AlphaSmart, Inc.

Dana Companion Guide

Version 1.4
Date: 7/1/2002

CONTACT INFORMATION

AlphaSmart, Inc.

973 University Ave.
Los Gatos, CA 95032
U.S.A.
408-355-1000
www.alphasmart.com

Palm Computing World Wide Web www.palm.com

Metrowerks World Wide Web www.metrowerks.com

Table of Contents

About This Document

The *Dana Companion Guide is* part of the Dana Software Development Kit (SDK). This document is intended as a guide for developers interested in creating applications for the Dana.

For the most recent version of the SDK, check at www.alphasmart.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 www.palmos.com. Relevant documents include:

- *Palm OS Reference*
- *Palm OS Companion*
- *Constructor for Palm OS*

This document assumes the reader is familiar with basic Palm OS concepts detailed in the *Palm OS 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 Companion*, are italicized.

Parts of the Dana feature set are based on technology licensed from HandEra, Inc. The APIs in the Dana SDK are similar to those in the HandEra SDK, but some modifications have been made in order to support the Dana hardware platform - in particular Dana's 160 x 560 pixel screen. Developers who have already modified their applications for the HandEra platform should find modifying their application for the Dana platform even simpler, since they are already familiar with many of the APIs and supporting Dana's larger screen requires fewer modifications

Chapter 1: Application Development

Tools

The recommended software development system for Dana is CodeWarrior for Palm OS from Metrowerks (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 1.6.2 or newer.

The header files provided by AlphaSmart 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** – Dana specific header files.
- **\examples** – CodeWarrior example projects.
- **\docs** – *Dana Companion Guide, Dana API Reference, PrintBoy Technical Note.*
- **\poser-** Dana specific poser for both the Mac and PC

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

- CodeWarrior 7
- Constructor v1.6
- Palm OS SDK 4.0

Palm OS Emulator (POSE)

The Palm OS® Emulator (POSE) is software that runs on PCs/Macs 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

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

Dana

Physical Characteristics

As shown in the picture above, the mechanical design of the Dana is radically different than any other Palm Powered device. Since Dana is a not strictly a handheld device it does not require the use of a cradle like other Palm Powered devices.

Power

The Dana supports both standard AA batteries and rechargeable Nickel-Metal Hydride batteries. External power can be supplied to the unit directly via the power jack on the back of the unit. The rechargeable batteries will be charged when connected to either to USB or an AlphaHub.

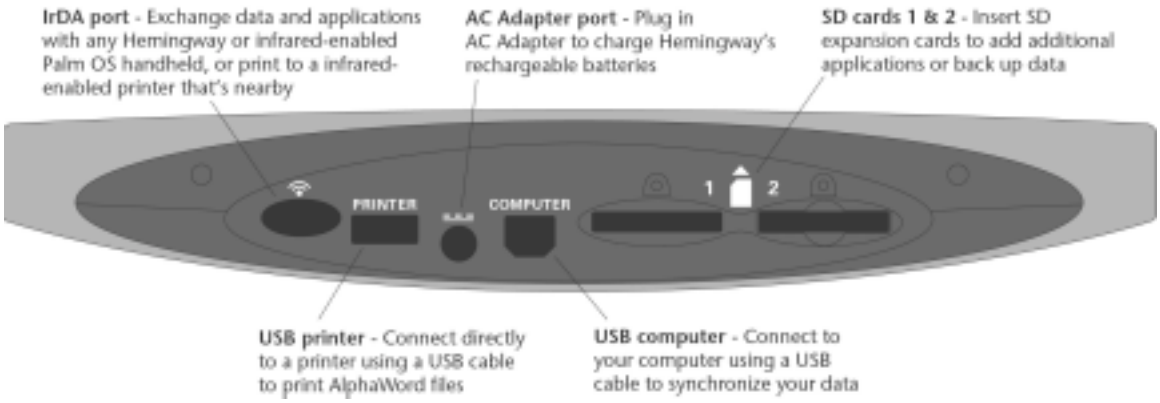
Memory and CPU

The Dana comes standard with 8 MB of DRAM and 4 MB of flash. It includes the Motorola Dragonball VZ processor running at 33 MHz.

Keyboard

The Dana has a full sized keyboard instead of just 4 standard Palm buttons found on most Palm Powered devices. The standard Palm buttons can be found on the top row of the keyboard along

with a variety of other task specific keys. A number of the keys have a secondary function associated with them. Those keys have the primary function printed in white with the secondary function printed in orange. Holding down the function/alt/option key and then pressing the key with the secondary function activates the secondary function of a key.



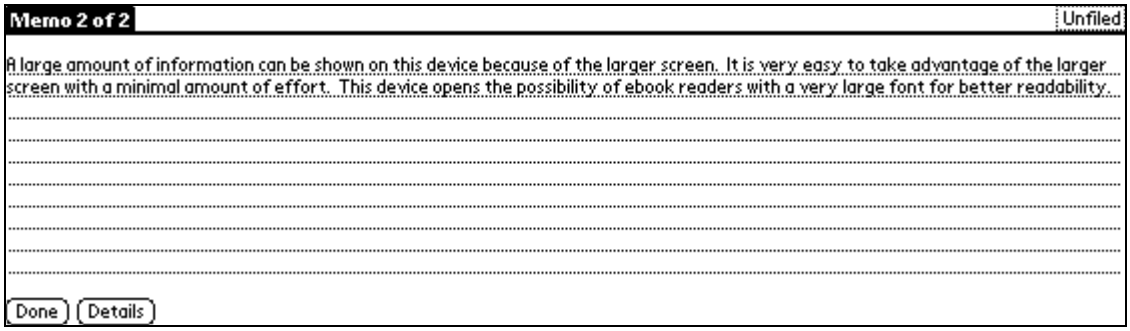
Expansion Slots

Expansion

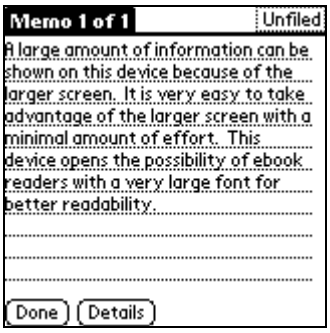
The Dana comes with two industry standard Secure Digital (SD) expansion slots. SD support is currently limited to storage devices at this time. In the future it will support SDIO cards such as Bluetooth and possibly a modem. The Dana's SD support is compatible with the Expansion and File Manager found in the Palm 4.0 SDK. Applications that take advantage of the expansion media on the Sony Clie and Palm m505 should also work on the Dana.

Screen

To date, most Palm Powered devices have featured a screen resolution of 160x160 pixels. The Dana features an LCD with a size of 560 pixels wide by 160 pixels tall. The screen has a bit depth of 2 bits NOT 4 bits like other Palm Powered devices! The default bit depth will be set to 1 and then applications can change it to 2 bits. Applications that assume a minimum bit depth will probably crash on the Dana. The pixel density on Dana is the same found on 160 x 160 Palm Powered devices. Since the Dana has the larger screen, it can display a large amount of information. The screen shots below show the difference in the amount of information that a user can see on a Dana.



Dana Memo Pad



Standard Palm Powered Device Memo Pad

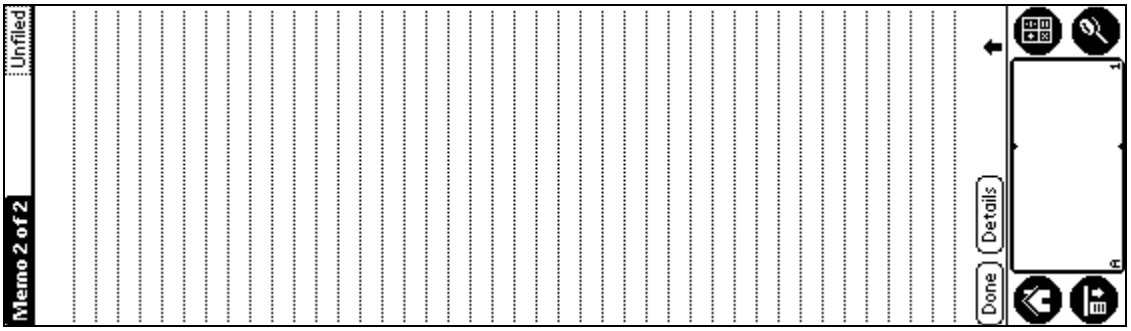
USB Slave
The Dana only supports USB as the means for performing a HotSync. A serial connection is not available on the device. The user attaches a standard USB between the PC/Mac and the Dana. In some special applications the Dana can be used as an extra keyboard when connected to a PC/Mac.

USB Master
Dana is the first Palm Powered device that can print directly to USB printers. The user can use a standard USB cable to connect a USB printer to the Dana. The same printing interface is used for both IrDA and USB. In fact, the application writes to one interface that handles both the IrDA and USB hardware.

Software

Software has been added to the standard Palm Powered ROM to take advantage of the new hardware functionality on the Dana.

Screen Extension
The Screen Extension enables applications that are Dana aware to take advantage of the larger screen. When the screen is not rotated (0 degree rotation mode), it is considered a "wide" screen. If the screen is rotated either at 90 degrees or 270 degrees the screen is considered "tall".



Rotated (tall) screen

A user might want to rotate the screen if they want to import or view information while holding the device in one arm like a notepad. By allowing the screen to rotate at 90 and 270 degrees it enables both right hand and left hand users to use the “tall” functionality.

The user through the use Screen application can rotate the screen. An application can change the screen rotation by using the Screen Extension API. This is normally not recommend, but this could be used if for some reason an application does not want to run in one of the screen rotations. When an application starts up it can check the rotation and then set it to something that it can deal with.

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

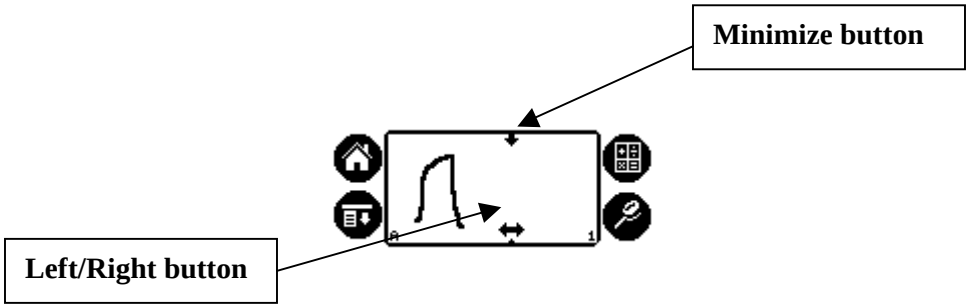
WritePad Extension
Since the introduction of the Pilot 1000, all Palm Powered devices have featured a silkscreened area immediately beneath the LCD. Traditionally, this area has included:

- 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.

This area is referred to as the **WritePad** for the purposes of this document. The WritePad for the Dana 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 at various locations on the screen depending on the screen rotation. When the device is running an application that uses the wide screen, Graffiti input is not available.

Application Launcher plus WritePad

The WritePad for the Dana also contains two extra buttons: minimize and left/right.



WritePad with Stylus Input

Tapping on the minimize button causes the WritePad to minimize into either the right or left bottom corner of the screen depending on which side of the screen it is currently on.

WritePad Minimized

Tapping on the left/right arrow button causes the WritePad to switch to the other side of the screen.

WritePad on the Left

It is important to note that the functionality of the virtualized WritePad 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 WritePad has the following advantages:

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

Keyboard Extension

The Dana keyboard is used mainly for text input, but keys have been setup so that high frequency stylus taps can be accomplished by one or more keystrokes. A menu key is on the keyboard so that a user can drop the menu down. Once the menu is down the user can "walk" through the menu using the up/down arrow keys. Pressing the left and right arrow keys will drop down the previous or next menu.

On screen buttons can be "tapped" by holding down the function key and pressing the tab key. Once an onscreen button is highlighted, it can be "tapped" by continuing to hold down the function key and pressing the enter key.

If an application form has the Graffiti shift indicator on it, then the Keyboard Extension will change its status based on the status of the shift and caps lock keyboard keys.

Other keyboard functionality is described in the *Dana User's Guide*. As an application developer all of this keyboard functionality should work seamlessly with no effort on your part.

Most of the keystrokes translate into regular Palm key events with the exception of the F1 through F8 keys and the Send key. The exact format of those key events will be discussed later in the document.

Printing

PrintBoy Documents from Bachmann Software is included in the Dana ROM. PrintBoy Documents aware applications should work without modifications on the Dana.

To add PrintBoy Document support to an application please read the *Bachmann Software PrintBoy Technical Note.*

Fonts

FontBucket from Hands High Software is included in the Dana ROM. A variety of FontBucket fonts will come installed in the ROM for use by all FontBucket aware applications.

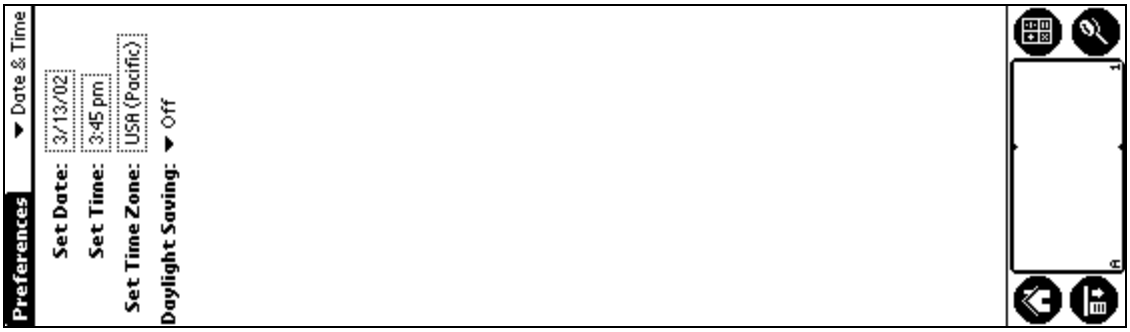
To add FontBucket support to an application please read the *FontBucket Developers.pdf.*

Chapter 3: Wide/Tall Screen Programming

Almost all standard Palm applications (legacy applications) will run on the Dana, unless they write directly to screen memory. Legacy applications are run in the middle 160 x 160 of the screen in the 0 degree rotation (wide mode). In the 90 or 270 degree rotation (tall mode) legacy applications run in the top 160 x 160 portion of the screen.

Legacy Mode 0 Degree Rotation

Here is an example of the Date & Time Preference Panel running in the 270 degree rotation mode. The 90 degree rotation mode is just the opposite of this. In 90 or 270 degree mode the WritePad cannot be minimized.



Legacy Mode 270 Degree Rotation

When a legacy application is running, the Palm OS routines related to screen size return the following information:

```
UInt32 width,
UInt32 height,
UInt32 depth,
Boolean colorUsable;

WinScreenMode(winScreenModeGet,&width, &height, &depth,
&colorUsable);

// returns hardware screen size
// width = 560
// height = 160
```

```
Coord extentX;
Coord extentY;
WinGetDisplayExtent (&extentX, &extentY);

// returns window size
// extentX = 160
// extentY = 160
```

Steps for Building Wide/Tall Apps

To modify an existing application for the wide/tall screen and maintain compatibility with existing Palm Powered devices, perform the following steps:

1. **Add a 'wTap' identifier resource**

   All wide/tall aware apps need to include a special 'wTap' (**w**ide **T**all **ap**p) resource. This resource is used to differentiate wide/tall apps from legacy apps.

2. **Create Wide/Tall forms**

   Support for wide/tall 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 Dana. Resizing a form at run-time is preferable since it eliminates the need for duplicate resources and code.

Adding 'wTap' Resource

If applications want to take advantage of the wide/tall screen they will need to add a resource of type 'wTap' and ID of 1000. Bit 0 in the resource should be set to 1. Here is the contents of the WideTallApp.r file that can be included in the application project:

```
type 'wTap'
{
    unsigned longint;
};

resource 'wTap' (1000, "Wide/Tall Aware")
{
    0x00000001; //Set bit 0 for wide tall aware.
};
```

Unfortunately, Constructor cannot create 'wTap' resources. However, this resource can be created in a .R file. Simply add WideTallApp.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 "wTap" ID 1000 0x00 0x00 0x00 0x01
```

When an application adds this resource, the Palm OS routines related to screen size return the following information:

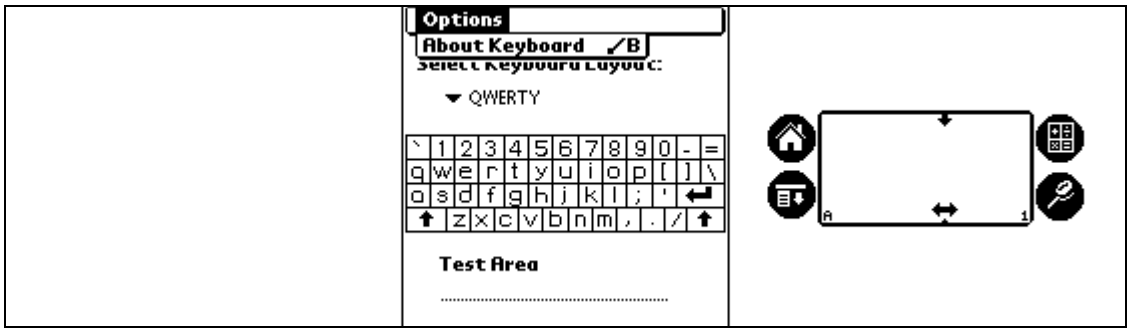
```
UInt32 width,
UInt32 height,
UInt32 depth,
Boolean colorUsable;

WinScreenMode(winScreenModeGet,&width, &height, &depth,
&colorUsable);

// returns hardware screen size
// returns in Wide Mode
// width = 560
// height = 160

// returns hardware screen size
// returns in Tall Mode
// width = 160
// height = 560


Coord extentX;
Coord extentY;
WinGetDisplayExtent (&extentX, &extentY);

// returns in Wide Mode
// extentX = 560
// extentY = 160

// returns in Tall Mode
// extentX = 160
// extentY = 499
```

Create Wide/Tall Forms

Wide/Tall forms refer to forms larger than 160 x 160 pixels. If application developers want to take advantage of the wide/tall screen on the Dana they do not have to modify all of their application forms. Most applications have maybe 1 or 2 forms that are good candidates for growing to fit the wide/tall screen. Applications such as word processors, for writing, and ebook applications for reading are two categories of applications that probably have 1 form that needs to be resized. The nonmodal forms that do not take advantage of the wide/tall screen, are called legacy forms.

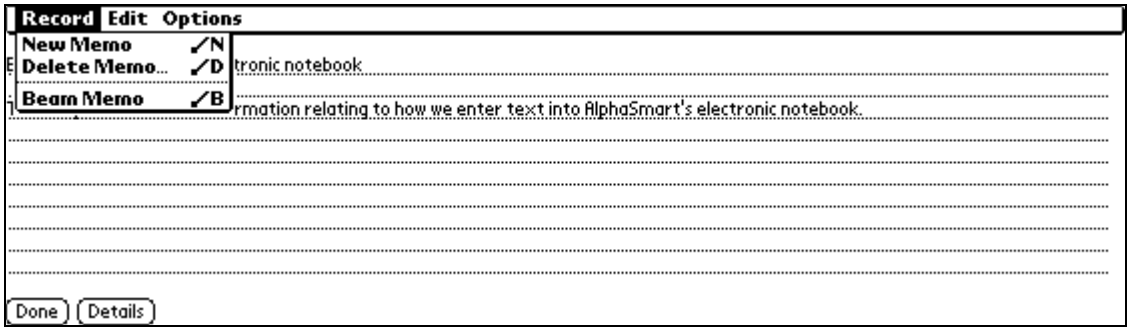
Legacy forms that are 160 x 160 or less in an application that has a 'wTap' resource, will be displayed just the same as a legacy application form. The only difference is that the menu will be on the left hand side of the screen instead of in the middle as in the case of legacy applications. So a majority of forms do not have to be modified when an application wants to take advantage of the wide/tall screen.

Shown below is a legacy application that does not have the 'wTap' resource. Notice that the menu bar is in the middle of the screen.



Legacy Application

Shown below is the same application, but with the 'wTap' resource included. Notice that the menu bar starts at the left side of the screen and that the WritePad is not available.



Legacy Form in Wide/Tall Application

If a wide/tall application draws directly to the screen using a legacy form, then the application will have to factor in the offset to the center of the screen. Here is a code snippet that is only needed if the application draws directly to the screen thus bypassing the Palm drawing routines.

```
UInt32          version;

if(_ScreenFeaturePresent(&vgaVersion))
{
    ScrnGetSystemState(ScrnSystemStateType *pState)

    if(pState->screenMode == screenModeOffset)
    {
        x += pState->offset_X;
        y += pState->offset_y;
    }
}
```

By default, the wide/tall forms used with the Dana should be 560 x 160 pixels in the wide mode and 160 x 499 pixels in the tall mode. The Tall mode forms are shorter because the WritePad is located below the form in 90 and 270 degree rotation mode. There are two methods to create wide/tall forms on the Dana:

- 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 wide/tall 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. 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 used 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);
}
```

When a `frmOpenEvent` event is received, the application needs to call the `SizeForm()` function or some similar function.

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

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

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

Obviously, this increases the workload for a developer. It requires more code to reposition each resource on the form. 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 Dana or any other Palm Powered device.

Program Method Example A

Example A in the Dana SDK provides an example of how developers can use the Program Method to modify their existing forms to be wide/tall forms when running on the Dana. The Example A project is a starter application generated from CodeWarrior 7. A form was built using Constructor, which contained the following objects:

- Text field – filling the width of the screen
- Button – placed on the left side of the screen
- Checkbox – placed on the right side of the screen
- Bitmap – placed in the middle of the screen

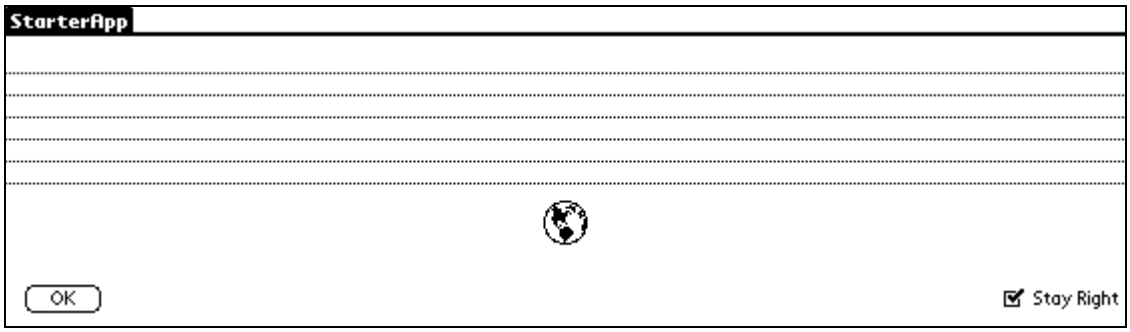
The code in `MainFormInit()` handles adjusting the objects on the form so that they still in the same relative position even if the form size changes. This function will rearrange the items correctly whether it runs on the Dana or other Palm Powered devices.

- **Note:** Remember to include the WideTallApp.r in your project otherwise the screen size will be reported as 160 x 160.

Without the 'wTap' resource, the application will be run in legacy mode. As shown below:

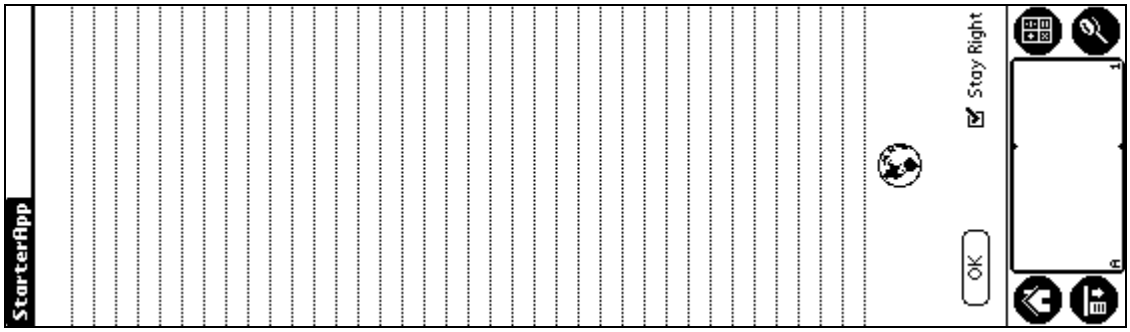
Example A without 'wTap'

With the 'wTap' resource, the application will grow the form to the size of the screen and thus hide the WritePad.



Example A with 'wTap' (Wide mode)

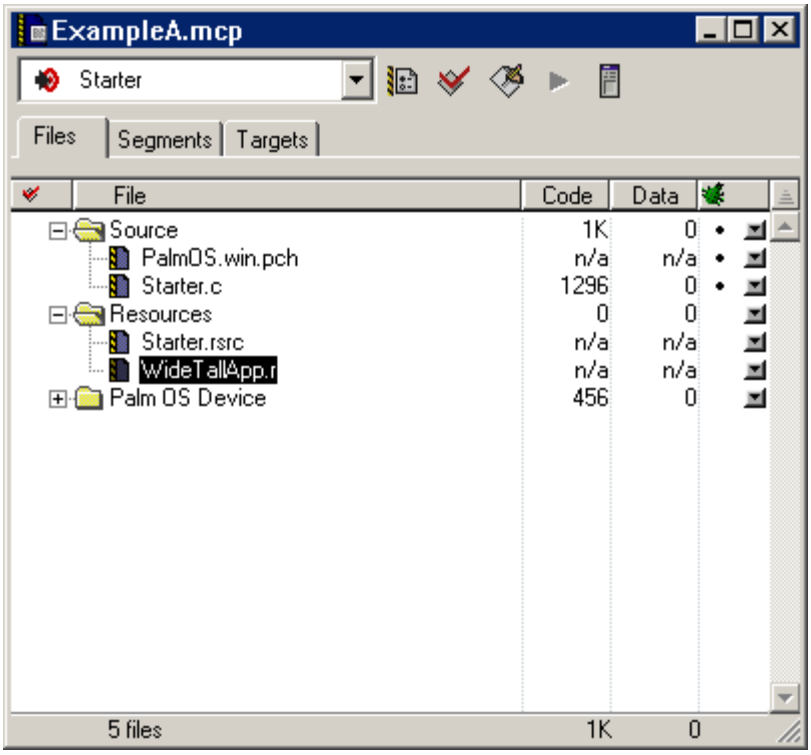
If the screen is rotated, then the code resizes the form to take advantage of the new screen orientation.



Example A with 'wTap' (Tall mode)

Perform the following steps:

1. Add the WideTallApp.r file to the project.



Example A with WideTallApp.r

2. Add code to the `MainFormInit()`routine to move the form objects to their proper location depending on screen rotation and size.

Resource Method

Applications using this method contain wide/tall 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 Dana and the current rotation of the screen.

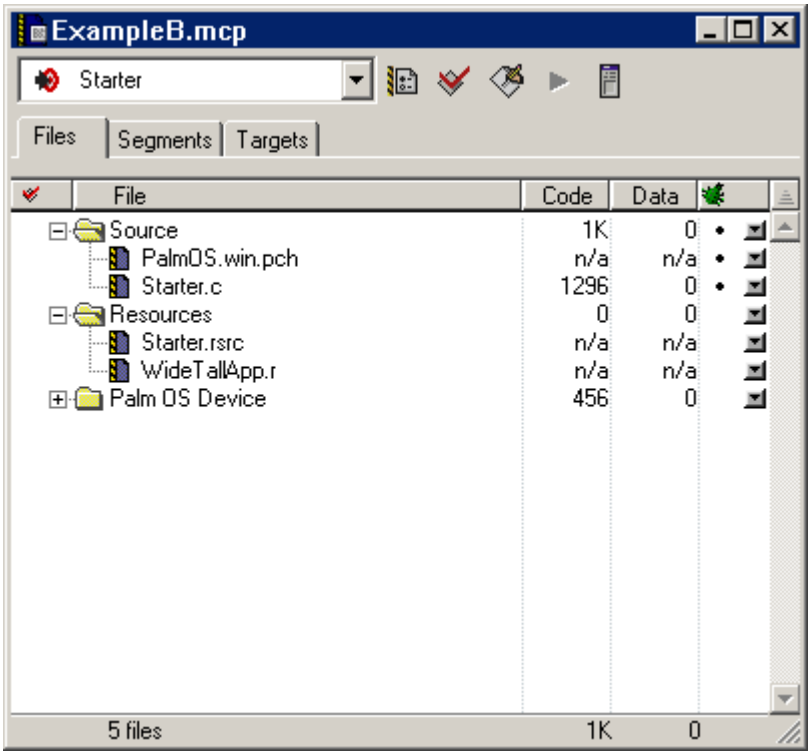
Each developer needs to decide which wide/tall method works for them. If they want to minimize the code changes then go with the Resource method. If they want to minimize the resource changes then they will go with the Programming method.

Resource Method Example B

Example B in the Dana SDK provides an example of how developers can modify the application's resource files to create the higher-resolution forms. The concept is to derez a current resource based form into its text based equivalent. CodeWarrior can be used to derez the current form and then also rez the new version back into the application. The new form and all of its objects must have unique numbers.

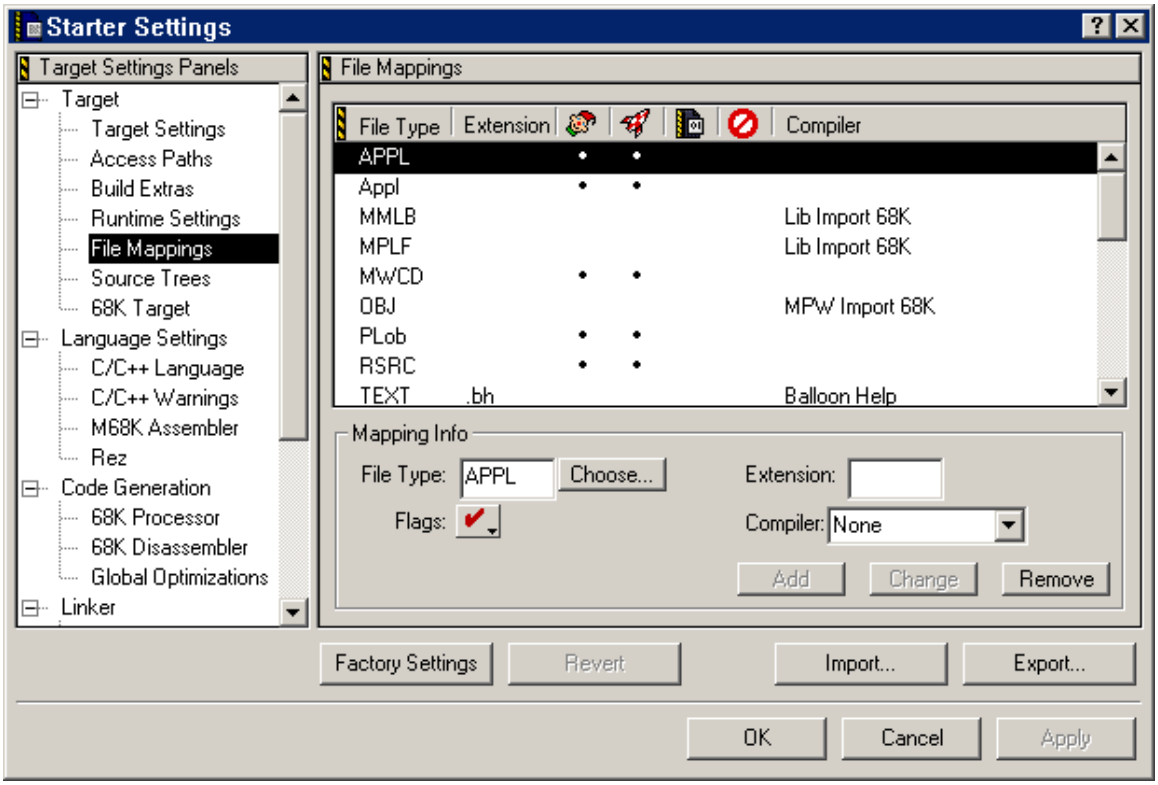
Perform the following steps:

1. Add the WideTallApp.r file to the project.



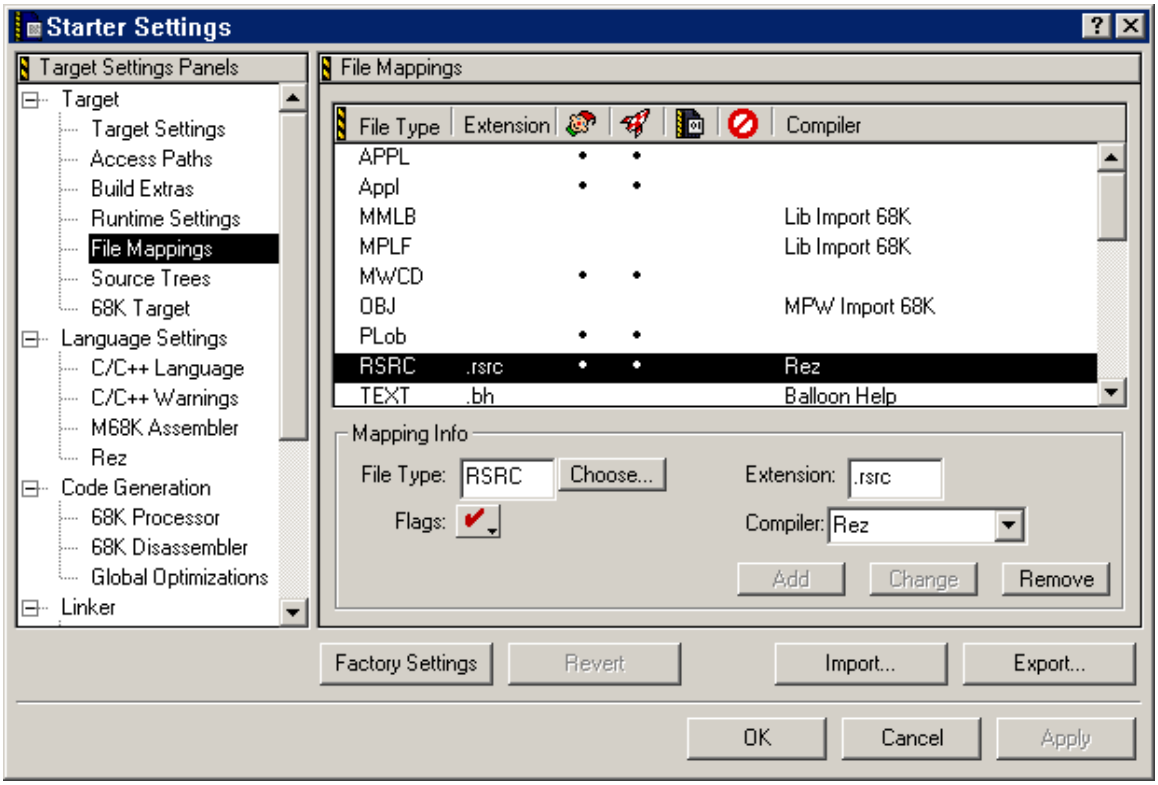
Example B with WideTallApp.r

2. Open Example B's Setting dialog and click on File Mapping the Target Settings Panel list.



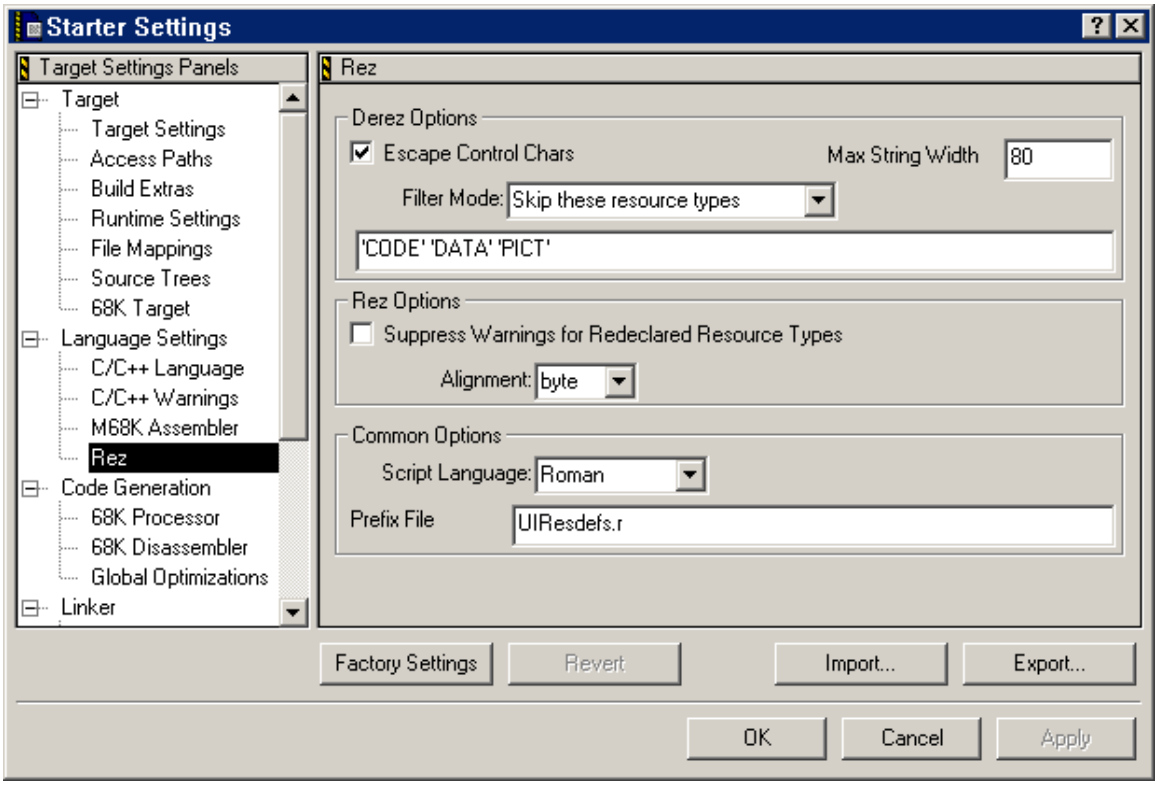
File Mapping

3. Click RSRC under the File Type column.
4. Change the Compiler to Rez.
5. Type in .rsrc in the Extension area.
6. Click Change to ensure the changes take effect.



RSRC File

7. Click Rez in the Target Settings Panel list
8. Set the Prefix File to UIResdefs.r



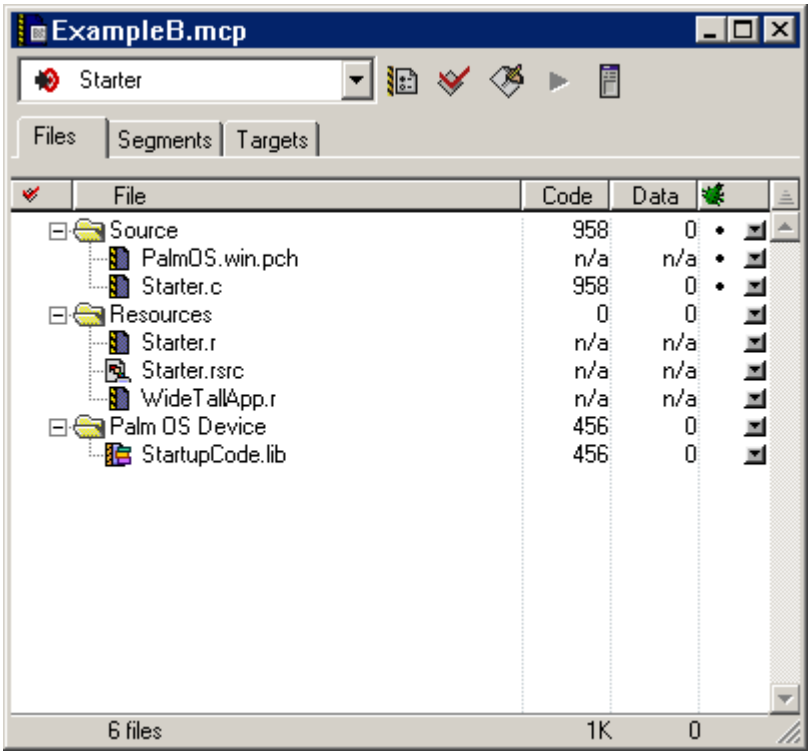
Rez prefix file

9. Save and Close the Settings dialog.

10. Right click Starter.rsrc and disassemble the file. This produces a .r text file of the resource. Only the Main form structure and title bar need to be saved, everything else can be deleted.

11. Save the file as Starter.r in the Rsc subdirectory.

12. Add the file Starter.r to the Example B project.



Starter.r added

13. Edit Starter.r to change the Main form numbers in Starter.r so that the forms do not collide when the project is built. This is the wide form.

14. Edit Starter.r to copy and duplicate the wide form to make the tall form. Change the form numbers of the tall form.

15. Edit Starter.c so that the correct form gets loaded depending on the screen orientation.

```
UInt32 version;
ScrnRotateModeType rotation;

if (_ScreenFeaturePresent(&version) == true)
{
    ScrnGetRotateMode(&rotation);
    if (rotation == rotateScrnMode0)
    {
        FrmGotoForm(wideMainForm);
    }
    else    // either 90 or 270 degree rotation
    {
        FrmGotoForm(tallMainForm);
    }
}
else    // non AlphaSmart device
{       // just use the 160 x 160 form
    FrmGotoForm(MainForm);
}
```

Chapter 4: Keyboard Programming

The biggest difference between the Dana and other Palm Powered devices, other than the screen size, is the full sized keyboard built into the device. Besides just adding the keyboard hardware, AlphaSmart has added keyboard functional to the Palm OS. All of this functionality will be available to all Palm applications, not just those modified to run on Dana. To get a comprehensive list and description of that keyboard functionality please read the Dana User's Guide.

Input from the keyboard is available to applications using a normal Palm event loop. Most developers do not need to worry about handling the keyboard, any differently than they handle Graffiti input.

```
static void AppEventLoop(void)
{
    UInt16 error;
    EventType event;

    do {
        EvtGetEvent(&event, evtWaitForever);

        if (! SysHandleEvent(&event))
            if (! MenuHandleEvent(0, &event, &error))
                if (! AppHandleEvent(&event))
                    FrmDispatchEvent(&event);

    } while (event.eType != appStopEvent);
}
```

Keyboard Input Format #1

For those key sequences that map to a Palm char the format of the keyDownEvent will be the same as any other keyDownEvent that was generated using the WritePad. The only exception is when the function key modifier is pressed then Format #2 will be used.

```
// Palm defined structure
struct _KeyDownEventType {
   WChar          chr;
   UInt16         keyCode;
   UInt16         modifiers;
   };

chr = Palm character found in Chars.h;
keycode = 0
modifiers = appropriate modifiers
```

Keyboard Input Format #2

For those key sequences that don't map directly to a corresponding Palm character (F1 – F8 keys), applications will see a keyDownEvent event type with the following information:

```
// Palm defined structure
struct _KeyDownEventType {
   WChar          chr;
```

```
   UInt16         keyCode;
   UInt16         modifiers;
   };

chr = AS_VIRTUAL_KEY found in WideTallAppChars.h
keycode = AlphaSmart keyboard event
modifiers = commandKeyMask
```

An AlphaSmart keyboard event is defined as: 8 bits of modifier key bit information + 8 bit icode = 16 bits. Icode and modifier key bit definitions `(KEYBOARD_MODIFIER_COMMAND, KEYBOARD_MODIFIER_FUNCTION, …)` can be found in WideTallAppChars.h include file.

Keyboard Input Format #3

The modifier keys (Caps Lock, Left/Right Shift, Ctrl, Alt/Option/Function, and Cmd keys) use the MODIFIER_KEY_EVENT event type found in WideTallAppChars.h include file. The data is encoded the same at Format #2.

```
// Palm defined structure
struct _KeyDownEventType {
   WChar          chr;
   UInt16         keyCode;
   UInt16         modifiers;
   };

chr = AS_VIRTUAL_KEY found in WideTallAppChars.h
keycode = AlphaSmart keyboard event
modifiers = commandKeyMask
```

There is also a set of keys that normally translate into a Palm char, but because of the current state of the modifiers, they no longer translate. These keys are to the right of the F8 key (date, address, …) on the top row. These keys have two functions: one that can be invoked by just pressing it, and the other function that is invoked by pressing the function key and then pressing the original key. If a modifier other than the function key is pressed then the format will follow the Format #2 mentioned earlier. When a key event like this occurs, the icode of the key follows the F8 key. An example is in order…if the ctrl-address key sequence is pressed then the icode for the event would be `ICODE_F10_KEY.`

This table details the translation that occurs between special keys and Palm chars.

Key Name	Palm Char	Comment
send	vchrSendData	commandKeyMask set
esc	chrEscape	Application only receives this is a Cancel button was not found on the form. Esc key will try to press form Cancel button.
enter	chrLineFeed	
del	chrDelete	

tab	tabChr	Functionality changed in
	vchrPrevField	Keyboard application
	vchrNextField	
backspace	chrBackspace	

In the United States version of the Dana, the user can change the layout of the keyboard using the Keyboard application. The `KybdGetLayout()` routine can be used to find out the current layout.

The current state of the modifier keys can be discovered by calling `KybdGetModifiers()`, which returns the logical state of the modifier keys. The logical and physical state of the keys could differ if the user has turned on Sticky Keys.

To enhance the user experience with the keyboard, an application could be modified so that the OK button or default button can be activated by press the enter key. The arrow keys might be used to navigate around the form if it makes sense in the form.

One item that does need to be mentioned is the Graffiti shift indicator. If a form has the Graffiti shift indictor it will be driven by the state of the shift key and caps lock key in some instances and by the shift capability of the WritePad in other instances. To create a shifted character the user must press the shift key or use the caps lock key. Shifted keyboard input cannot be entered by using a combination of the WritePad and the keyboard. The user cannot pen a stroke in the WritePad that sets the Graffiti shift indicator and then type a ‘c’ and expect to get a ‘C’.

AlphaSmart Key Example C

Example C in the Dana SDK provides an example of how developers can parse the F1 through F8 keys. Code has also been added to detect when the esc key is pressed.

```
static Boolean MainFormHandleEvent(EventPtr eventP)
{
    …

    switch (eventP->eType)
    {

        …

        case keyDownEvent:
            if (eventP->data.keyDown.chr == AS_VIRTUAL_KEY)
            {
                if (eventP->data.keyDown.keyCode == ICODE_F1_KEY)
                {
                    // put the code to handle the F1 key
                    FrmGotoForm(CoolForm);
                    handled = true;
                }
            }
            else    // not an AlphaSmart virtual key
            {       // get the Escape key being pressed
                if (eventP->data.keyDown.chr == chrEscape)
```

```
                {
                    EvtEnqueueKey(vchrLaunch, 0, commandKeyMask);
                    handled = true;
                }
            }
            break;
        }
    }
```

Keyboard Modifiers Example D

Example D in the Dana SDK provides an example of how developers can gain access to the current logical state of the modifier keys. The project will build an application that tracks the state of the modifiers by showing a form with a text for each modifier key. If the key is not pressed then the modifier key name will not be visible. If the modifier is currently down, pressed, then the modifier key name will be visible. A Graffiti shift indicator has been added to the form to demonstrate the system control of that object.

The `UpdateModifierIndicators()` routine is called from the form handle event routine.

```
static void UpdateModifierIndicators(void)
{
    KeyboardEvent   keyboardModifiers;
    FormPtr frmP;

    keyboardModifiers = KybdGetModifiers();

    frmP = FrmGetActiveForm();

    if (keyboardModifiers & KEYBOARD_MODIFIER_CAPS_LOCK)
    {
        FrmShowObject(frmP, FrmGetObjectIndex(frmP,
    MainCapsLockLabel));
    }
    else
    {
        FrmHideObject(frmP, FrmGetObjectIndex(frmP,
    MainCapsLockLabel));
}

    …
```

Chapter 5: WritePad Programming

The WritePad for the Dana is *virtualized*, so that the Graffiti areas and the icons are actually rendered on the LCD. This is what allows support for the minimized and maximized forms of the WritePad.

It is possible for applications to modify the WritePad area. One such example can be seen in the Figure below, which draws a globe over the WritePad.

BitMap Example E

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

Graphic on the WritePad

Drawing to the WritePad is as simple as retrieving a WinHandle and then drawing to the window using standard Palm window drawing routines. When the WritePad is maximized, Example E displays a picture of the circle with a number in the WritePad. The example shows a simple animation of numbers counting from 1 to 4. If the left/right button is pressed the WritePad will move and then about 1/3 of a second later the animation will disappear for a moment. The cause of the situation is that the WritePad detects input inside the writing area (tapping the left/right button), so it determines it needs to erase the pen input after the WritePad has moved.

The main form of the application displays information about the current state of the WritePad. A notification will be sent out whenever the minimize or left/right button is pressed. Information in the WritePad notification structure can be used to determine the current state of the WritePad.

The following code is used to register a WritePad notification:

```
error = SysCurAppDatabase(&cardNo, &dbID);
error = SysNotifyRegister(cardNo, dbID, notifyWritePadEvent,
        NULL, sysNotifyNormalPriority , NULL);
```

A notification will be delivered through a launch code (`sysAppLaunchCmdNotify`).

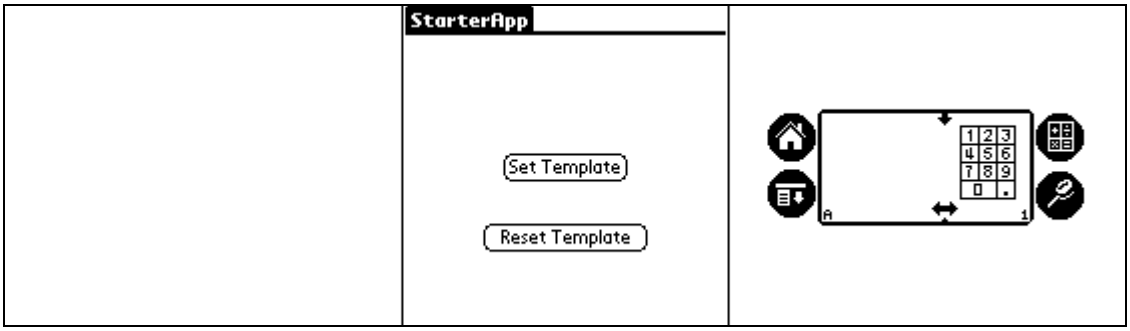
The changes made to the WritePad by Example E are not permanent because the WritePad Extension redraws the area on all program exits with the WritePad template. Refer to the *Dana API Reference* for a discussion on the complete Dana API.

Template Example F

Example F demonstrates how to make permanent changes to the WritePad and how to add custom buttons.

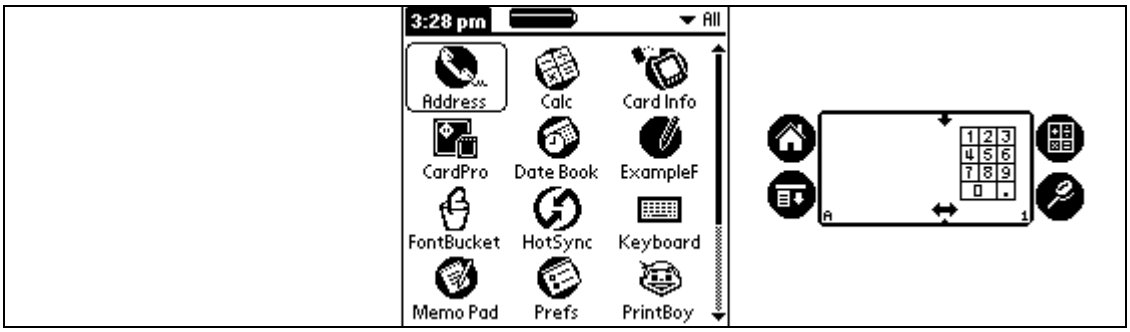
Each time an application exits, the WritePad Extension redraws the WritePad. The WritePad is redrawn from a template that can be changed by the developer with a call to `WrtpSetTemplateBitmaps()`. The WritePad Extension also provides the call `WrtpRestoreDefaultTemplate()` to restore the WritePad back to its default image and button list.

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



Example F

The Figure below shows the look of the WritePad after Example F has exited. The WritePad change remains even after the Launcher is launched. The change will remain until the unit is reset, when the WritePad is restored to its default configuration and removes the changes made by Example F. Example F could be modified to reinstall the new template when the unit is reset.



Permanent WritePad Change

Chapter 6: Fonts

AlphaSmart has licensed the Hands High Software's FontBucket technology for the Dana device. This technology enables applications to share fonts between applications. For more information on how to use this functionality from an application, please read the *FontBucket Developers.pdf* found in the SDK\docs directory

Fonts Example G

Example G demonstrates how to add FontBucket support to an application. It is very straightforward to add the support and users will enjoy the freedom of selecting their fonts in each application.

Chapter 7: Printing

AlphaSmart has licensed the Bachmann Software PrintBoy Documents technology for the Dana device. This technology enables applications to print Palm DOC files directly to USB or IrDA printers.

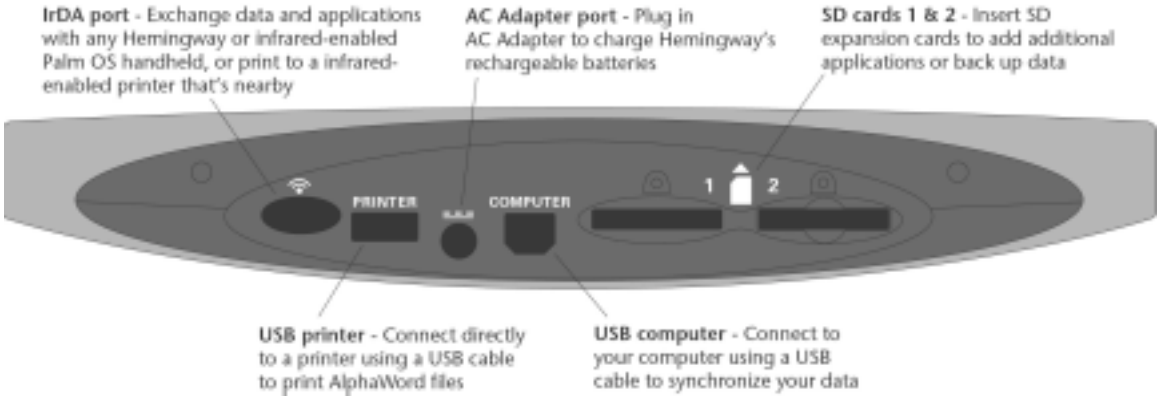
For more information on how to use this functionality from an application, please read the *Bachmann Software PrintBoy Technical Note* found in the SDK\docs directory

Chapter 8: VFS and Expansion Managers

Palm OS 4.0 introduced 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.



Expansion Slots

The Dana includes two Secure Digital (SD) expansion slots. The Dana provides compatibility with the 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 Dana fully implements the Expansion API as documented in Chapter 30: Expansion Manager of the *Palm OS 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: Compatibility Issues

Backward Compatibility

The enhanced features of the Dana are implemented as extensions to Palm OS version 4.1. 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 detailed discussion on accessing the feature manager, refer to the *Palm OS 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 Dana. This allows applications to be compatible with future generation products.