resource Database

High-resolution icons and bitmaps families do not need to reside with in the application itself. These resources can be put into a separate database, referred to as a **Sibling Resource Database**. The screen extension first looks for these resources in a database with the same creator as the application with a type of 'strt'. If the resource is not found in the separate database, the extension looks for it in the application itself. Finally, if the resource is not found in the application, the extension expands the resource by 1.5.

The disadvantage of this method is that an additional file must be included with the application. The advantage of this method is that a new version of an application does not have to be created. The Sibling Resource Database can be created for existing apps.

Programming Examples in this Chapter

Example A is a simple ‘Hello World’ app that contains bitmaps and text. It does not contain any modifications to run on the QVGA screen. It serves as a starting point for the subsequent examples.

Example B builds on Example A. It demonstrates how high-resolution icons and bitmaps can be added to an application.

Example C is a Sibling Resource Database for Example A. It demonstrates how high-resolution icons and bitmaps can be added to a secondary file, without having to modify the original.

Example A – Hello World

Example A is an example of a **legacy application** – that is an application designed to run on 160x160 displays. This section walks you through the steps involved in creating and installing a simple "Hello World" application on HandEra 330. Example A displays a picture of the globe on the screen with the text "Hello World" printed below it.

Example A is a starting point for some rudimentary discussions of text and graphics – it does not use any HandEra 330 screen specific features. Subsequent chapters build on this simple application to demonstrate how to take advantage of the features specific to HandEra 330.

Perform the following steps:

1. Copy the files from the `\Source\ExampleA` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleA Project**.
3. Open the resource file **Starter.rsrc** using Constructor. Refer to Figure 4-8.

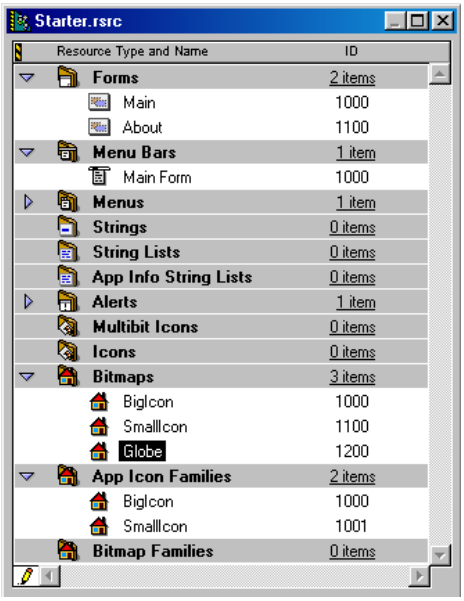

Figure 4-8. Starter.rsrc

This initial example project contains very few resources:

- **Main form** – Blank 160x160 full-screen non-modal form. The main forms only object is a title. Refer to Figure 4-9.
- **About form** – About form found in many Palm applications.
- **MainForm** – Menu bar contains *Font* and *About ExampleA* options.
- **BigIcon** – 22x22 icon used by the launcher in Icon View.
- **SmallIcon** – 15x9 icon used by the launcher in List View.

- **Globe** – 60x60 bitmap of the world. Refer to Figure 4-10.

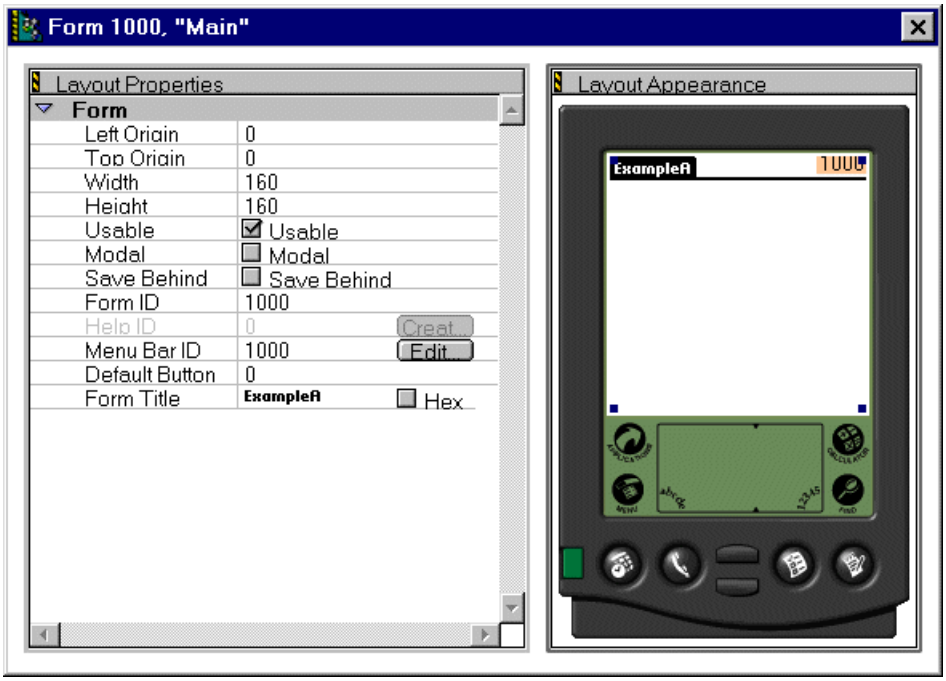

Figure 4-9. Main Form

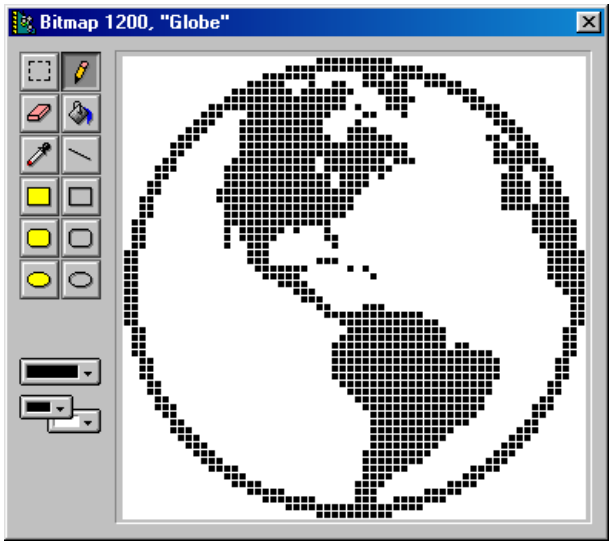

Figure 4-10. Globe

6. **Install ExampleA.prc** on a HandEra 330 or a HandEra 330-compatible version of POSE.

7. **Launch ExampleA.prc**.

HandEra 330 expands the bitmaps associated with Example A, 1.5 times to fit the 240x240 display area. The application's icon and the globe bitmap are both scaled by the OS extensions to fit the display. Refer to Figure 4-11 and Figure 4-12.

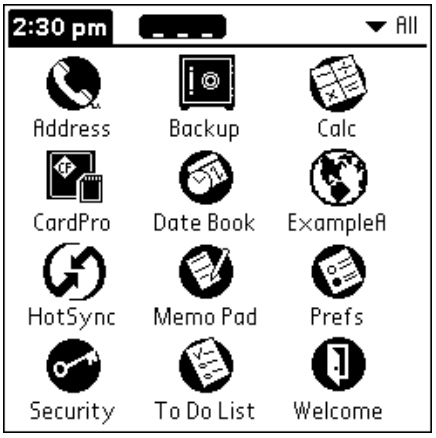

Figure 4-11. Example A Icon

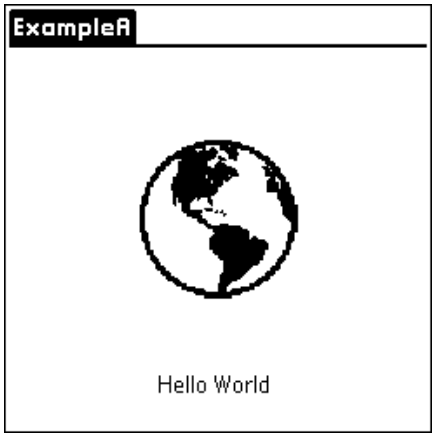

Figure 4-12. Example A

Example B – Icons and Bitmaps

Example B provides examples of ...

- *High-resolution icons*
- *High-resolution bitmaps*

This example explores the issues involved in creating high-resolution icons and bitmaps for HandEra 330. Example B shows that with no code changes, developers can add high-resolution icons and bitmaps to their applications to take advantage of HandEra 330's higher-resolution screen.

Example B builds on the "Hello World" example. In Example A, HandEra 330 expanded the icons and bitmaps automatically at run time. Higher-resolution graphics may be preferred. HandEra 330 looks for the presence of high-resolution versions of bitmap and icon resources within an application, and if present, uses the high-resolution versions instead. Other Palm Computing devices simply ignore these duplicate resources.

HandEra 330 has added support for two additional icon sizes that are approximately 1.5 times larger than the two standard icons. For example: In the "Large Icon View" in the HandEra 330 launcher, icons with ID of 2000 display if present. Otherwise, the original icon expands by 1.5.

HandEra 330 also allows you to provide both a low and a high-resolution version of each bitmap in their application.

- **Low-resolution** – Original bitmap designed for 160x160 displays.
- **High-resolution** – New bitmap is 1.5 times larger than the low-resolution bitmap with a resource ID equal to the low resolution's ID plus 0x1000.

In Figure 4-13, notice that Example B contains both a low and a high-resolution version of the globe bitmap.

- **Globe** – ID 1200 (0x04B0) is a 60x60 bitmap
- **VGAGlobe** – ID 5296 (0x14B0) is a 90x90 bitmap.

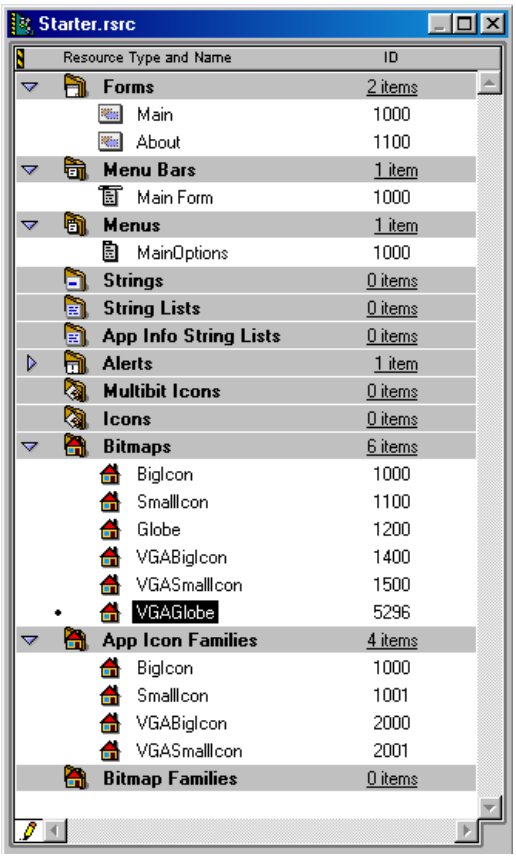

Figure 4-13.

Perform the following:

1. Install ExampleB.prc on a HandEra 330 or a HandEra 330 compatible version of POSE.
2. Launch ExampleB.prc.

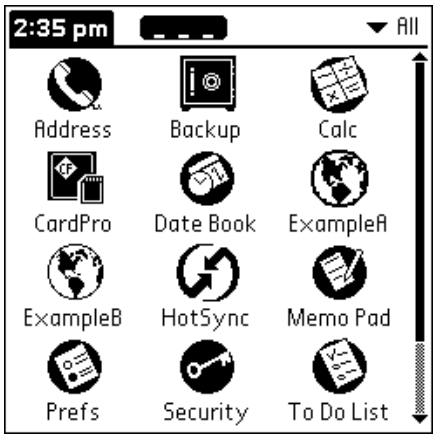

Figure 4-14. Example B Icon

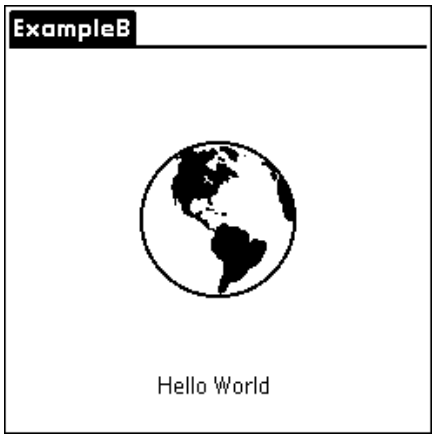

Figure 4-15. Example B

Notice in Figure 4-14 how ExampleB's icon is sharper than ExampleA's icon. ExampleB produces the output in Figure 4-15. Notice how much sharper the globe appears in ExampleB, Figure 4-15, compared to ExampleA, Figure 4-12.

Example C – Sibling Resource Database

Example C provides examples of ...

- *High-resolution icons*
- *High-resolution bitmaps*
- *Sibling resource database*

Example C demonstrates how to create a sibling resource database. High-resolution bitmaps and icons can be placed in a separate database with the same creator as the application and a type of 'strt'. The VGA extension will look for these resources first in this database, then the application itself. If the high-resolution resource does not exist, the extension will expand the resource by 1.5.

Perform the following steps:

1. Copy the files from the `\Source\ExampleC` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleC Project**.
3. Open the Settings dialog.

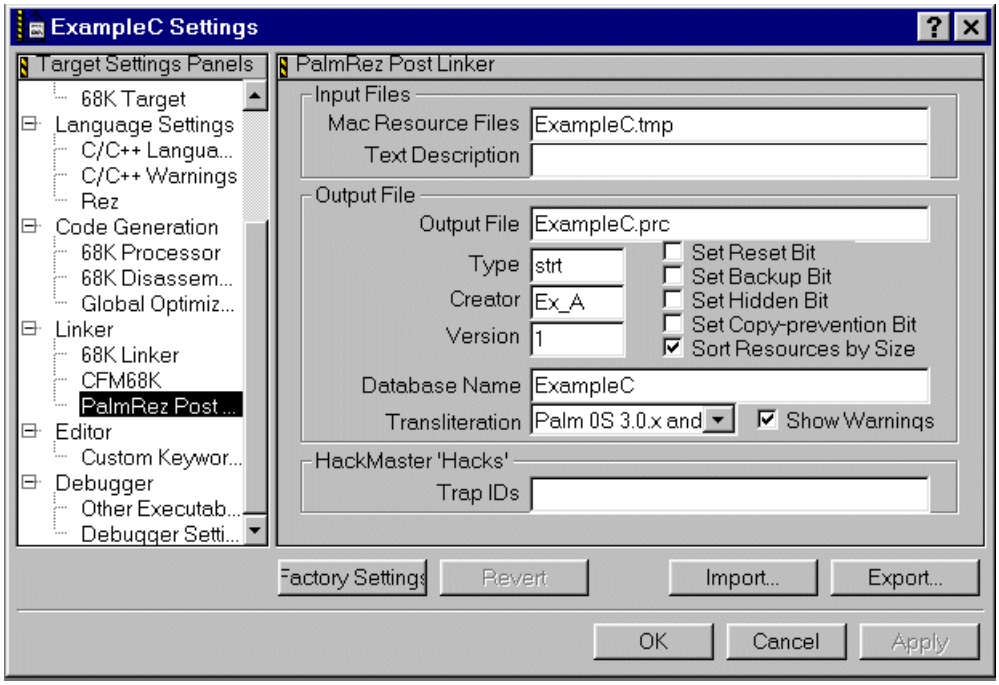

Figure 4-16. Type 'strt'

Note that Example C is of type 'strt'. This defines it as a Sibling Resource Database. Its creator is 'Ex_A'. This defines it as the sibling of Example A.

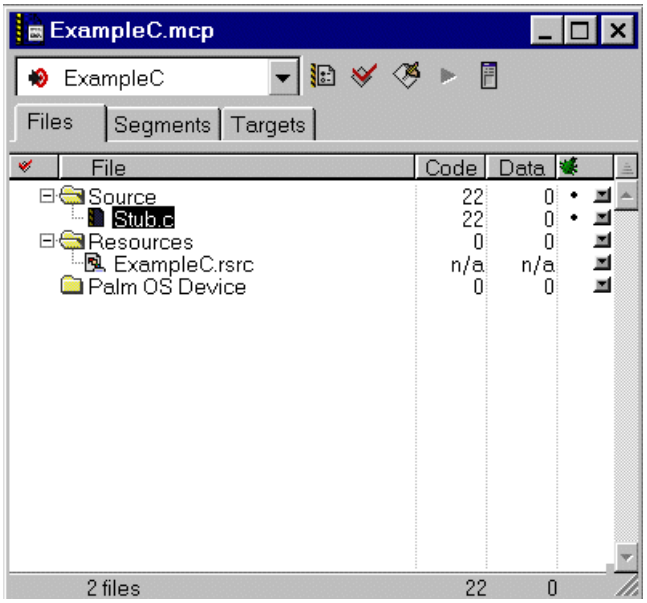

Figure 4-17. Example C Source Files

Note that Example C is a resource database, not an application. It does not contain `Starter.c`. Since Code Warrior can not create a database without a code0 resource, Example C contains a stubbed out __Startup()__ function (see `stub.c`).

To test this project, first load ExampleA.prc into POSE or onto a HandEra 330. Run ExampleA and notice the icons and Globe Bitmap. Now load up its Sibling Resource Database, `ExampleC.prc`. Then run ExampleA again. Both the icons and Globe bitmap should now appear sharper.

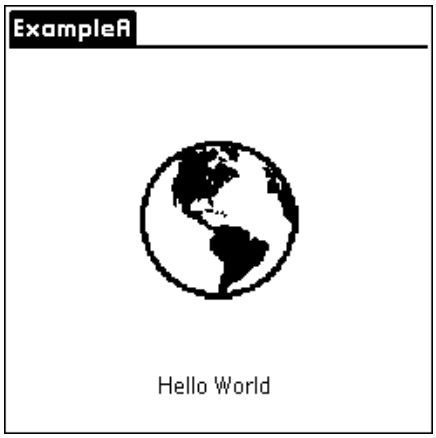

Figure 4-18. Before

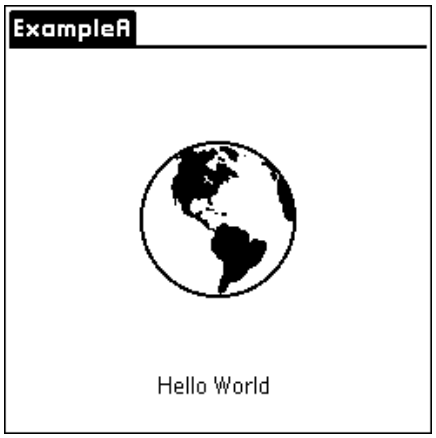

Figure 4-19. After

Chapter 5: High Resolution Forms

This chapter provides the steps necessary to assist developers in creating forms that take advantage of the higher resolution screen available on the HandEra 330, while maintaining compatibility with other Palm Powered devices. Unlike the steps discussed in the previous chapter, these will involve code changes and special API (Application Program Interface) calls.

VGA Extension

The HandEra 330 includes an extension to the standard Palm OS for managing all of the higher-resolution graphics issues called the **VGA Extension –** which is the code that implements the support for the QVGA screen.

Application developers should refer to the HandEra Programmer's Reference Manual for a complete list of the API calls supported by the VGA Extension.

Before calling any of the new functions, an application needs to ensure that the VGA Extension itself is present. Attempting to make VGA Extension calls on a non-HandEra device will crash the application – this is an inherent limitation of any Palm OS extension. The application should make a `FtrGet` function call to determine if the extension is present.

```
UInt32 version;
if FtrGet(TRGSysFtrID, TRGVgaFtrNum, &version) == 0)
{
  // Extension is present
}
```

Screen Modes

By default, applications are assumed to be **legacy applications,** created with a maximum form size of 160x160 pixels.

In order to change the screen mode, the VGA Extension includes the function call `VGASetScreenMode`. With this call, developers can set their application to either of the following screen modes:

- Scale To Fit
- 1 to 1 Mapping

Developers should use *Scale To Fit* mode with 160x160 forms, and *1 to 1 Mapping* mode with higher resolution forms.

❖ Note: The function `VGASetScreenMode()` must be called before the form is loaded.

```
VgaSetScreenMode(screenMode1To1, rotateModeNone);
FrmGotoForm(NewForm);
```

Screen Rotation

The VGA Extension also supports screen rotation. Forms can be displayed at 90°, 180°, 270° angles from the origin. The second argument of the API Call `VGASetScreenMode()` allows a rotation to be specified. Rotation is only valid for *1 to 1 Mapping* mode.

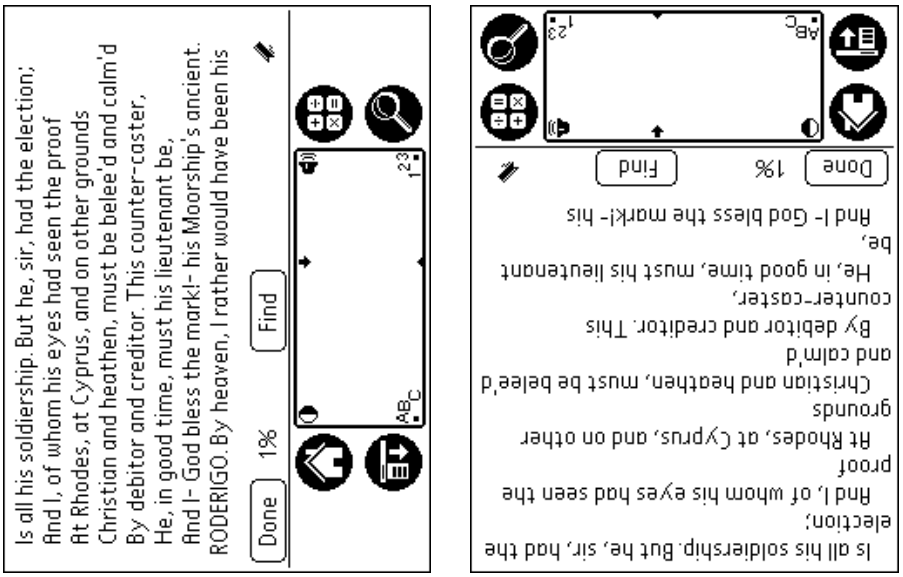

Figure 5-1. Screen Rotations

- Warning: Due to limitations of the Dragonball Vz processor, screen rotations other than 0° are not recommended for applications using animation.

Higher Resolution Forms

Higher-resolution forms refer to forms larger than 160x160 pixels. By default, the higher resolution forms used with the HandEra 330 should be 240x240 pixels. These forms are loaded in the upper 240x240 pixels of the LCD. The lower 240x80 pixels are used by the Silkscreen. A discussion of the Silkscreen, and its effect on form size, can be found in the next section. There are two methods to create higher resolution forms on the HandEra 330:

- Program Method
- Resource Method

❖ **Note:** Constructor v1.6 does no support forms larger than 160x160. However, .R files can be used with Code Warrior to overcome this problem. GNU tools does not have the same limitation as Constructor. These tools can be used to create 240x240 forms in the .rcp file.

Program Method

This method involves no changes to the application's resources as they exist in the .PRC file. Instead, the application developer is responsible for relocating and resizing the forms and the all the resources within the form. This should occur after the form has been initialized and before the form is drawn. The examples in the SDK have standardized on resizing the form on the `frmOpenEvent` event.

Obviously, this increases the workload for a developer. It requires more code to reposition each resource on the form. Fortunately, the VGA Extension does provide a function call to ease the workload – `VgaModifyForm`. This dynamically repositions and resizes all of a form's objects for the new screen size.

By using the Program Method, an application can use the same form and event-handling code for the application, whether it is running on a HandEra 330 or any other Palm Powered device.

Almost all the examples in the SDK including Example D provide an example of how an application can programmatically create higher-resolution forms.

Resource Method

Applications using this method contain higher-resolution versions of its 160**x**160 forms. As contrasted with the Program Method, these forms are created during compile time, through the use of .R files. The application itself is responsible for loading the appropriate version of a form, based on whether or not the application is running on a HandEra 330.

Example X in the HandEra SDK provides an example of how developers can modify the application's resource files to create the higher-resolution forms.

New QVGA Resource Requirement

Applications with forms larger than 160x160 need to include a resource of type 'sKst' and ID of 1000. This resource is new to the Palm Powered platform. The presence of this resource informs the Application Preference Panel that the application uses forms larger than 160x160 – these applications are displayed as "High Res". Although the built-in launcher does not use this feature, 3rd party launchers and utilities could categorize these apps based on this resource.

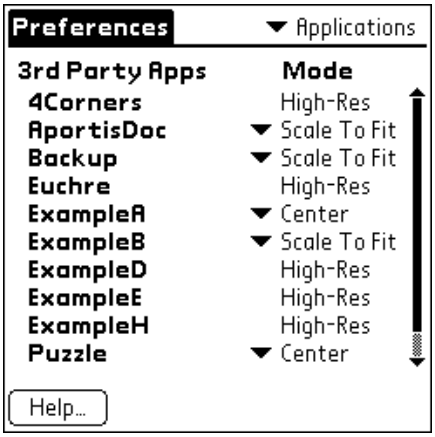

Figure 5-2. Application Preference Panel

Unfortunately, Constructor cannot create 'sKst' resources. However, this resource can be created in a .R file. Simply add HandEra.r, which can be found in the include subdirectory of the SDK, to the Code Warrior Project. Developers using PilRC with the GNU tools need to include the following line to the .RCP file.

```
HEX "sKst" ID 1000 0x00 0x00 0x00 0x00
```

Text and Fonts

This section applies to all text present in the user interface — whether entered by the user or generated by the application itself.

Starting with version 3.0, the Palm OS included support for the function `FontSelect`, which displays a dialog that allowed a user to select one of three fonts for alphanumeric generation:

- stdFont
- boldFont
- largeBoldFont

❖ **Note:** Additional fonts are also included within the OS, but they are not presented in the dialog. For example, largeFont is now included with the Palm OS.

If an application is running in *Scale To Fit* screen mode on a HandEra 330, `FontSelect` generates the same dialog as seen in Figure 5-3.

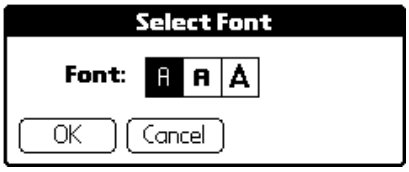

Figure 5-3. Fonts

However, when an application is running in *1 to 1 Mapping* mode on a HandEra 330, the number of available fonts displayed in the `FontSelect` dialog increases. Each of the original fonts, including largeFont, has an equivalent higher-resolution sibling that is 1.5 times its size. Figure 5-4 shows the new font dialog.

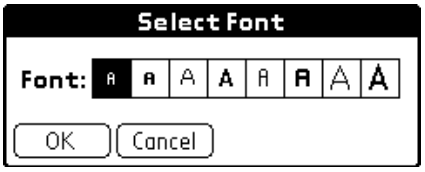

Figure 5-4. New Font Dialog

The dialog displays the following fonts (listed left to right):

- stdFont
- boldFont
- largeFont
- largeBoldFont
- VGA stdFont
- VGA boldFont
- VGA largeFont
- VGA largeBoldFont

The default font used with *1 to 1 Mapping* mode is VGA stdFont. Applications running in *1 to 1 Mapping* mode can make the API call `VgaBaseToVgaFont` to convert the original font ID's to their higher-resolution equivalent. For example, the expanded version of standard font can be set by calling:

```
FntSetFont(VgaBaseToVgaFont(stdFont));
```

❖ **Note:** A font character is made up of an array of pixels. In the higher resolution screen modes, the original font characters actually appear smaller, since the pixel size itself is smaller. An application can use any mix of original or expanded font characters in its display.

Rotate Common Dialog

New to the Palm Powered platform is the VgaRotateSelect dialog box (see Figure 5-5). This dialog provides a common interface for applications that support screen rotation. For more information on how to use this dialog, see the Programmer's Reference Manual.

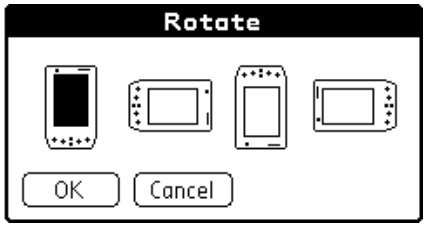

Figure 5-5. Rotate Common Dialog

Example D – High Resolution Form

Example D provides examples of …

- *High-resolution icons*
- *High-resolution bitmaps*
- *High-resolution forms*
- *QVGA Identifier resource*
- *Additional fonts*
- *Screen rotation*
- *vgaFormModify API call*

Example D builds on Example B by displaying high-resolution forms and additional fonts. Example D uses the API call `vgaFormModify` to enlarge the 160x160 Main form to the desired size.

Perform the following steps:

1. Copy the files from the `\Source\ExampleD` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleD Project**.

   ❖ **Note:** Example D has been coded in order to run on non-HandEra 330 devices as well. Example D checks for the presence of the VGA Extension and uses it only if present. Example D should look identical to Example A and Exanple B when run on non-HandEra 330 devices.

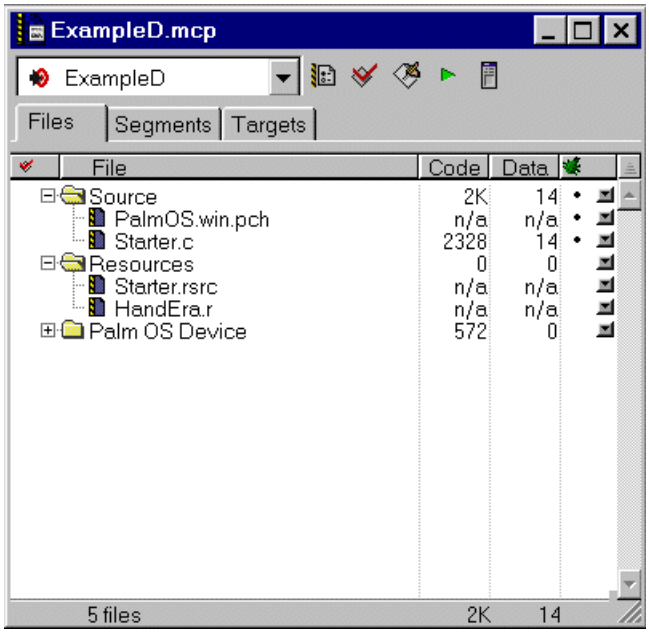

Figure 5-6. `HandEra.r` file

Note that Example D also includes the file `HandEra.r`. (See Figure 5-6). This file contains the QVGA Identifier resource ‘sKst’.

```
type 'sKst' {

    //Currently no data is stored in resource, need dummy
    //otherwise resource is not linked in
    unsigned longint;
};

resource 'sKst' (1000, "HandEra Aware") {
   0x00000000;
};
```

Example D allows the user to rotate the screen mode by providing a menu list of screen options as seen in Figure 5-7. This menu list does not display when run on non-HandEra 330 devices.

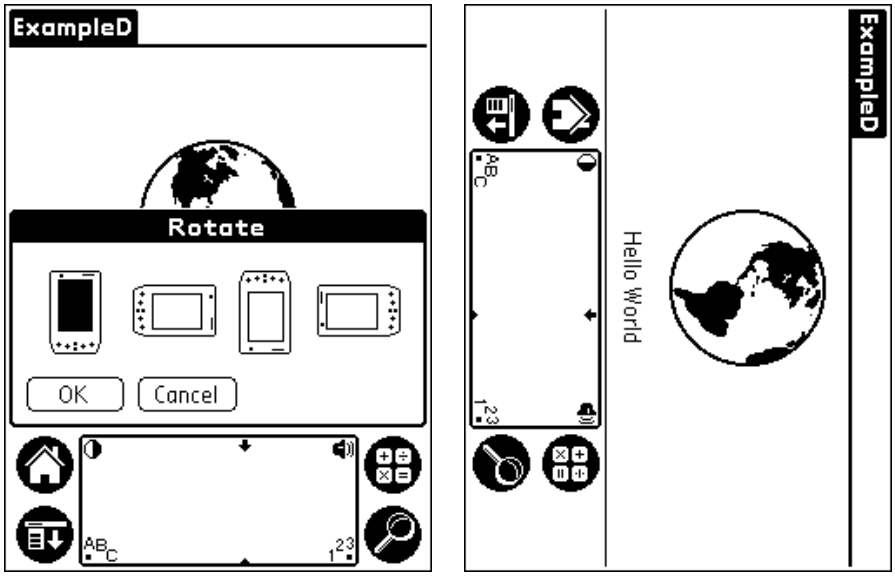

Figure 5-7. Rotate

Currently, the Palm OS supports the following fonts: stdFont, boldFont, largeFont, symbolFont, symbol11Font, symbol7Font, ledFont, and largeBoldFont. HandEra 330 provides these fonts as well as an equivalent font that is 1.5 times larger than each of the standard fonts. Example D uses the `FontSelect()` dialog to change the fonts. Since Example D runs in *1 to 1 Mode*, the user is presented additional fonts from which to choose. In the previous examples, font selection was limited to the three built-in system fonts provided by the Palm OS. Example D, however, allows the user to select from eight fonts. Refer to Figure 5-8.

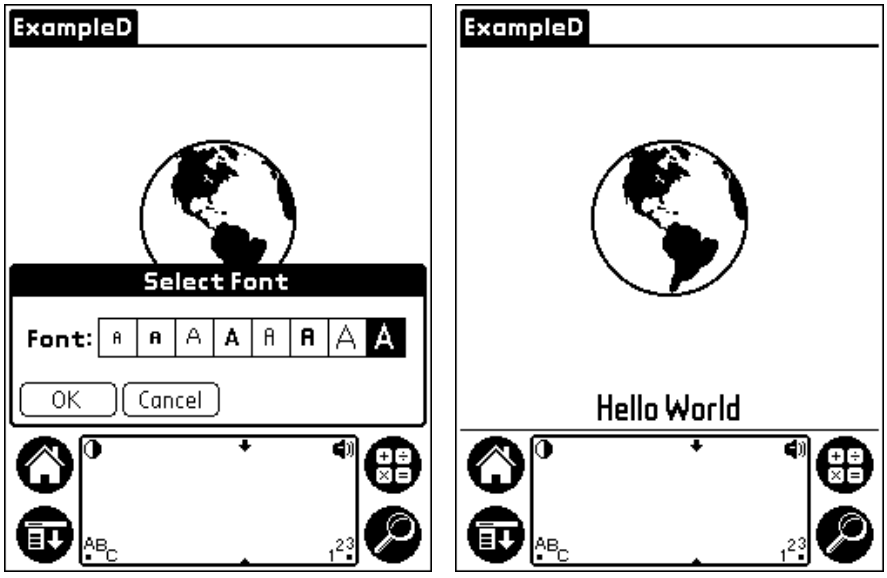

Figure 5-8. Fonts

When the user taps the Minimize Arrow in the Silkscreen, the Silkscreen will be redrawn in Minimized mode. The application itself doesn't respond to the change. To be fully QVGA-compliant, however, it should be able to handle changes to the Silkscreen. The following chapter discusses how to handle display extent change events within an application.

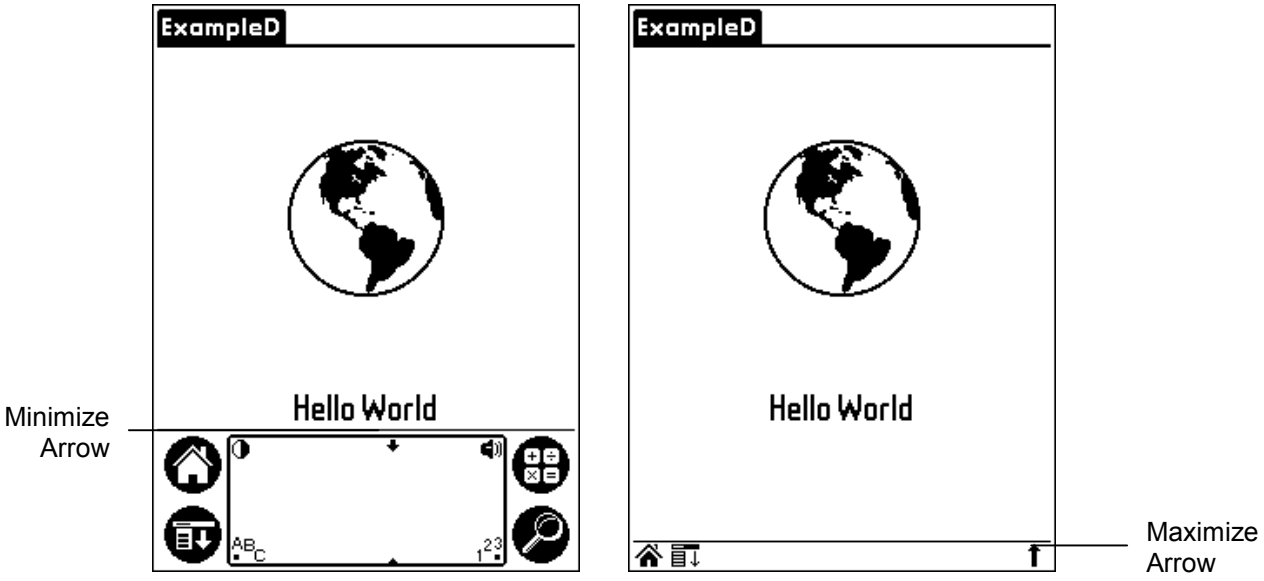

Figure 5-9. Minimized Silkscreen

Chapter 6: Display Extent

This chapter discusses the issues involved with the display extent. For previous generations of Palm Powered devices, the display extent was fixed – its height and width was 160 pixels. However, because QVGA devices support both a minimized and maximized virtual silkscreen, as well as screen rotation, the display extent is dynamic. Applications need to be able to resize based on the available screen real estate.

Figure 6-1 shows how the Address Book has been modified to take advantage of the area vacated when the Silkscreen is minimized. The application determines the size of its form, repositions the buttons to the bottom of the display area, and creates the appropriate number of entry lines.

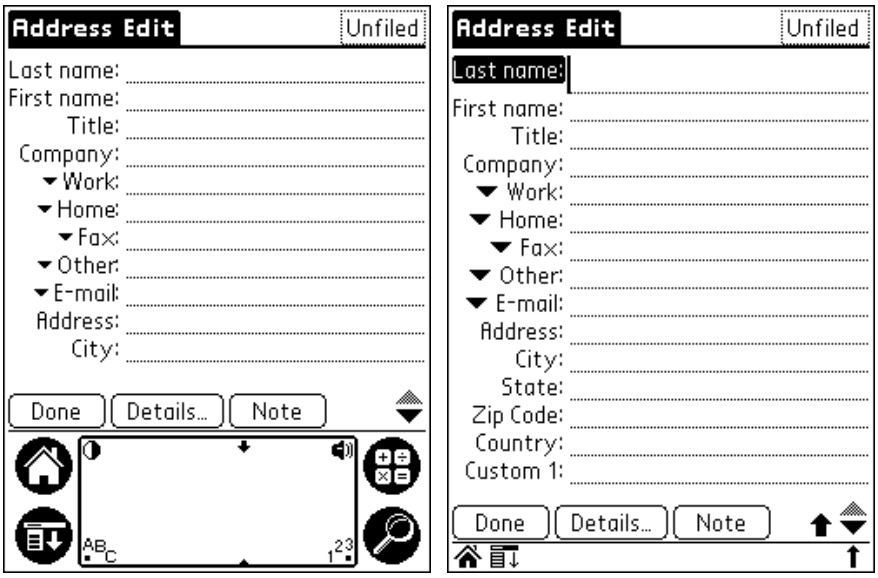

Figure 6-1. Silkscreen

Sizing the Form

In order to support a dynamic display extent, the application must be able to determine how much of the screen area is currently available for use by a full-size form. This can be accomplished by using standard Palm API calls, such as in the following function.

```
void SizeForm(FormPtr *frmP)
{
   RectangleType rect;

   rect.topLeft.x = 0;
   rect.topLeft.y = 0;
   WinGetDisplayExtent(&rect.extent.x, &rect.extent.y);
   WinSetWindowBounds(FrmGetWindowHandle(frmP), &rect);
}
```

Event Handling

QVGA-aware applications need to respond to the following event:

frmOpenEvent

When a `frmOpenEvent` event is received, the application needs to call the SizeForm() function mentioned above as its first step. Resizing the form could be preformed on the `frmLoadEvent` instead, however, HandEra has standardized on the `frmOpenEvent`.

displayExtentChangedEvent

`displayExtentChangedEvent` is a new event code for the Palm Powered platform. This event is generated when the user minimizes or maximizes the Silkscreen, or when the application changes the screen rotation. The application should then resize the form.

```
Boolean MainFormHandleEvent(EventPtr eventP)
{
    Boolean handled = false;

    switch (eventP->eType)
    {
        case frmOpenEvent:
            SizeForm(FrmGetActiveForm());
            // Add any other initialization code
            FrmDrawForm(FrmGetActiveForm());
            handled = true;
            break;

        case displayExtentChangedEvent :
            SizeForm(FrmGetActiveForm());
            // Add any other event handling code
            FrmDrawForm(FrmGetActiveForm());
            handled = true;
            break;

        default:
            break;
    }
    return(handled);
```

Example E – Display Extent

Example E provides examples of ...

- *High-resolution icons*
- *High-resolution bitmaps*
- *High-resolution forms*
- *QVGA Identifier resource*
- *Additional fonts*
- *Screen rotation*
- *vgaFormModify API call*
- *Display extent event-handling*

Perform the following steps:

1. Copy the files from the `\Source\ExampleE` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleE Project**.
3. Install and launch `ExampleE.prc` on POSE or a HandEra 330.

Example E builds on Example D by including support for display extent events. Whenever the user minimizes or maximizes the silkscreen, the application resizes the current form and recenters the globe (see Figure 6-2).

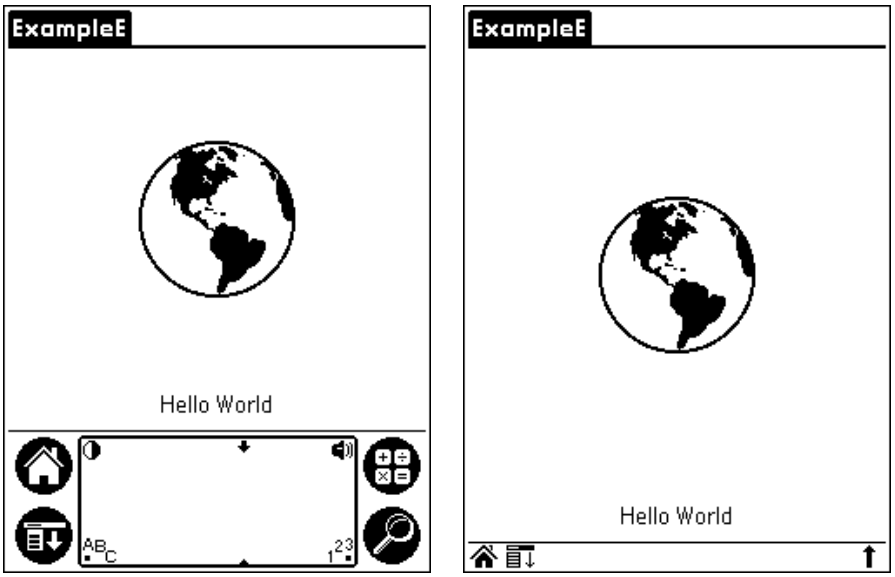

Figure 6-2. Minimize

Note Example E differs from Example D by only four lines of code. The following was added to the event-handling loop to support display extent events:

```
case displayExtentChangedEvent :
    MainFormResize(FrmGetActiveForm(), true);
    handled = true;
    break;
```

Note that the state of the Silkscreen will not change, even if the current form rotates the screen. If the Silkscreen was minimized before the rotation, it will be minimized after the rotation.

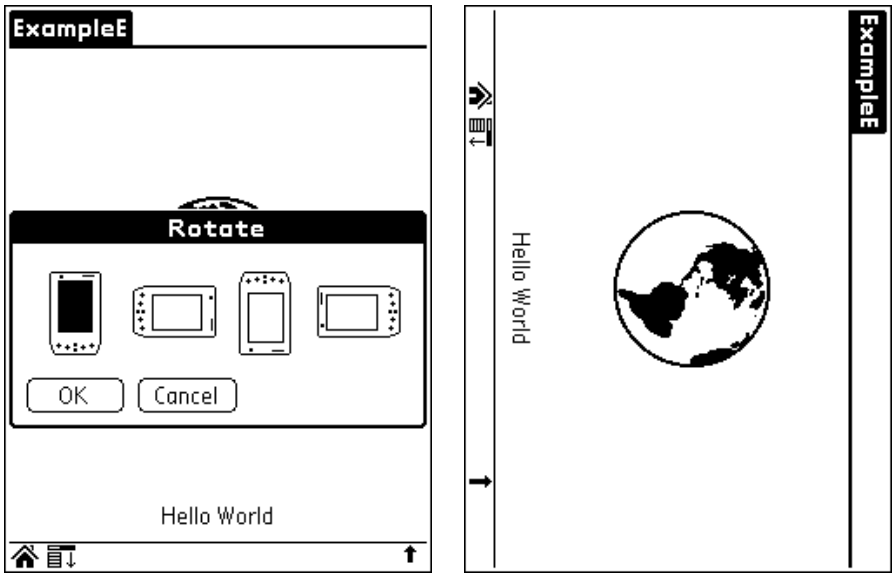

Figure 6-3. Rotate

Chapter 7: Jog Wheel and Auxiliary Button

There is growing support within the Palm Powered® platform for **one-hand** applications. These applications allow a user to hold and operate the Palm Powered device with one hand. One-hand applications must observe the following conventions:

- A user must be able to navigate via the hardware buttons between forms and dialogs.
- Pen taps and graffiti stroke events are accepted, but not required, for navigation.

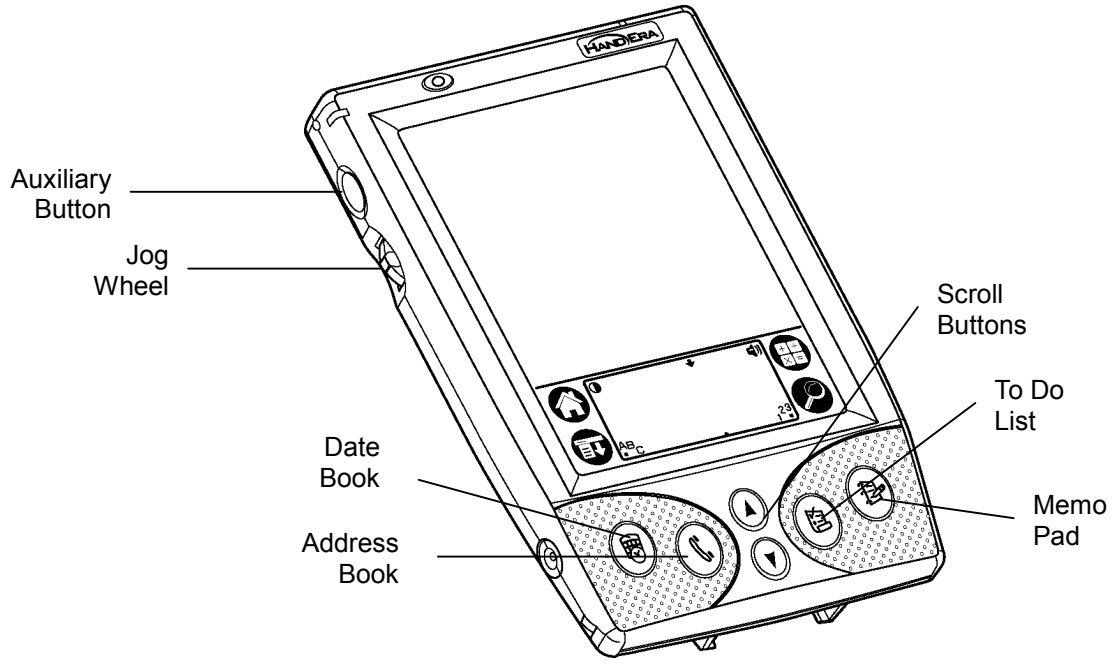

Figure 7-1. Hardware Buttons

To help provide support for one-hand applications, the HandEra 330 features two additional hardware buttons. The **Auxiliary** button is located on the left-hand side of the unit. The **Jog Wheel** is located just below the Auxiliary button on the HandEra 330. Unlike the other hardware buttons, the Jog Wheel can be rotated up/down, or pressed in. Refer to Figure 7-2.

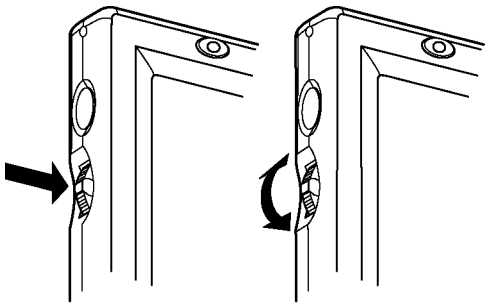

Figure 7-2. Jog Wheel Actions

Application developers do not need to create special code or include new API calls to process Jog Wheel or Auxiliary button input. The event manager returns a `keyDownEvent` to an application whenever any of the hardware buttons, including the Auxiliary button and Jog Wheel, is pressed. The application is responsible for processing the appropriate key code stored in the event record.

- ❖ Note: Key definitions can be found in the header file "TrgChars.h" in the HandEra 330 SDK Include folder.

Button Key Values

By default, the Jog Wheel and Auxiliary button actions are defined as follows:

Action	***Key***
Jog Wheel rotated up	vchrPrevField
Jog Wheel rotated down	vchrNextField
Jog Wheel pressed	chrCarriageReturn
Auxiliary button pressed	chrEscape

Handling these keys is the same as handling any other keyDownEvent.

```
case keyDownEvent:
     switch (eventP->data.keyDown.chr)
     {
         case vchrPageUp :
             handled = HandlePageUp();
             break;
         case vchrPageDown :
             handled = HandlePageDown();
             break;
         case vchrPrevField :
             handled = HandlePrev();
             break;
         case vchrNextField :
             handled = HandleNext();
             break;
         case chrCarriageReturn :
             handled = HandleReturn();
             break;
     }
```

If `chrEscape` is not handled within the application's event loop processing, the key is translated within the context of the application.

- ♦ For modal forms in an application: the key is equivalent to tapping on the default button of a dialog.
- ♦ For non-modals forms in an application: the key is translated as vchrLaunch.

To change the behavior of `chrEscape`, the application will need to handle the key as part of its event-handling loop.

Programming Examples in this Chapter

Example H demonstrates how a one-hand application is implemented. It also demonstrates how to reposition the Graffiti shift indicator during display extent event-handling.

Example I details how changes were made to the standard version of MemoPad to make it HandEra 330 - compliant.

Example H – Tables

Example H provides examples of ...

- *High-resolution icons*
- *High-resolution bitmaps*
- *High-resolution forms*
- *Resizing forms*
- *Repositioning Graffiti shift indicator*
- *Resizing tables*
- *Additional fonts*
- *Screen rotation*
- *VgaFormModify function call*
- *Repositioning controls*
- *Resizing controls*
- *One-hand operation*

Perform the following steps:

1. Copy the files from the `\Source\ExampleH` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleH** Project.
3. Install and launch `ExampleH.prc` on POSE or a HandEra 330.

Example H is an example of a one-hand application. The user can rotate the Jog Wheel up or down to highlight to move the highlight bar to a different row in the table. Pressing the Jog Wheel will cause the highlighted row to be selected. (See Figure 7-3).

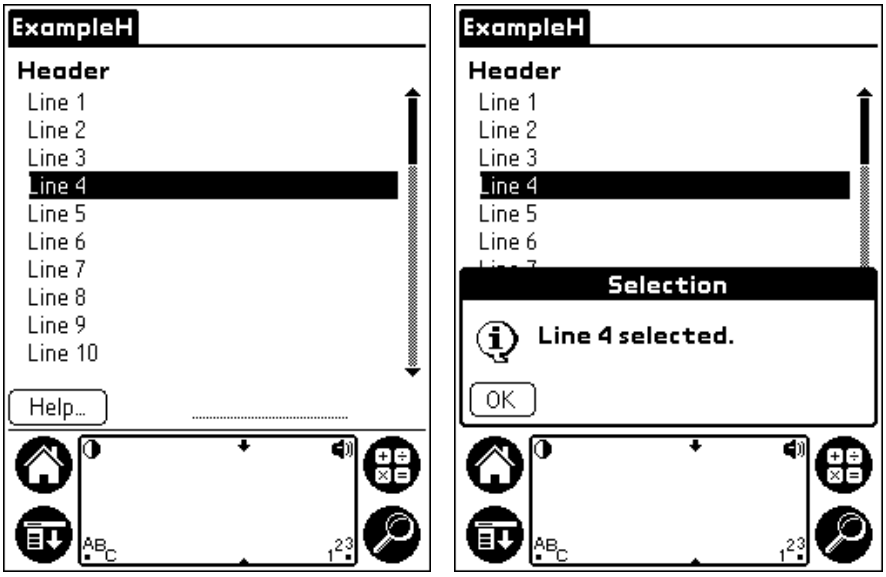

Figure 7-3. Selecting a line.

Pressing the Auxiliary button will cause Example H to exit, and will cause the Application Launcher to run. The code to handle the one-hand keys is fairly straightforward. Inside the event-handling loop, the following code is included for `keyDownEvent` events.

```
switch (eventP->data.keyDown.chr)
{
    case vchrPrevField :
        HandlePrevKey();
        handled = true;
        break;
    case vchrNextField :
        HandleNextKey();
        handled = true;
        break;
    case chrCarriageReturn :
        if (currentSelection != -1)
           HandleSelectItem(currentSelection);
        handled = true;
        break;
}
```

Since the default behavior of the Auxiliary button is used, the application doesn't have to include any special handling of `chrEscape`.

Example H also demonstrates how tables are used with QVGA devices. It is important to note that the number of rows for table resources cannot be increased dynamically. In order to create an application that can run on QVGA and non-QVGA devices, it is important to create a table with the maximum number of rows needed for the QVGA screen. When the application is run on a non-QVGA device, the application will need to programmatically reduce the number of rows.

Examine the Table for Example H (Figure 7-4). The number of rows is set to 20. The application determines how many rows can actually be displayed by dividing the table's height by the font height. If the result is 16, for example, then rows 17-20 are disabled by calling `TblSetRowUsable` with the third parameter set to `false`.

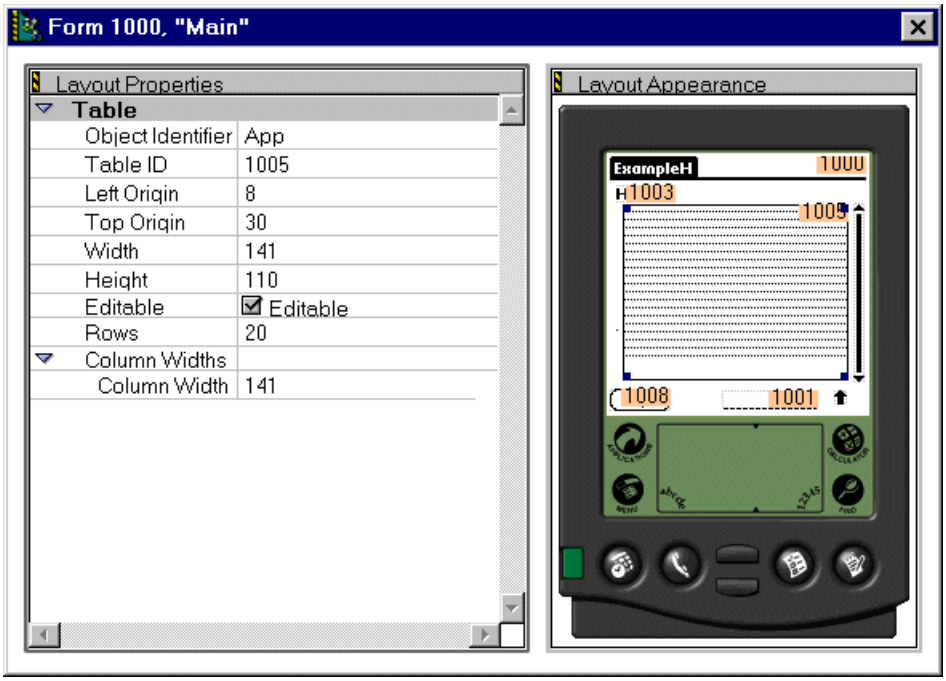

Figure 7-4. Example H Table

Example H also includes support for handling display extent events. It resizes the current form whenever it receives `displayExtentChangedEvent` events. The code is also able to handle the repositioning of the Graffiti shift indicator (GSI), bottom-justifying the control when the Silkscreen is minimized (see Figure 7-5).

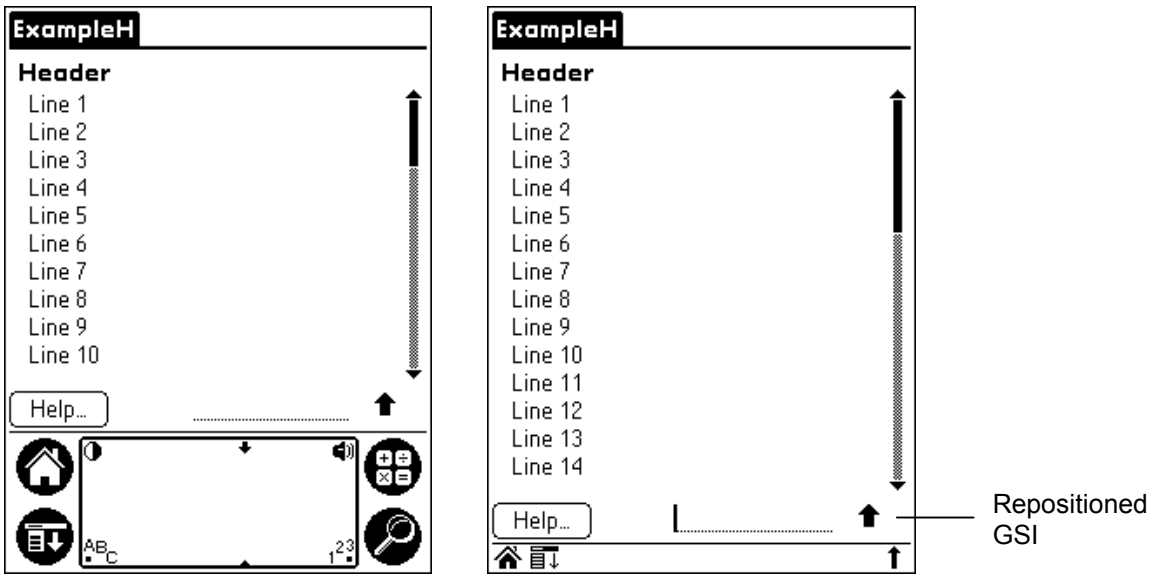

Figure 7-5. Graffiti Shift Indicator (GSI)

There is additional work required to move Graffiti Shift Indicators (GSI). `FrmGetObjectIndex()` does not work on GSI's, therefore, this task must be performed manually by the application. In order to get the index to the GSI object, the following code snippet must be used:

```
UInt16 i, numObjects;

numObjects = FrmGetNumberOfObjects(frmP);

for (i=0; i<numObjects; i++)
{
    if (FrmGetObjectType(frmP, i) == frmGraffitiStateObj)
        return(i);
}
return(-1);
```

Once the application has located the object index of the GSI, it is then possible to reposition the object similar to other objects in the form.

Example I – MemoPad

Example I provides examples of ...

- *High-resolution icons*
- *High-resolution bitmaps*
- *High-resolution forms*
- *Silkscreen event-handling*
- *Additional fonts*
- *One-hand operation*
- *vgaFormModify function call*

Example I demonstrates the steps taken by HandEra to convert the MemoPad project provided in Palm's SDK to take advantage of the features of the HandEra 330.

Perform the following steps:

1. Copy the files from the `\Source\ExampleI` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the MemoPad Project.
3. Install and launch `MemoPad.prc` on POSE or a HandEra 330.

Example I is the MemoPad application that is shipped with the HandEra 330. This section describes that changes that were made to the version of MemoPad that ships with the PalmOS 3.5 SDK to become HandEra 330 - compliant.

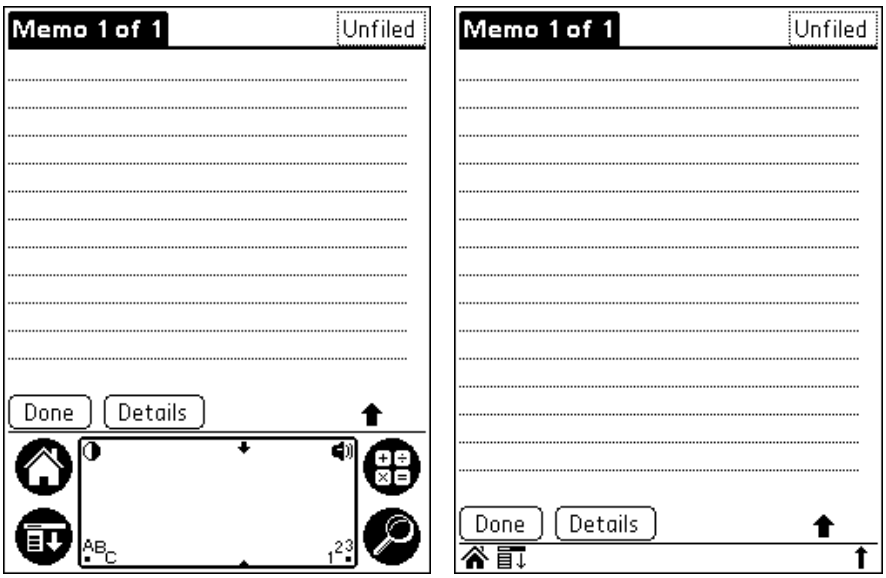

Figure 7-6. Memo Pad

Changes required to convert MemoPad to a HandEra aware application

All source code changes were limited to MemoMain.c and are surround by the following comments: /* TRG */ and /* end TRG */.

Resource changes

1. Added 32x32 and 22x15 icons.
2. Increased table rows to 25 for ListView table.
3. Added HandEra.r resource file to indicate program uses forms larger than 160x160.
4. Fixed typo in AlertID 2000 and removed extra space before first period.

Support for the higher-resolution screen

Switching to *1 to 1 Mapping* mode (240x240) gives access to the additional fonts and allows support for the minimized silkscreen.

1. Added `#include "vga.h"` to the top of the source file.
2. Added the global variable `vgaExists` (Boolean).
3. Added a check for VGA Extension presence in `StartApplication()` and set the `vgaExists` global.
4. Added a call to `VgaSetScreenMode()` in `StartApplication()`.
5. Added call to `VgaFormModify()` in `EditViewHandleEvent` and `ListViewHandleEvent frmOpenEvent` cases.
6. Added `VgaFormModify()` to the modal forms in: `DetailsInit()`, `DeleteRecord()`, `EditViewDeleteRecord()`, and `PreferencesInit()`.

Font issues

Cleaned up font issues. Added calls to `VgaTableUseBaseFont()` for locations where the table is using small fonts.

1. In `ListViewHandleEvent() frmOpenCase`, added `VgaTableUseBaseFont(, !VgaIsVgaFont())`.
2. In `SelectFont()`, added `VgaTableUseBaseFont(,!VgaIsVgaFont())` if it is ListView.
3. In `MemoLoadPrefs()`, added `VgaBaseToVgaFont()` to get the correct default font.

Support for the virtual Silkscreen

1. Added functions `EditViewResizeForm()` and `ListViewResizeForm()`.

2. Called the form's resize function from `frmOpenEvent` and `frmGotoEvent` to handle the case where the app is launched with the silk minimized.

3. Added `vgaChangedEvent` to `ListViewHandleEvent` and `EditViewHandleEvent` and call the form's resize function.

Support for one-hand operation

1. Add handler for `chrCarriageReturn` and `chrEscape` to `EditViewHandleEvent()` which returns to the ListView.

2. Add handlers for `chrCarriageReturn`, `vchrNextField`, and `vchrPrevField` to `ListViewHandleEvent()`.

Chapter 8: VFS and Expansion Managers

Palm OS 4.0 introduces two system extensions to help manage expansion slots for Palm Powered devices.

The **Virtual File System (VFS)** Manager is a layer of software that manages all installed file system libraries. It provides a unified API to application developers while allowing them to seamlessly access many different types of file systems —such as VFAT, HFS, and NFS—on the different types of media.

The **Expansion Manager** is an optional system extension that adds support for hardware expansion cards on Palm Powered devices. The Expansion Manager's primary function is to manage device slots and the drivers associated with those slots. Individual slot drivers—which are provided by device manufacturers—provide support for various expansion card types including SD and MMC cards, Compact Flash, Sony's Memory Stick, and others.

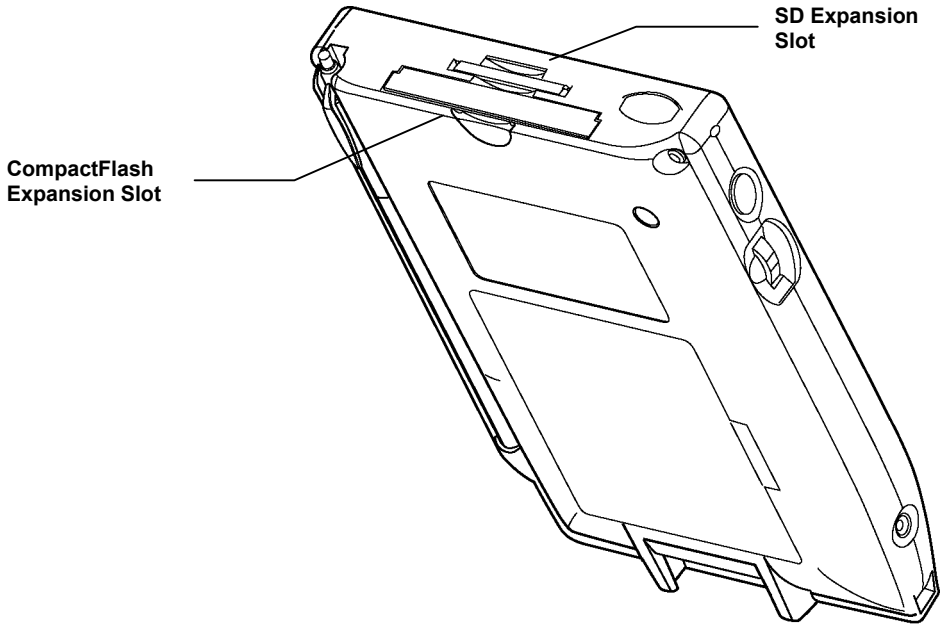

Figure 8-1. Expansion Slots

The HandEra 330 includes two expansion slots. The smaller slot supports Secure Digital (SD) cards and Multimedia (MMC) cards. The larger slot supports CompactFlash (CF) cards.

The HandEra 330 provides compatibility with the Palm OS 4.0 system extensions for expansion slots. It fully implements the VFS API as documented in Chapter 51: Virtual File System Manager of the *Palm OS API Reference* in the Palm OS 4.0 SDK, with the exception of the following functions:

- VFSInstallFSLib() returns *expErrNotOpen* unless creator is 'ffsd' in which case returns *errNone*.
- VFSRemoveFSLib() always returns *errNone*.
- VFSCustomControl() returns *expErrUnsupportedOperation*.
- VFSVolumeFormat() does not support alternate file systems.
- VFSVolumeMount() does not support alternate file systems.
- VFSDirEntryEnumerate() does not support nested directory searches.

The HandEra 330 fully implements the Expansion API as documented in Chapter 30: Expansion Manager of the *Palm OS API Reference* in the Palm OS 4.0 SDK, with the exception of the following functions:

- ExpSlotDriverInstall() returns *expErrUnimplemented*.
- ExpSlotDriverRemove() returns *expErrUnimplemented*.
- ExpSlotLibFind() if slot is valid, returns *errNone* and passed back a dummy library reference, otherwise returns *expErrInvalidSlotRefNum.*
- ExpSlotRegister() returns *expErrUnimplemented*.
- ExpSlotUnregister() returns *expErrUnimplemented*.
- ExpCardInserted() returns *expErrUnimplemented*.
- ExpCardRemoved() returns *expErrUnimplemented*.

Chapter 9: Audio

Audio Hardware

The HandEra 330 offers enhanced sound capabilities. It includes a microphone at the top of the unit for speech-recording. It also includes a higher-fidelity speaker and drive electronics which allow for improved audio performance. This speaker can play speech quality sound and deliver significant volume for alarms.

The HandEra 330 produces sound by creating a modulated PWM signal. The output from the CPU is then run through a low pass filter that removes the PWM carrier frequency and leaves only frequencies between 200 Hz and 4 kHz un-attenuated. After filtering, the audio output signal is acted upon by a variable gain audio amplifier.

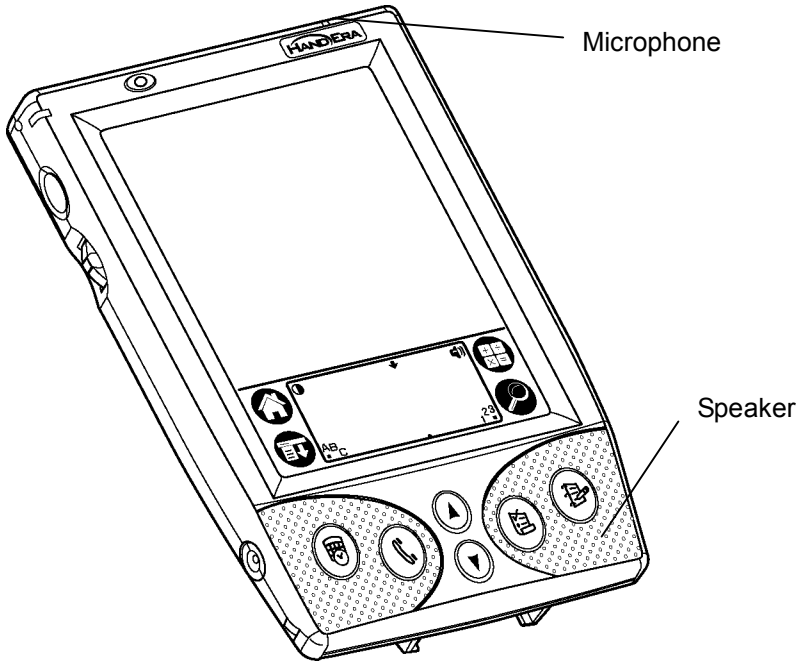

Figure 9-1. Audio

Audio Extension

The HandEra 330 includes an extension to the standard Palm OS for managing all of the audio issues called the **Audio Extension**.

Application developers should refer to the HandEra Programmer's Reference Manual for a complete list of the API calls supported by the Audio Extension.

Before calling any of the new functions, an application needs to ensure that the Audio Extension itself is present. Attempting to make Audio Extension calls on a non-HandEra device will crash the application. The application should make a `FtrGet` function call to determine if the extension is present.

```
UInt32 version;
if FtrGet(TRGSysFtrID, TRGAudioFtrNum, &version) == 0)
{
  // Extension is present
}
```

The API supports two basic sound types.

1. DTMF Tones
2. Waveform Audio

DTMF Tones

DTMF Tones (Dual Tone Multi-Frequency) are the touch tones used for dialing in telephone systems. The tones are the amplitudes of two sine waves of different frequencies – one a low tone and one a high tone. The low tone specifies the row and the high tone specifies the column of a standard telephone keypad.

The Audio API includes functions to play DTMF tones. This allows the HandEra 330 to function as a phone dialer.

Figure 9-2. Telephone Keypad

Waveform Audio

Waveform audio is the sound format commonly used with Microsoft Windows. Waveform filenames include the .WAV extension. The waveform refers to the vibrations associated with translating sound into electric current. The amplitude of the waveform is its loudness. The frequency is the pitch.

The Audio Extension supports the recording of waveform audio files with a sampling rate of 8 KHz. The sampling rate refers to the maximum frequency of sound that can be digitized for a given sample. The sampling size for HandEra waveform audio files is 8-bit mono. The size determines the dynamic range of a sample.

The Audio Extension supports the playback of waveform audio files with sampling rates of 44.1 KHz, 22.05 KHz, 11.025 KHz, and 8 KHz. The only sampling size supported is 8-bit mono.

Figure 9-3. Record

Example J – Voice Recorder

Example J provides examples of …

- *High-resolution icons*
- *High-resolution forms*
- *Audio API calls*
- *Auxiliary button press-and-hold*
- *High-resolution bitmaps*
- *vgaFormModify API call*
- *VFS API calls*

Example J demonstrates how to implement the basic record and playback functions from the Audio API. It also demonstrates how to use the VFS API.

Perform the following steps:

1. Copy the files from the `\Source\ExampleJ` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleJ** Project.
3. Install `ExampleJ.prc` on a HandEra 330. The current version of POSE does not support the Audio API.

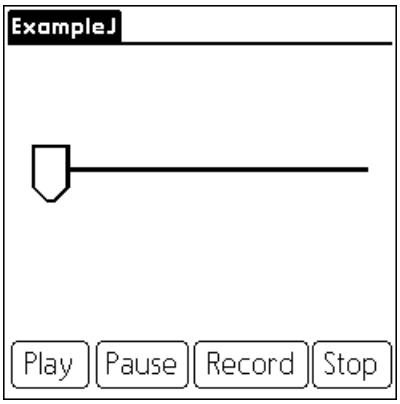

Figure 9-4. Example J Main Form

Example J is a project that implements a rudimentary digital speech recorder. It creates waveform audio files with a sampling rate of 8 KHz and a sampling size of 8-bit mono. The application uses the API functions `AudioRecordData` and `AudioPlayData` to record and playback the audio.

The Main Form for Example J also demonstrates how to handle `audioProgressEvents`, events generated while the functions `AudioRecordData` and `AudioPlayData` are in progress. When the Main Form receives an `audioProgressEvent`, it updates the progress bar to give the user visual feedback. During `AudioRecordData`, `audioProgressEvents` are generated at the rate of two per second. During `AudioPlayData`, `audioProgressEvents` are generated at the rate of four per second.

Example J stores the audio file on a memory card in the CompactFlash slot. The application uses the API functions `VFSFileRead` and `VFSFileWrite` to read and write from the memory card.

Chapter 10: Programming Summary

Checklist for QVGA-aware Applications

In order for an application to be considered QVGA-aware, it should include the following:

- High resolution icons and bitmaps must be included
- High-resolution forms must be supported, preferably through use of the `vgaFormModify` API call.
- QVGA identifier resource must be included
- Display extent event-handling support must be included
- One-hand operation support can optionally be added

Compatibility Issues

Backward Compatibility

The enhanced features of the HandEra 330 are implemented as extensions to Palm OS version 3.5. Developers interested in writing applications that can run on multiple Palm Powered devices should determine which features are supported to ensure backward compatibility. At the very least, applications should exit gracefully if a necessary feature is not present.

Checks of the Palm OS version number will not reliably ensure that a feature is present on the device. Programmatically, developers should use the feature manager to determine if a particular feature is available on the device the application is running on. For a discussion on accessing the feature manager, refer to the *Palm OS Programmer's Companion*.

Forward Compatibility

Any implementation-specific information is provided as background explanations Developers should not use any of this information to hard-code data constants into their code. Instead, developers should rely on the Application Program Interface (API) provided for the HandEra 330. This allows applications to be compatible with future generation products.

❖ **Note:** HandEra is committed to the standards that define the Palm Powered platform. As new graphics standards emerge, HandEra will work to ensure that its API set complements that of the platform.

Design Practices

HandEra 330-aware applications should be developed following the same general user interface guidelines established for all Palm Powered devices along with these suggestions.

- The recommended resource values for width and height, as detailed by *Palm OS SDK Reference*, should be multiplied by 1.5 for the higher resolution screen modes.
- Buttons should be aligned with the bottom of the display area, above the virtualized Silkscreen.
- The application should use the same screen rotation between forms.
- The application should be able to be run successfully on different Palm Powered devices.
- It is preferred that the user control the minimizing/maximizing of the Silkscreen. The application should support different form sizes based on the state of the Silkscreen. If, however, an application does need to minimize or maximize the Silkscreen, it should save the current state of Silkscreen on entry and restore it on exit.

Chapter 11: Advanced Topics

The majority of application developers can skip the topics discussed in this chapter. It presents issues more of a concern for developers creating more unorthodox applications, such as hacks or app launchers.

Modifying the Silkscreen

The Silkscreen for the HandEra 300 is *virtualized*, so that the Graffiti areas and the icons are actually rendered on the bottom 240x80 pixels of the LCD. This is what allows support for the minimized and maximized forms of the Silkscreen.

It is also possible for applications to modify the Silkscreen area. One such example can be seen in Figure 11-1, which implements a pop-up keyboard over the Silkscreen.

Example F is the first of two examples to demonstrate the flexibility of this virtualized Silkscreen. This example displays a bitmap on the Silkscreen window. The changes to the silkscreen area are not persistent and are erased when the program is exited. Example G shows how to make permanent changes to the Silkscreen.

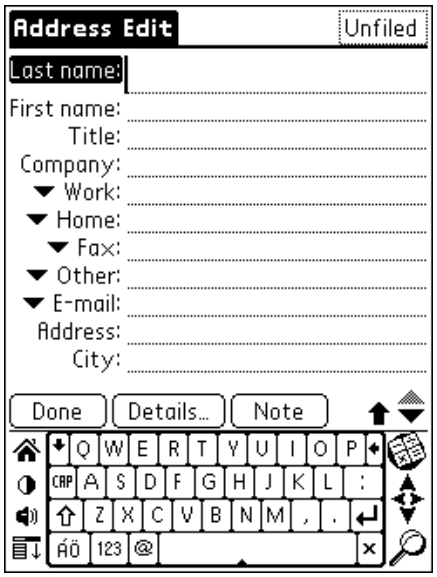

Figure 11-1. Silkscreen Keyboard

Example F – Silkscreen Bitmap

Example F provides examples of ...

- *High-resolution icons*
- *Silkscreen modification*

Perform the following steps:

1. Copy the files from the `\Source\ExampleF` directory of the SDK to a directory on a local drive.

2. Launch the **CodeWarrior IDE** and open the **ExampleF** Project.

3. Install and launch `ExampleF.prc` on POSE or a HandEra 330.

Drawing to the Silkscreen is as simple as retrieving a WinHandle and then drawing to the window using standard Palm window drawing routines. When the silkscreen is maximized, Example F displays a picture of the globe in the Silkscreen and when minimized, simply prints the text "Hello World" on the silkscreen.

The changes made to the Silkscreen by Example F are not permanent because the Silk Extension redraws the area on all program exits with the silkscreen template. Refer to the *HandEra 330 Programmer's Reference Manual* for a discussion on the complete Silk API. Example G demonstrates how to change the template.

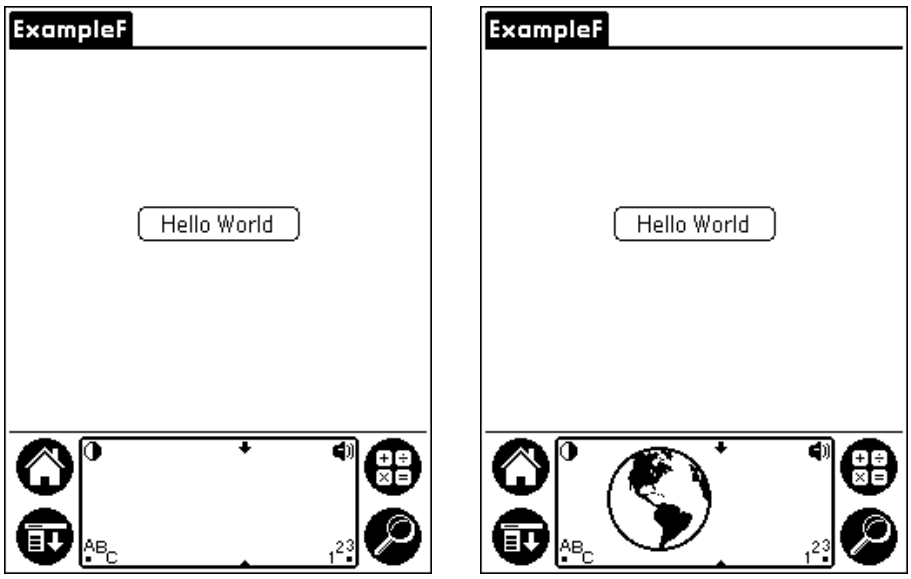

Figure 11-2. Silkscreen Bitmap

Example G – Silkscreen Buttons

Example G provides examples of …

- *High-resolution icons*
- *Silkscreen template modification*
- *Silkscreen button modification*

Perform the following steps:

1. Copy the files from the `\Source\ExampleG` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleG** Project.
3. Install and launch `ExampleG.prc` on POSE or a HandEra 330.

Example G demonstrates how to make permanent changes to the Silkscreen and how to add custom buttons.

Each time an application exits, the VGA extension sets the screen mode back to *1 to 1 Mode* and the Silk Extension redraws the Silkscreen. The Silkscreen is redrawn from a template that can be changed by the developer with a call to `SilkSetTemplateBitmaps()`. The Silk Extension also provides the call `SilkRestoreDefaultTemplate()` to restore the Silkscreen back to its default image and button list.

Associated with the Silkscreen are Graffiti areas and buttons. Example G adds the buttons '0'-'9' and '.' to the Silkscreen button list. Example G also removes buttons and graphics for the default numeric pop-up keyboard.

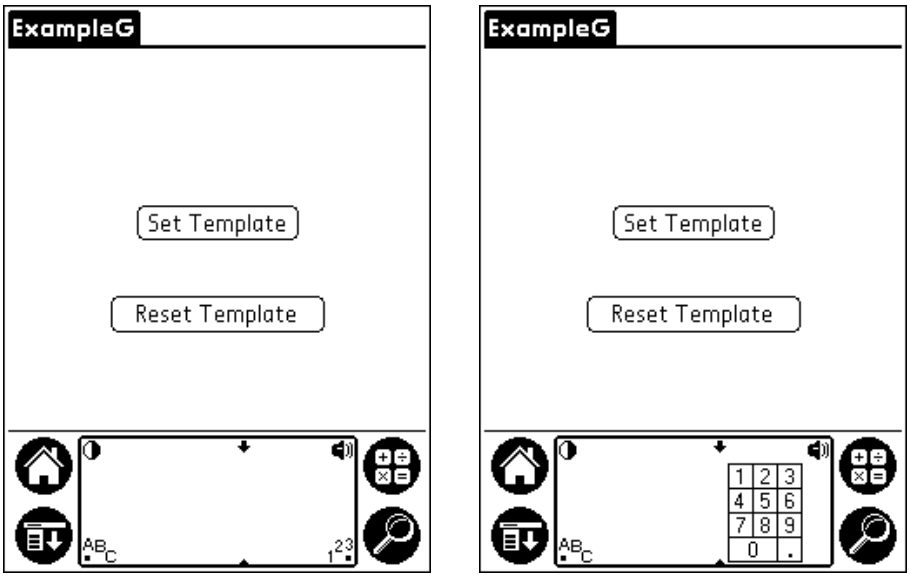

Figure 11-3. Example G

Figure 11-4 shows the look of the Silkscreen after Example G has exited. The Silkscreen change remains even after the Address Book is launched. The change will remain until the unit is reset, when the Silkscreen is restored to its default configuration and removes the changes made by Example G. Example G could be modified to reinstall the new template when the unit is reset.

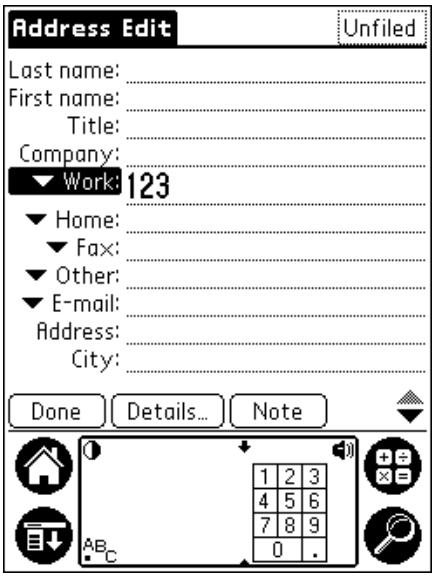

Figure 11-4. Permanent Silkscreen Change

Registering for Silkscreen events

If an application doesn't include a form event-handler, code will need to be added to handle silkscreen launch codes. To receive this launch code, the application must first register for it.

```
UInt16 cardNo;
LocalID dbID;

SysCurAppDatabase(&cardNo, &dbID);
SysNotifyRegister(cardNo, dbID, trgNotifySilkEvent, NULL, 0,
NULL);
```

On exit, the application must unregister.

```
UInt16 cardNo;
LocalID dbID;

SysCurAppDatabase(&cardNo, &dbID);
SysNotifyUnregister(cardNo, dbID, trgNotifySilkEvent, 0);
```

Inside of the `PilotMain()` function, the application must include code to handle these special launch codes.

```
UInt32 PilotMain (UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{
   SysNotifyParamType   *notify_param;

   switch(cmd)
   {
       case sysAppLaunchCmdNotify :
           notify_param = (SysNotifyParamType *)cmdPBP;
           switch (notify_param->notifyType)
           {
              case trgNotifySilkEvent :
                  // insert code to handle display extent changes
                  break;

           }
           break;

   // standard PilotMain() code follows …
```

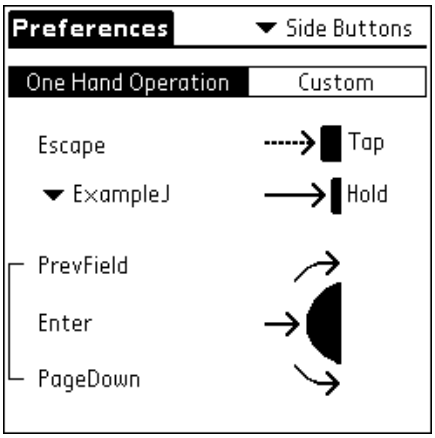

Figure 11-5. Side Buttons Preferences Panel

With the Side Buttons Preferences Panel, the user can select different key mappings for the Jog Wheel and Auxiliary Button. For example, the Jog Wheel up and down rotations can be remapped from PrevField and NextField to PageUp and PageDown.

- Warning: It is not recommended that applications override the user preferences.

The user can also select which application is launched when the Auxiliary button is pressed and held. By default, this is set to Voice Pad. However, pressing and holding the Jog Wheel will always bring up the Contrast dialog. This cannot be remapped.

Voice Pad (as well as Example J) needs to be able to determine if it was launched via the Auxiliary button, or from the Application Launcher. If it was launched from the Auxiliary button, it will automatically begin recording. In order to determine if the Auxiliary button is being held, the following code snippet needs to be run at application start up.

```
if (_TRGKeyFeaturePresent(&keys) && (keys & keyBitAux))
{
    if ((KeyCurrentState() & keyBitAux) != 0))
        launchedFromButton = true;
    else
        launchedFromButton = false;
}
```

If an application needs to override the user selections, the following virtual key values will need to be handled in the event-handling loop.

Action	*Virtual Key*
Jog Wheel rotated up	vchrJogUp
Jog Wheel rotated down	vchrJogDown
Jog Wheel pressed	vchrJogPush
Auxiliary button pressed	chrAux

Example K – Alarms

Example K provides examples of ...

- *vgaGetScreenState API call*
- *vgaSetScreenState API calls*
- *Alarms*

Perform the following steps:

1. Copy the files from the `\Source\ExampleK` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the **ExampleK** Project.
3. Install `ExampleK.prc` on a HandEra 330 or POSE.
4. Set up a non-QVGA app to run in "Centered" mode in the Application Preferences Panel.
5. Launch `ExampleK.prc`. Tap on the "Set Alarm" button.
6. Launch the non-QVGA app.

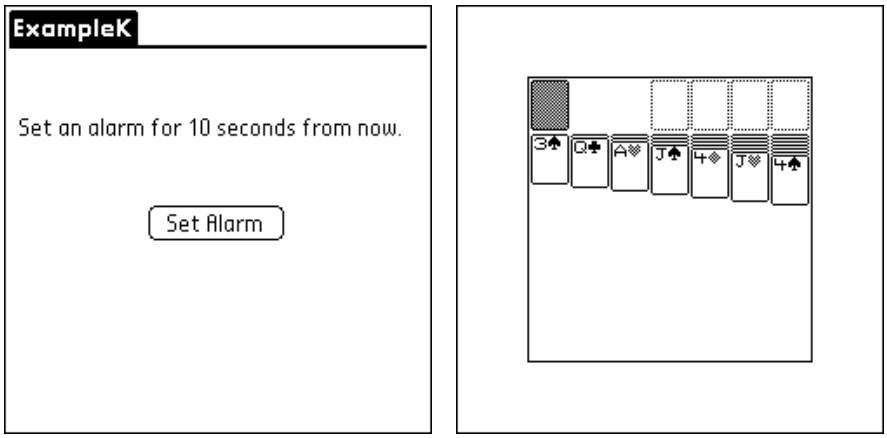

Figure 11-6. ExampleK

7. After 10 seconds, the Example K dialog will pop up over the non-QVGA app. (See Figure 11-7).

8. Tap the OK button. The non-QVGA app should now appear.

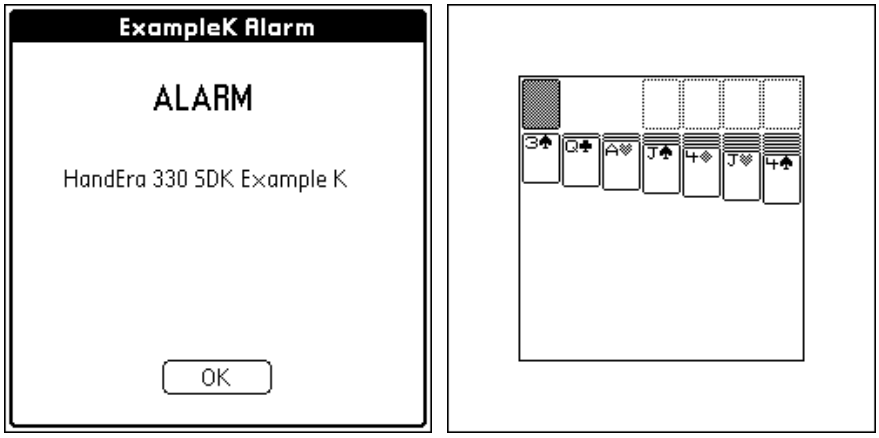

Figure 11-7. ExampleK Alarm.

To successfully pop up a dialog over an application, the alarm application must include the API calls `VgaSaveScreenState()` and `VgaRestoreScreenState()`. These calls are used to save and restore the non-QVGA app. For more information on these API calls, refer to the *HandEra 330 Programmer's Reference*.

Example L – Icon Scaling

Example L provides examples of ...

- *Scaling low-res icons*

Perform the following steps:

1. Copy the files from the `\Source\ExampleL` directory of the SDK to a directory on a local drive.
2. Launch the **CodeWarrior IDE** and open the MemoPad Project.
3. Install and launch `ExampleL.prc` on POSE or a HandEra 330.

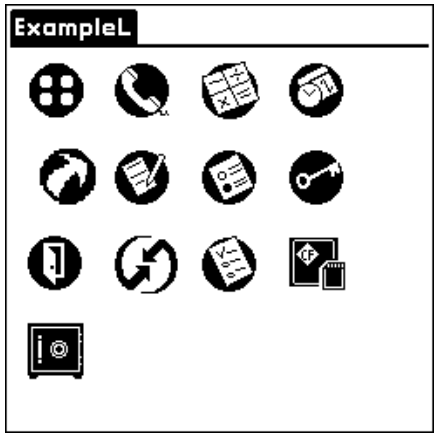

Figure 11-8. ExampleL

Example L is intended for developers interested in creating application launchers. This example displays an array of icons from the currently installed apps. If a high-resolution, 32x32 icon is present (an icon with an ID of 2000), then it is displayed. Otherwise the 22x22 application icon is scaled by 1.5.

Example X – .R File

Example X provides examples of ...

- *High-resolution icons*
- *High-resolution bitmaps*
- *High-resolution forms*
- *Additional fonts*
- *Screen rotation*

Example X walks you through the steps in creating and using .r text files with CodeWarrior. Constructor, the GUI resource editor, currently does not support forms larger than 160x160. However, to overcome this problem, convert the binary resource file to a .r text file. CodeWarrior supports the use of both .rsrc binary files and .r text files in the same project. Therefore, only the forms larger than 160x160 need to be converted to the .r text file format.

The following steps were taken to convert Example A to Example X. These steps describe how to extract the Main form resource from the binary resource file and change it into a .r text resource file. A few code changes to Starter.c are also necessary.

Perform the following:

1. Copy the Example X subdirectory from the SDK.
2. Open the **ExampleX** project file.
3. Open the Example X's **Settings** dialog.
4. Click on **File Mapping** in the Target Settings Panel list. Refer to Figure 11-9.

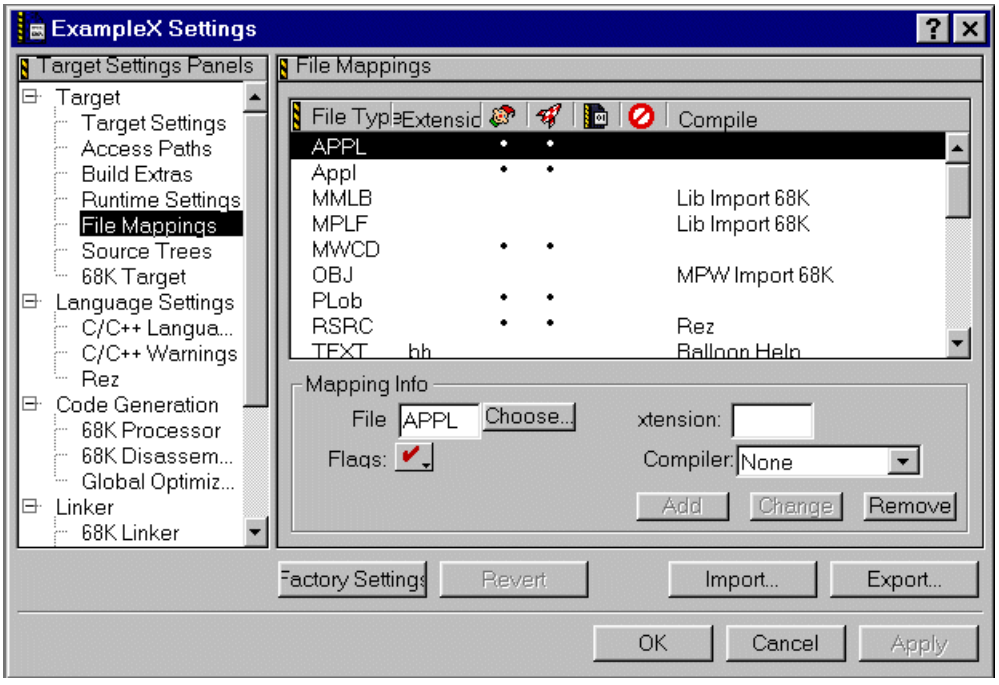

Figure 11-9. File Mapping

5. Click **RSRC** under the File Type column. Refer to Figure 11-10.

6. Change the Compiler to **Rez**.

7. Click **Change** to ensure the changes take effect.

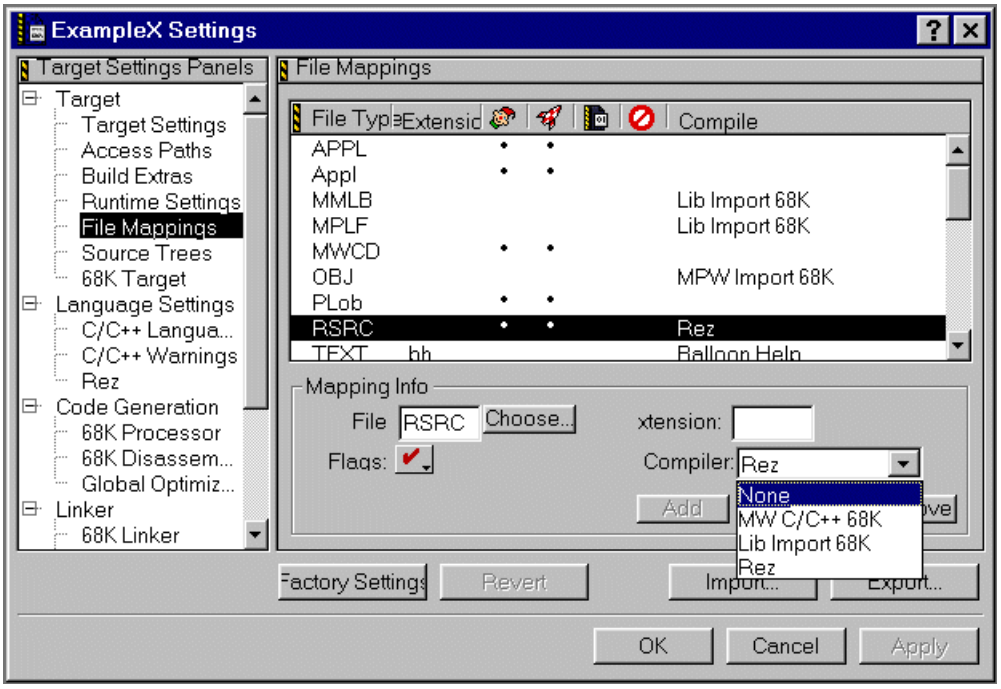

Figure 11-10. RSRC File

8. Click on **.rsrc** under the Extension column.

10. Change the Compiler to Rez.

11. Click Rez in the Target Settings Panel list.

12. Set the Prefix File to **UIResdefs.r**.

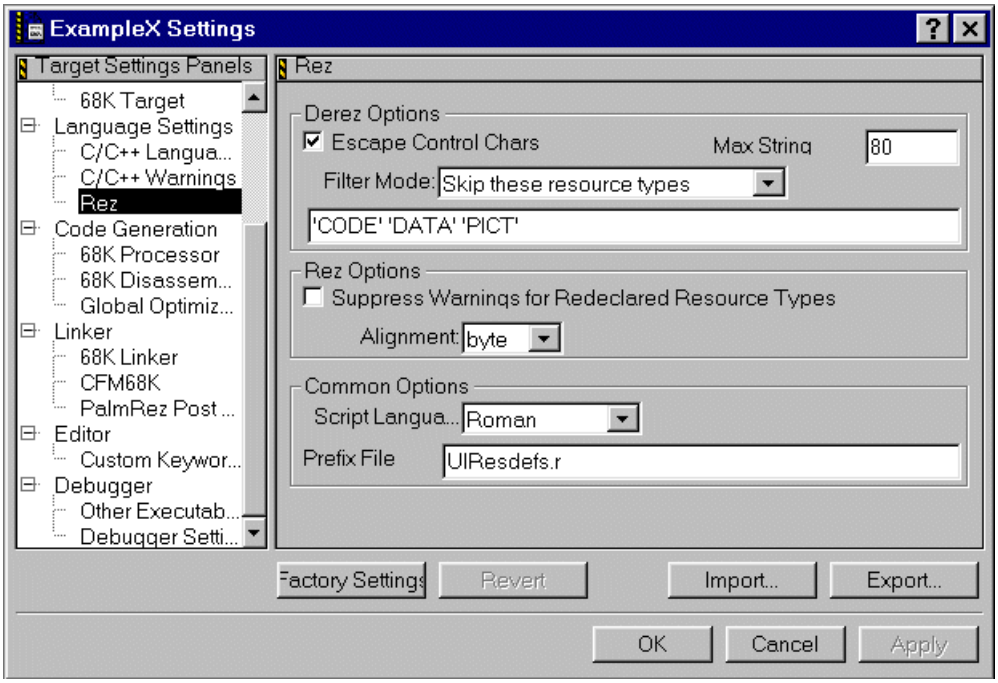

Figure 11-11. Rez

13. **Save** and **Close** the settings dialog.

14. Right click **Starter.rsrc** and **disassemble** the file. This produces a .r text file of the resource. Only the Main forms structure and title bar structure need saved.

15. **Save** the file as **Starter.r** in the rsc subdirectory.

16. Add the file **Starter.r** to the Example X project. Refer to Figure 11-12.

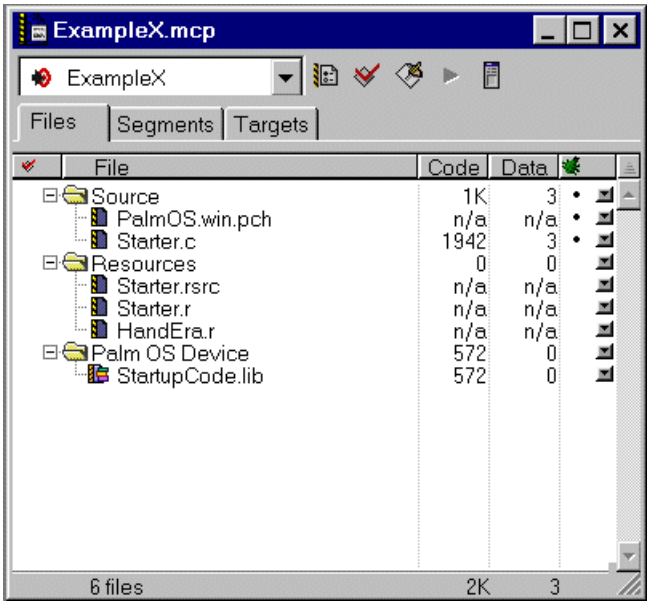

Figure 11-12. Starter.r

Definitions for the structures found in the .r text resource files can be found in the file UIResDefs.r in the PalmOS SDK header files.

At this point, Example A and Example X basically produce the same output.

1. Convert Main form to a **240x240** form.

2. Edit the file **`Starter.r`** and change the width and height fields in the form from **160x160** to **240x240**.

3. Edit the file **`Starter.c`** to add calls to setup the screen mode.

Index

M

N

O

P

R

S

T

U

V

W