Palm Programming in Basic

JON KILBURN

Apress™

ISBN (pbk): 1-893115-49-6

Printed and bound in the United States of America 12345678910

Technical Reviewers: Hody Couch, Carl Ganz, George Henne, Frank O'Brien
Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Karen Watterson
Marketing Manager: Stephanie Rodriguez
Managing Editor: Grace Wong
Project Manager: Tracy Brown
Development Editors: Tracy Brown, Valerie Perry
Copy Editors: Tom Gillen, Ami Knox
Production Editor: Sofia Marchant
Compositor: Impressions Book and Journal Services, Inc.
Indexer: Nancy A. Guenther
Cover Designer: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010
and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.
In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com.
Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 901 Grayson Street, Suite 204, Berkeley, CA 94710.
Phone 510-549-5938, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 3

Introduction to AppForge

When I started writing code for the Palm platform a while ago, I found it to almost be a step backwards for me at first. I say this for a number of reasons, not the least of which is the step back from using SQL to using flat files. Perhaps it was because I had lost touch with my C programming side, but I think the real reason it took so long to get my software running on the Palm was because I found the resources (books, example code, documentation) available for developers either confusing, contradictory, or just plain poor.

AppForge

AppForge is a whole new way of programming for the Palm. Okay, maybe it's not entirely new in some respects (I'll cover what those points are later), but it's a darn good start. What is AppForge? Well, AppForge is a compiler written for Visual Basic that integrates into the VB IDE and allows you to compile from within the Visual Basic environment directly to a Palm device program file (or PRC file).

At last, VB programmers now have a compiler that (in some fashion) works using a familiar language. AppForge offers a great way to get up to speed on Palm application development.

That's not to say that AppForge is the perfect solution, but it brings the term Rapid Application Development (RAD) screaming into the Palm world. Before I jump right in and start talking about using AppForge in conjunction with Visual Basic to develop software for the Palm operating system, let me be clear on something. Doing development with Visual Basic for a different platform (in this case, the Palm OS) simply gives you the ease of a language you are familiar with, but in no way does this indicate that you can continue to write programs in the same manner that you do for Windows applications.

For example, consider that you have memory constraints within the Palm models themselves. Some Palm devices come with 2MB of memory, and some come with up to 8MB of memory. So the development (and what you can or cannot do) can be limited by the memory in the device. With these limitations of the

environment also come some limitations of the compiler. Make no mistake, AppForge is *not* an add-in or a library—it is a compiler (more on this later). Most of the supported functions are listed in the documentation, but there are a few hiccups you should be aware of (see the sidebar "AppForge Limitations").

AppForge Limitations

Here are a few of the limitations (and the reasons for some of them) that I've found with using AppForge:

- Standard API calls don't work, and if you think about it, this makes complete sense. API calls are direct programming interface calls to an underlying subsystem (in this example, I'm referring to Windows), so API calls to such functions as GetPrivateProfileString won't work, simply because the functions don't exist.
- The AppForge MsgBox function does not support the Title parameter. This parameter is determined by the severity of the message box displayed, and is set to either Error, Warning, Information, or Confirm. If a severity level is not specified via the Buttons parameter, the message box defaults to Information, unlike VB, which defaults to no icon.
- There are currently no pop-up forms. You can only have one window visible at a time.
- Use of the With clause is not supported. So you cannot use With [object] and End With.
- Use of For Each [object in collection] Next is not supported. This is most likely due to the fact that collections are not supported.
- Dynamic arrays are unsupported.
- Optional parameters are not supported. Presumably this is because the AppForge compiler tries to resolve all parameters and their types at compile time.
- The use of the vbModal parameter with Show is not supported. In this case, it appears that the use of vbModal would halt program execution until such time as the new form is unloaded, which would require maintaining a state within the calling form.
- The Format function is not supported. You instead have all the individual functions, such as FormatCurrency.
- The use of variant data types is not supported.
- Automatic data conversions are not supported by the compiler. By this I mean that when you assign an integer value to a string variable type, the AppForge compiler complains conversions of <type1> to <type2> may be potentially unsafe. Note the use of the word potentially. The compiler doesn't say, "Don't do this"—it says it's potentially unsafe, and then it pre-

vents you from compiling until you do explicit conversions using CStr(), CBool(), CLng(), and so on.
- When using the Palm Database (PDB) files, I could not find a method for filtering. You must search for records that match your condition, and then you must process the records and compare each one.
- Only ascending indexes are supported.

Check out the "Supported Visual Basic Functions" and "Unsupported Visual Basic Functions" documents in the online AppForge User's Guide (`http://www.appforge.com/dev/usersguide.html`). These documents give you a complete list of the functions that are supported and unsupported.

Okay, I know there will be some limitations, but this is not surprising. After all, this is a compiler for a different operating system (which has limited memory and uses flat files), so many of the restrictions imposed are there to help improve performance.

AppForge supports a very large subset of VB.[1] You will find that most of the more common Visual Basic functions are available and work identically to their desktop PC counterparts. Differences in language support are generally a result of limitations imposed by the Palm OS and represent decisions made by the AppForge compiler team when designing the AppForge compiler.

You will find that, for the most part, AppForge attempts to mirror Visual Basic, so much so that AppForge even has a comparable equivalent to the IntelliSense feature within Visual Basic.

NOTE *For most up-to-date articles about specifics of Visual Basic support, including techniques for optimizing your software using AppForge, you should visit the Developer Sector at* `http://www.appforge.com`.

AppForge was designed specifically to meet the special programming needs of the mobile devices. Speed, efficiency, and reliability are critical when writing software for these types of devices.

[1] For a complete list of the supported commands, you should take the time to review the "Supported Commands" section of the AppForge help file. It should be noted that with each release, AppForge adds new supported functions. In some cases, you will find function libraries on the AppForge Web site (`http://www.appforge.com`) under the Developer Sector Knowledge Base, such as the Format function library (the use of the Format function is not currently supported).

Beware!

There are a few things about AppForge that you need to be aware of, and I have listed them here.

- *Limited RAM:* Reading and writing to RAM requires power, which reduces battery life. As a general rule, it is advisable to reduce RAM usage wherever possible. Programming styles that are less RAM intensive will translate well to the mobile software environment. For example, the use of Public variables when not needed should be minimized; instead, include local variables to reduce the amount of memory in constant use.

- *Slower execution:* To minimize per-unit costs and maximize the life of small batteries, low-speed processors are often used (the current Palm processors are running at between 16 and 33 MHz). Because of this, certain more CPU-intensive features may not be available.

- *In-the-field updates are difficult:* Hence, mobile applications should be as reliable as possible. Preventable errors, such as those related to type safety, should be identified at compile time, rather than after deployment to a mobile device. For this reason, AppForge does not do some things at compile time that you might be used to getting away with. For an example, see Listing 3-1.

Listing 3-1. Code That Will Generate an AppForge Compiler Error

```
Dim cCharString As String
Dim nValue As Integer

cCharString = "12"

' Assign this value to a numeric, this thows an error in AppForge
' This works fine in VB because at runtime VB will resolve the
' value to an Integer doing an "on the fly" conversion.
nValue = cCharString

' To make it work in AppForge
nValue = Cint(cCharString)
```

All of AppForge's design decisions were made in an attempt to balance the issues in such a way as to present a useful and efficient subset of Visual Basic.

Virtual Machines

Now, I said I'd tell you a bit more about the AppForge compiler, so I'll do that here. AppForge is a compiler, but it also requires a virtual machine. This is similar to Java in that you need a runtime interpreter (or virtual machine) to execute your code. In older compilers such as Clipper (a DBase compiler), the runtime interpreter was actually compiled into the final EXE. You will also discover that several of the compilers available for the Palm (but not necessarily reviewed or explained in this book) also use this technique.

So what exactly is a virtual machine? The term *virtual machine* has been used to mean either an operating system or any program that runs on a computer.

A running program is often referred to as a virtual machine—a machine that doesn't exist as a matter of physical reality. The idea of a virtual machine is itself one of the most elegant in the history of technology, and is a crucial step in the evolution of ideas about software. In order to come up with it, scientists and technologists had to recognize that a computer running a program isn't merely a washing machine doing some dirty laundry. On the contrary, a washing machine is a washing machine no matter what kind of clothes you put inside, but when you put a new program in a computer, presto, by virtue of the new program it is also a new virtual machine.[2]

In the most recent computer usage, virtual machine is a term that was first used by Sun Microsystems (the developers of the Java programming language and runtime environment), to describe software that acts as an interface between compiled Java bytecode and the microprocessor (or hardware platform) that actually performs the program's instructions.

Once a virtual machine has been written for a particular platform, any program written to run within the specifications of that virtual machine will now run on that platform. Java was designed to allow application programs to be built that could be run on many platforms without having to be rewritten or recompiled by the programmer for each separate platform.

For example, take the Windows OS and Macintosh OS. Both are operating systems with their own very specific instruction sets. The goal behind Java is that a program written and compiled with the Java compiler could be ported from the Windows OS to the MAC OS by simply moving the code.[3]

Virtual machines are what make all this possible. The Java virtual machine has a specification, which is defined as an abstract, rather than a real machine (or processor), which specifies an instruction set, a set of registers, a stack, a garbage

[2] This discussion on virtual machines is based on the David Gelernter article, "Truth, Beauty, and the Virtual Machine," *Discover Magazine*, September 1997.

[3] Of course, this is the scenario played out as the ideal. In reality, there are so many steps to port an application written in Java for the Windows OS to the Macintosh OS, it would take another book to explain them all.

heap, and a method area. The real implementation of this abstract (or logically defined processor) is done in other code that is recognized by the real processor. The output of "compiling" a Java source program is called bytecode. A Java virtual machine can either interpret the bytecode one instruction at a time (mapping it to a real microprocessor instruction) or the bytecode can be compiled further for the real microprocessor using what is called a just-in-time compiler.

AppForge has taken the virtual machine approach and applied it to handheld devices. AppForge already exists for Palm OS and Pocket PC devices. In the case of AppForge, the virtual machine is called Booster. Before any AppForge application can run, Booster must be installed on the target device.

Using the Palm OS Emulator with AppForge

In a small aside here, I figured I should explain a few simple things about the Palm OS Emulator (POSE). First, Booster is required in order for any AppForge Palm application to run. Booster is a virtual machine, as it runs your compiled bytecode application. So, before you can test your application in the emulator, you must first install and run the POSE

When POSE has been installed and configured, you can then right-click the emulator to view a pop-up menu of choices. Once you select the Install Application/Database option, you can locate the PRC files in the Platforms\PalmOS\TargetImage directory under your AppForge installation and upload them to the emulator. Now, before POSE will function properly, you must reset it (similar to having to restart Windows after an installation). Once you reset the emulator, you can then install your application and databases on it. This is only required if you choose the Install option rather than using the HotSync menu option.

Before You Start Programming

In this section, I'll give you a small list of the things you need to start programming, and some specific tips I learned as I became familiar with AppForge.

The Palm SDK

Although you do not need anything except AppForge, a copy of Visual Basic (you can use any edition of Visual Basic 6, including Visual Basic Working Model,

which is included on the CD), and either a real Palm OS device or the Palm OS Emulator (POSE) to get started, I would suggest that you obtain a copy of the Palm OS SDK, including the latest documentation (available from Palm Web site). You can also download their Conduit Developers Kit 4.01 for COM, which supports any COM-compliant language, including Visual Basic. This means that you can now develop custom conduits[4] using Visual Basic.

NOTE *Neither the Learning Edition or the Working Model of Visual Basic allow you to compile a Windows EXE file. This is not a problem, as AppForge installs itself as an add-in with its own AppForge menu. From the AppForge menu, you can select the Compile option, which will compile your VB code as a Palm OS–compatible application file (PRC).*

The SDK contains three important items:

- *Example code:* You can peruse this code (although it's all in C/C++) to get a feel for how POSE works.

- *Documentation:* Think of the documentation as a dictionary in which you can look up all the possible system functions available for your code. The SDK documentation is not very interesting to read, unless you find reading a dictionary exciting.

- *Palm OS Emulator:* The emulator, shown in Figure 3-1, essentially functions as a virtual Palm on your desktop. POSE allows you to debug and run the programs you develop on your computer, instead of performing a HotSync every time you make a new build. You definitely need the emulator, unless you enjoy resetting your Palm device several times while writing your software. (See Chapter 2 for more information on using the POSE.)

[4] A conduit is a means of communication between a Palm OS device and a Desktop PC. For a more detailed explanation of conduits, see Chapter 9.

Figure 3-1. The Palm OS Emulator

The 1000-Foot Overview

Before you write any code, let's take a moment to review some simple basics. Since you are probably new to the AppForge compiler and the Palm OS, you should consider that not everything you want to do can necessarily be done the way you did them in VB. Let me give you an example:

In AppForge, there is currently no support for dynamic arrays. This means that if you want to have a dynamic array, it cannot be done in the traditional sense (that is, using Dim, ReDim. . .Preserve). However, you can easily solve this problem. The trick is quite simple. Using an AppForge AFListBox control (or a grid if you need multiple elements), which is hidden on a form, you place the values you want to serve as an array in the listbox (see Listing 3-2).

Listing 3-2. Code to Simulate a Dynamic Array

```
Private Sub ArrayAdd(ByVal cElement As String)
'-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
'
'    Sub          :   ArrayAdd(cElement)
'    Params       :   cElement - Element to Add
'    Returns      :   None
'
'    Author       :   Vivid Software Inc. - Jon Kilburn
'                      http://www.VividSoftware.com
'
```

```
'   Client       :   Apress
'   Purpose      :   Add Array Element to hidden Listbox
'
'-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

    ' Array Insert
    Me.lstArray.AddItem cElement

End Sub

Private Sub ArrayDel(ByVal nPos As Integer)
'-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
'
'   Sub          :   ArrayDel(nPos)
'   Params       :   nPos - Element to Delete
'   Returns      :   None
'
'   Author       :   Vivid Software Inc. - Jon Kilburn
'                    http://www.VividSoftware.com
'
'   Client       :   Apress
'   Purpose      :   Add Array Element to hidden Listbox
'
'-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

    ' Array Delete
    Me.lstArray.RemoveItem nPos

End Sub
```

So what's my point? Simply this: just change your thinking slightly, and you can accomplish almost anything in AppForge for the Palm OS that you can do in Windows. The code just won't look exactly the same.

Basically there are three steps to building an AppForge Palm application:

1. Create the application in Visual Basic and compile it using AppForge to a PRC file.

2. Install Booster onto the Palm device.

3. Upload your AppForge application.

There will be some additional steps, such as creating a device package or converting Access files to Palm Database files and the like, but overall this is the way it works. Now let's get to it.

Getting Started

Once you have completed the installation of the AppForge compiler (by running the Install program), you will notice that when you start Visual Basic, you may now select a new type of application, an AppForge application (see Figure 3-2).

Figure 3-2. The AppForge project

You have two ways to create an AppForge project. The first way is by selecting Start | Programs | AppForge and then clicking the Start AppForge option. This will open the AppForge Project Manager window, which prompts you to edit an existing project or to create a new one (see Figure 3-3). The other way is to open Visual Basic and choose to create a new project. You then select AppForge Project from the New Project window. Once you have opened the new project, a single form is added to the project. The form properties are set according to the size of the Palm window. Changing the form size will result in an AppForge compiler error.

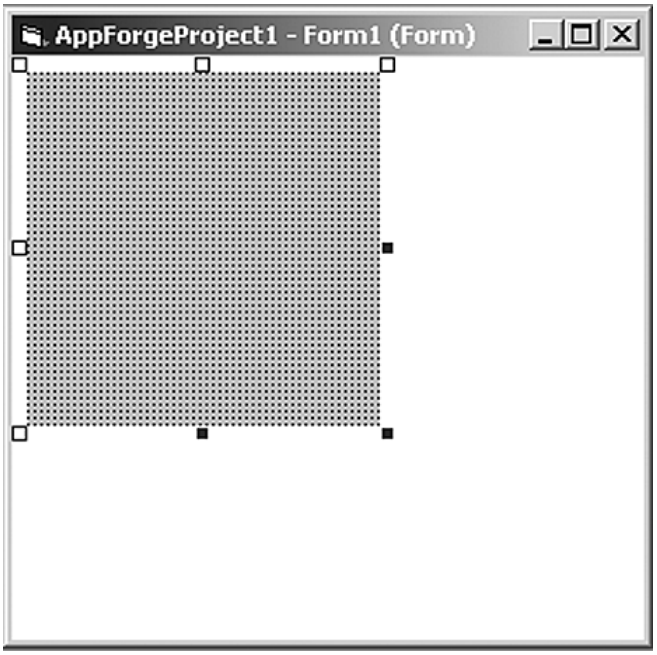

Figure 3-3. The AppForge form

Once you have started to create a new AppForge project, if you are using the Professional Edition of AppForge, you will be prompted to choose the type of AppForge project (see Figure 3-4).

Figure 3-4. The AppForge Select Target Platform dialog box

It is important to note that although you may still see the standard list of events and properties, not all of the displayed events or properties are supported by AppForge. This is not because AppForge has not made an effort to do so, but rather because the Palm OS is different from Windows. A classic example of this is the MsgBox() function.

The standard Visual Basic MsgBox function consists of the following parameters:

```
MsgBox(Prompt, [Buttons], [Title], [Helpfile], [Context])
```

The AppForge compiler supports the MsgBox() function, but the parameters are different. The AppForge version of the MsgBox() function has the following form:

```
MsgBox(Prompt, [Buttons])
```

Notice that there is no support for the Title, Helpfile, or Context parameters. Neither the Helpfile or Context parameter is supported, and the Title value is determined by the severity of the message box displayed, and is set to Error, Warning, Information, or Confirm. However, the Visual Basic IntelliSense feature will *not* display only those parameters that AppForge supports. This is true of all

the AppForge functions, which do not have all the same corresponding parameters that exist in their Windows counterparts.

If a severity level is not specified via the Buttons parameter, the message box defaults to Information, unlike with Visual Basic, in which the message box defaults to no icon. The AppForge MsgBox does not support the following constants:

```
VbApplicationModal
VbSystemModal
vbMsgBoxSetForeground
```

The reason for the lack of support is that all of these values are meaningless under Palm OS. Along those same lines, none of the vbDefaultButtonX parameters are supported—after all, there are no default buttons on a handheld device since there is no Enter key (unless you're using an attached Palm keyboard). There are several differences like this one that run throughout the AppForge compiler. As I mentioned earlier, you should study the "Supported Visual Basic Functions" document of the online AppForge User's Guide so that you can familiarize yourself with AppForge's flavor of the Visual Basic functions.

NOTE *In AppForge, only* one *form can be shown at a time. Therefore, before showing a new form, the old form must be hidden first. (Otherwise, an "overlapping" form error will occur.)*

Generally, you should use the Hide method when a form is no longer needed (setting the Visible property to false and then showing the form by setting Visible = True can cause weird behaviors on the actual Palm device). This prevents the overlapping error and allows for access to values on the hidden form. If no forms are shown, the screen will be blank. Be aware that you should try to keep from going too deep when hiding forms, as most Palm devices have less than 256KB (yes, that's kilobytes) of memory for the application to work with. Although I have successfully managed to go as many as five forms deep (using a Symbol 1500 with 8MB of RAM), the practical limit is probably around three forms. I say this because the limit can vary based on the number of controls you have on a form.

NOTE *Hiding an application that has only one form and no modules will also exit the application.*

Also, another trick I've used is to create an Owner property in the called form. By setting an Owner property you can create a "stack" which allows you to return to the previous form. Why do I do that instead of just using the form name? Two reasons:

First of all, since there is no support for vbModal when a form is shown, you cannot keep it visible or halt your code until the called form is unloaded. Usually you would perform a search lookup like so:

```
Sub cmdLookup_Click()
          frmLookup.Show vbModal
End Sub
```

In an application I developed, the same lookup form could be called from a couple of different places (that is, from different forms). So I had to figure out how to get around this problem—I solved it by creating an Owner variable in the frmLookup form.

```
Public Owner As Form
```

There is a caveat to doing this. You can only reference the functions and values of a Form object. You cannot call any specific Public subs or functions you may have designed for your form, unless you declare the Owner with the explicit name of your form. This has to do with the AppForge compiler resolving data types at compile time to prevent errors at runtime.

Next, when I called the frmLookup form, I hid the current form, set the owner of the frmLookup form, and then showed the whole thing:

```
Sub cmdLookup_Click()
          ' Hide this form
          Me.Hide
```

```
            ' Set the Owner form
            Set frmLookup.Owner = Me

           ' Show the Lookup form
           frmLookup.Show
End Sub
```

Next, in the cmdClose_Click event of the frmLookup, I simply showed the Owner form:

```
Sub cmdClose_Click()
           ' Hide this form
           Me.Hide

           ' Show the Owner
           Me.Owner.Show

           ' Unload this form
           Unload Me
End Sub
```

The second reason: This allows me to have a global return function, which can be called by all forms and requires less code.

The AppForge Ingots

Now let's move on to building some forms. In your toolbox, you will now find a new list of controls (see Figure 3-5). These controls are called *Ingots*. Ingots are AppForge ActiveX components that are similar to standard Visual Basic controls. They provide the feel and functionality of Visual Basic controls and are programmed using properties, events, and methods. The only difference is that they work in Windows, Windows CE, and Palm OS. Please note, these controls currently do not have the full range and functionality of Visual Basic controls.

Figure 3-5. AppForge Ingots added to the Visual Basic Toolbox

NOTE *The Ingots shown here are provided with AppForge Professional Edition.*

There are currently 24 Ingots available for the AppForge compiler. Not all of these Ingots are available with each version (as I note later); however, by the time you complete the sections of this book on AppForge, you should have a good grasp of how to use each of these Ingots.

- AFButton
- AFCheckBox
- AFClientSocket
- AFComboBox
- AFFilmstrip
- AFGraphic

- AFGraphicButton
- AFGrid
- AFHScrollbar
- AFINetHTTP
- AFLabel
- AFListBox
- AFMovie
- AFRadioButton
- AFScanner
- AFSerial
- AFShape
- AFSignatureCapture
- AFSlider
- AFTextBox
- AFTimer
- AFTone
- AFVScrollbar
- Form

The AppForge Converters

AppForge provides a suite of converters and viewers that convert standard media files to AppForge-specific files. There are currently five converters and their associated viewers available for the AppForge compiler:

- Font
- Graphic
- Movie
- Project (this converter migrates prior AppForge Projects to 2.0)
- Database

Not all of these converters (like their Ingot counterparts) ship with each version of AppForge. All AppForge software editions include a graphic converter and viewer, project converter, and a database file converter.

AppForge Professional Edition provides additional converters and viewers for fonts and movies. Table 2-1 summarizes the file types supported by the AppForge file converters and viewers.

Table 3-1. File Types Supported by AppForge Converters and Viewers

MEDIA	DESCRIPTION	OUTPUT FILE TYPE
Database	Microsoft Access databases (*.mdb)	Palm Database file (*.pdb)
Graphics	Standard Windows bitmap (*.bmp)	AppForge graphic (*.rgx)
Font*	True Type Font (*.ttf)	AppForge font (*.cmf)
Movie*	Windows AVI File (*.avi)	AppForge movie (*.rvm)

* Denotes those converters included only in the Personal and Professional Editions

I will review the converters more in depth in the next chapter when I demonstrate how to build a more complicated AppForge application than the one presented later in this chapter.

The AppForge Shipping Versions

Currently there are two shipping versions of AppForge, AppForge Professional Edition and AppForge Personal Edition. AppForge Professional Edition is available for Palm OS, Pocket PC, or both.

NOTE *Times change, and so do companies, versions, and shipping editions. The current shipping information is based on what AppForge has listed as their available editions on both their Web site and those of their preferred vendors (such as Handango and VBXtras).*

The AppForge Personal Edition

The Personal Edition is the low-cost, entry-level development version. This is an ideal starting point for the hobbyist programmer or the programmer who is working on learning Palm programming at his or her own pace. It allows you to begin programming and developing applications for the Palm OS without having to spend a whole lot of money. This edition has some limitations, but overall is a good value, and if you decide to upgrade, AppForge will give you 100 percent credit towards the Professional Edition.

The AppForge Personal Edition offers the following:

- Access to several commonly used Palm Functions such as the Date Picker
- Most AppForge Ingots (see `http://www.appforge.com/prod/featurelist.html` for a complete listing)
- The ability to communicate through the serial and infrared ports
- A low introductory cost
- Purchase price is applied when upgrading to AppForge Professional Edition

The AppForge Professional Edition

The AppForge Professional Edition is the heavy hitter. This high-level development platform is geared for the serious developer who may wish to create applications using the advanced features (such as scanner integration and wireless Internet support) of the Palm device. The AppForge Professional Edition supports all the features in the Personal Edition, and adds the following power-user enhancements:

- *Palm OS Extensibility Library:* This library allows developers to augment AppForge features with code written in C/C++.

- *Wireless Internet support:* The AppForge Wireless Internet Ingot makes it easy to add the Internet to your application. A stock quote sample is included with the Professional Edition to help you get started writing Internet-enabled Palm applications.

- *AppForge AFScanner Ingot:* The AFScanner Ingot can be used to provide barcode scanning for inventory control, point-of-sale operations, and identification, and for reading virtually anything that has a barcode. The AFScanner Ingot supports Symbol Technologies' SPT-1500/1700 Symbol with Palm OS 3.5 (you must upgrade the OS from version 3.1), and the CSM-150 Barcode Scanner for the Handspring Visor.

- *The Universal Conduit:* The UC is a program that allows databases to be synchronized between AppForge-created applications[5] and ODBC data sources without writing code—that is, it creates a conduit without any code (for more information on the Universal Conduit, see Chapter 9).

- *The Font Converter/Viewer:* The Font Converter will convert any TrueType font to an AppForge font file for use on Palm OS devices (this is an exclusive feature of AppForge made possible by Booster).

- *The Movie Converter/Viewer:* The Movie Converter will convert standard Windows AVI files for use with the Movie Ingot. Please note that at the time of this writing, there are some limitations as to the size of AVI files that can be converted and stored on the Palm devices.

- *The Signature Capture Ingot:* The Signature Capture Ingot adds signature input, storage, and display capabilities to your AppForge applications.

- *AFScrollBar and AFSlider Ingots:* The AFScrollbar and AFSlider Ingots provide a mechanism to produce a range indicator. They have a minimum and maximum setting.

- *AFClientSocket Ingot:* The AFClientSocket Ingot allows TCP/IP communication on network-enabled handheld devices.

[5] The UC does not always work with applications that have been developed in other languages. This is due to the fact that AppForge writes schema information into the header of Palm Database files, which may not be available if the PDB file has been created using another tool.

NOTE *Upgrading the Symbol 1500/1700 to the 3.5 OS is described in detail at both the AppForge Web site (in the Developer Sector Knowledge Base) and the Symbol Web site.*

Working with Menus

In Windows, it's a common practice to use a menu for assigning and grouping actions. Not surprisingly, AppForge 2.0 supports menus. Menus provide a way to access multiple commands without occupying too much screen real estate. Each menu should contain one or more menu items, to which a command should be associated. You can visually group items using a separator bar, just like in Windows. Menus themselves are contained within a menu bar, and there can only be one menu bar per form. To create a new menu, select either the Menu Editor option from the Tools menu or click the icon on the standard toolbar.

Follow these steps to create a menu test program:

1. Start by creating a new Visual Basic AppForge project.

2. Name the application MenuTest.

3. Create the main form by renaming the default Form1 form as frmMenu.

4. Select the Tools | Menu Editor option.

5. Add a Main Menu Item named Test, set the Name property to mnuMain, and set the index property to 0 (see Figure 3-6).

6. Click the Next Button. This adds a new blank menu.

7. Press the Indent Button (the right arrow key).

8. Name the subitem Item1. Set the name property to mnuItem and the index to 1.

9. Now save the project.

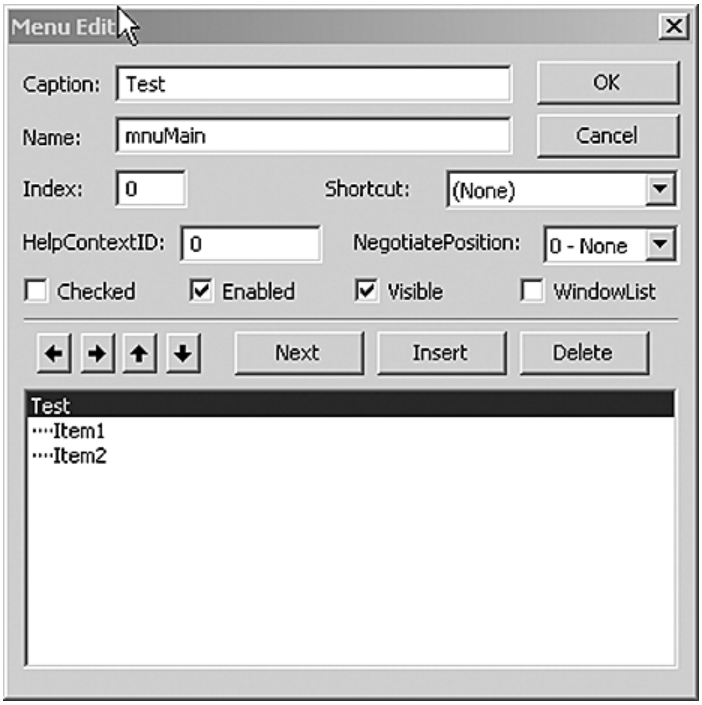

Figure 3-6. Using the Visual Basic menu editor

From the AppForge menu, select Compile and Validate. This will compile the project and validate all the referenced objects. When you choose to compile an AppForge project for the first time, you will be prompted to enter a CreatorID (see Figure 3-7).

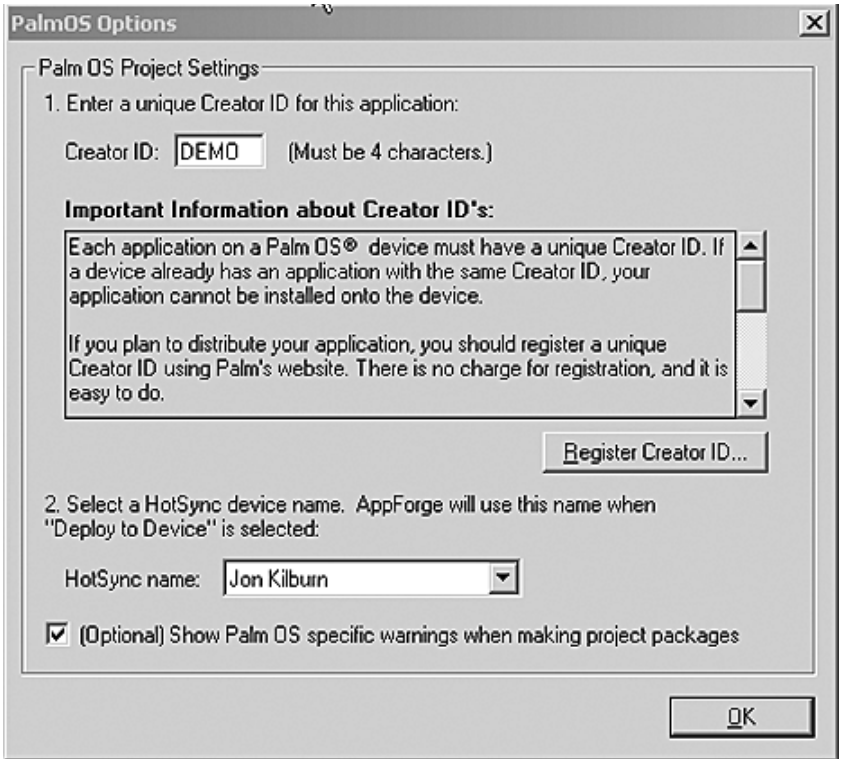

Figure 3-7. Entering a CreatorID

Before you can run any AppForge program, you must first install the AppForge virtual machine, or Booster, on the target device. Once the compile process has been completed, you can now create the Booster download. Booster comprises a number of separate runtime files. These files are combined on the Palm device to make up the final Booster based on what features you have included in your AppForge application.

NOTE *If more than one user is set up to use the Palm desktop software, a window will prompt you to select a user to receive Booster. Once Booster has been copied, the Install directory file list (see Figure 3-8) will show all of the files to be installed on the specified handheld device during the next HotSync operation.*

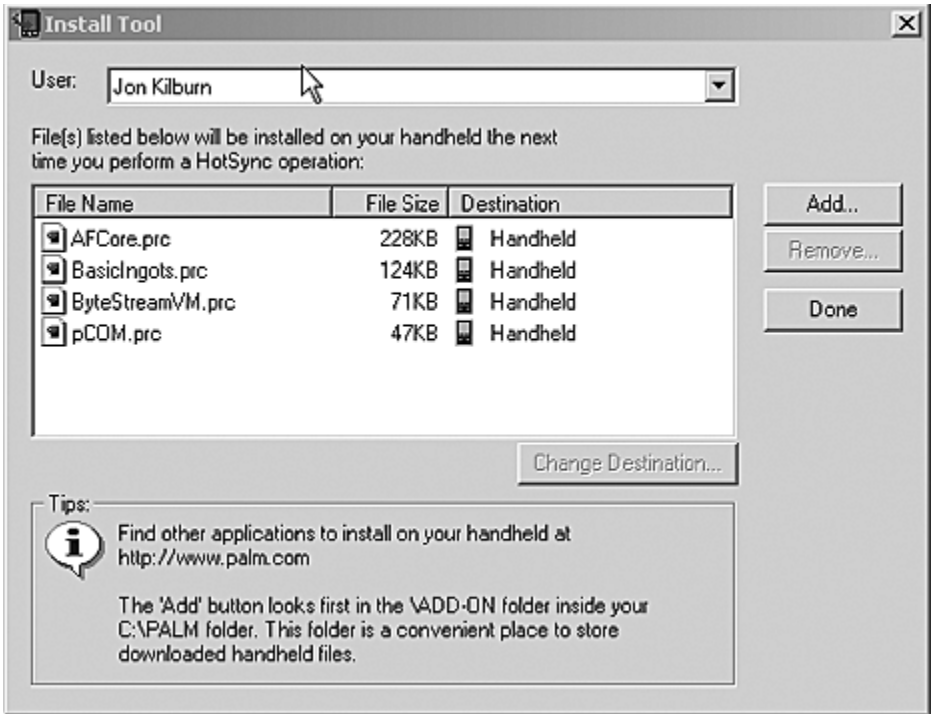

Figure 3-8. Booster files set for installation

To create the Booster package, access the AppForge menu and select the Install Booster To Device option. From the Booster options submenu, select the Palm OS platform. This will copy the required Booster files to the installation directory for installation during your next HotSync.

Once Booster has been installed on the target device, you can then install your MenuTest application. You have a couple of ways to do this. Starting from the AppForge menu, you can choose Deploy to Palm OS | Deploy to Device. Next, select the Palm OS option, or select the Save Project Package option, which will generate the Palm application in the folder of your choosing. Figure 3-9 shows the final menu working in the POSE.

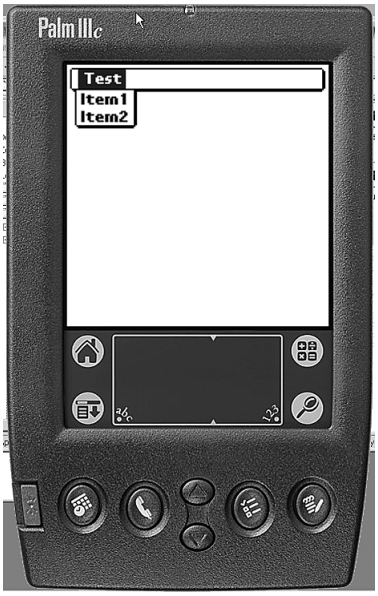

Figure 3-9. The final menu running under POSE

Building Your First AppForge Application

Now that you've seen the basic components of AppForge, let's move on to creating a simple application and syncing the application to a Palm device. For this first program, let's build a simple application to calculate the daily production rate for piecework assembly. I got this idea from a program I wrote for one of my clients, which tracks assembled inventory. Since my client uses robots for product assembly, this program really wouldn't do the client much good, but it would be very useful to help a line supervisor estimate the average assembly time and average daily production for a given product.

Creating a VB AppForge Project

This simple application will take the number of pieces that are assembled in a minute and calculate the assembled production estimates for one hour and for a standard eight-hour day. Start building this application by creating a new Visual Basic AppForge project. Follow these steps:

1. Name the application ProductCalc.

2. Create the main calculation form by renaming the default Form1 form to frmCalc.

3. Clear the form's Caption property. If the form caption is set, AppForge will automatically create a Palm title bar for your form.

4. Now save the project. You have to save the project before AppForge will allow you to add any controls.

5. Next, select the AFLabel control from the Visual Basic Toolbox and drop the label onto the AppForge form.

6. Change the name of the AFLabel to "label".

7. Change the background color of the AFLabel to black and the foreground color to white.

8. Set the Alignment of the label to 2-Center, and change the FontName property to AFPalm Bold 12.

9. Set the caption of the label to Production Calculator.

10. Next, size the label to the width of the form and set the Height property to 16. AppForge uses pixels for its unit of measurement.

11. Add a label and set the caption to Assembled Pieces. Make this label's Height property 16, and change the font to AFPalm Bold 12. Now center the label.

12. Change the background to black and the foreground to white like the previous label. Next, change the name of the label to "label", and when Visual Basic prompts you to create a control array, select Yes. The label will automatically be renamed Label(1). Position this label slightly above the middle of the form.

13. Now add three more labels with the captions Per Minute:, Per Hour:, and Per Shift:.

14. Select an AFTextBox control from the Visual Basic Toolbox and place it to the right of the Per Minute: label.

15. Although it looks like you have three textboxes, you actually do not. Select the AFLabel control again and place two labels to the right of the Per Hour: and the Per Shift: labels. In the Properties window, change the Border Style property to be 1- Fixed Single. Next, change the Alignment property to 1- Right Justified. Lastly, change the Caption property to have a value 0. Now these two labels look like AFTextBox controls.

16. Drop a couple of AFButton controls on the form and name them cmdCalc and cmdClose. Change their captions to Calculate and Close.

17. Finally, make the form look a little more spiffy by adding a few AFShape controls. Set the AFShape controls' Border Width property to 1 so it appears to be a line instead of a rectangle. Now place three of these lines around the controls connected to the Assembled Pieces label to make a box. Figure 3-10 shows what your final form should look like.

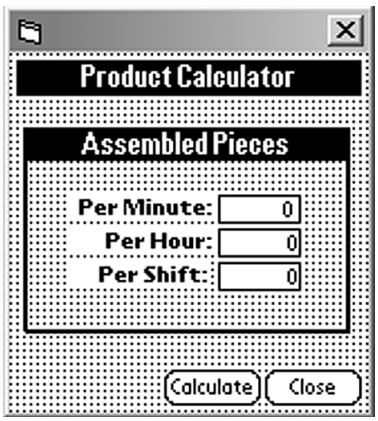

Figure 3-10. The completed Product Calculator form

Writing the Code

Now comes the fun part, adding the code. First, you'll write the code to exit the program. When exiting the Production Calculator program, you should confirm that the user does indeed wish to exit. Do this using the AppForge equivalent of the MsgBox function (see Listing 3-3).

Listing 3-3. Exit Code for Production Calculator

```
Private Sub cmdClose_Click()
'-=========================================================
'
'    Sub          :    cmdClose_Click()
'    Params       :    None
'    Returns      :    None
'
'    Author       :    Vivid Software Inc. - Jon Kilburn
'                       http://www.VividSoftware.com
'
'    Client       :    Apress
'    Purpose      :    Confirm Exit
'
'-=========================================================

    If MsgBox("Exit Production Calculator?", vbQuestion + vbYesNo) = vbYes Then
        ' Quit the Program
        End
    End If
End Sub
```

Now let's build the code to perform the piecework calculation. Assume that there are no breaks taken in each hour so you can calculate against a true sixty minutes. The shift workers get one full hour for lunch; however, they work eight hours in a shift (so they are scheduled in nine-hour increments). To calculate the pieces per hour, multiply the contents of txtPieces by a value of 60. You then take the resulting value and multiply it by 8, which will yield the number of estimated pieces per shift (see Listing 3-4). Write the resulting values into the Caption property of the two labels, LabelHour and LabelShift.

Listing 3-4. Calculation Code

```
Private Sub cmdCalc_Click()
'-=========================================================
'
'    Sub          :    cmdCalc_Click()
'    Params       :    None
'    Returns      :    None
'
```

```
'   Author      :   Vivid Software Inc. - Jon Kilburn
'                    http://www.VividSoftware.com
'
'   Client      :   Apress
'   Purpose     :   Calculate the number of pieces per hour,
'                    and per shift.
'
'=================================================================
    Dim nHour   As Long
    Dim nShift  As Long

    ' Set Trap
    On Error GoTo Trap

    ' Calculate
    If Me.txtPieces.Text = vbNullString Then
        ' Must have a valid value
        MsgBox "Please enter the number of pieces.", vbExclamation
        Exit Sub
    Else
        ' Calculate Number of Pieces assembled per hour
        nHour = CInt(Me.txtPieces.Text) * 60

        ' Per Shift (8 Hours per shift)
        nShift = nHour * 8

        ' Fill in the labels
        Me.LabelHour.Caption = Trim(CStr(nHour))
        Me.LabelShift.Caption = Trim(CStr(nShift))
    End If

Exit_Rtn:
    Exit Sub

Trap:
    MsgBox "Sytem Error!" & vbCrLf & _
        Err.Description, vbExclamation

    GoTo Exit_Rtn

End Sub
```

Running Your Application

Once you have completed the code, you can now run the resulting AppForge application from inside the Visual Basic and Windows environment by selecting the Run menu option. This will launch a form that will look and act like a Palm form.

CAUTION: *Just because the application runs in the Windows environment without errors does not imply that it will function properly when uploaded to the Palm device. You should always upload your compiled project to a Palm OS device—at the very least the POSE—for testing.*

Now that you have tested the basic application functionality under the Windows environment, you should compile the Production Calculator using the AppForge compiler. Select the AppForge menu option and choose the Compile Project option. This will launch the AppForge compiler. When compiling a project, AppForge analyzes the code for errors and possible conflicts that may prevent the compiled program from functioning properly once it has been uploaded to the Palm OS device. When the compile operation is successful, the progress window will disappear to indicate the operation is complete.

With the application fully compiled, you must now begin the process of installing and testing the finished application.

Once you have performed the HotSync operation, the Booster.prc application will appear in your Palm device's main applications area (see Figure 3-11).

NOTE *For the POSE, you can also right-click and choose the Install Application/Database menu option. If you install Booster using this method, you must then select the Reset option.*

Figure 3-11. Booster in the Palm main applications area

Now that you have installed Booster on the target device, you have two options for uploading the compiled application. Although you have compiled the application, you have not created a device package (this is the step where the actual .PRC file is created). If you select the AppForge menu option Upload Project, then AppForge will create a PRC file and also copy the file into the Palm Install directory (in the same manner as it does for Booster). Upon your next HotSync, your application will be copied onto the target device. If you choose the Create Device Package option, AppForge will create the PRC file, but not place it in the Palm Install directory.

Finally, once you have selected the Deploy To Palm OS menu option (or used the Palm Install tool to select the ProductCalc.prc file) and specified you want to HotSync the file to the target device, the new program, Production Calculator, will now appear in your program group. Figure 3-12 shows the final compiled and uploaded application running.

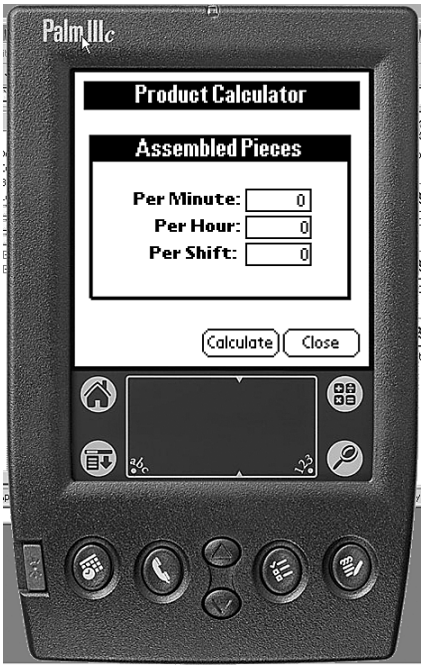

Figure 3-12. The Production Calculator running in the POSE

The Project Properties

The final item I would like to discuss is project preferences. From the AppForge menu, select AppForge Settings. This will bring up the AppForge Settings dialog box (see Figure 3-13). From this dialog box, you can control all aspects of the AppForge project.

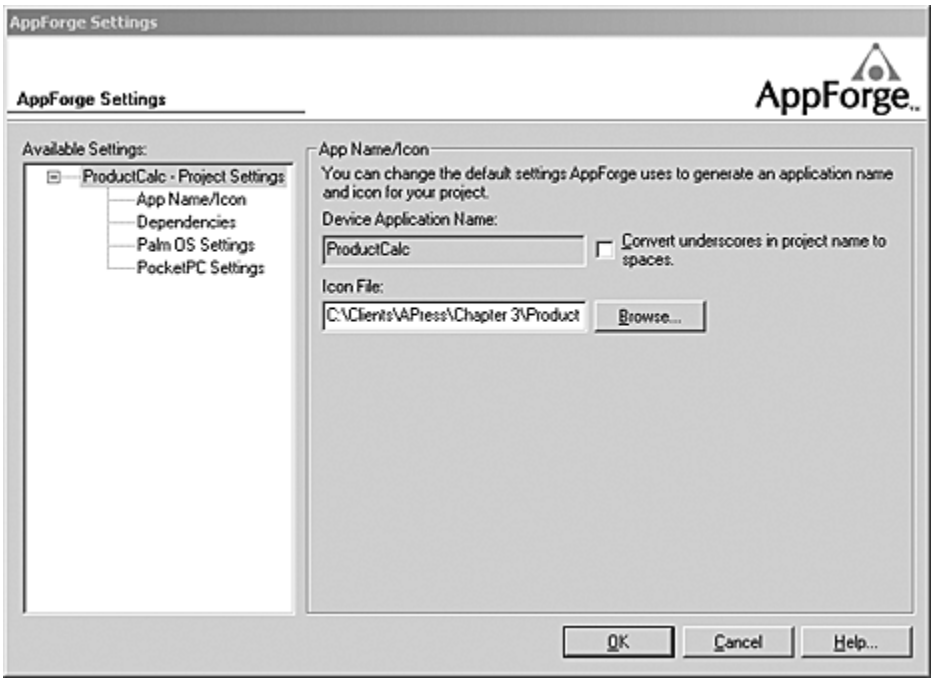

Figure 3-13. The AppForge Settings dialog box

The first item in the tree view is the App Name/Icon setting. Here you will see the name of the application (as it will appear on the Palm), and you can also assign an application icon.

The second item in the tree view is Dependencies. When you highlight this item, the main panel to the right will change to display the dependencies of your project (see Figure 3-14).

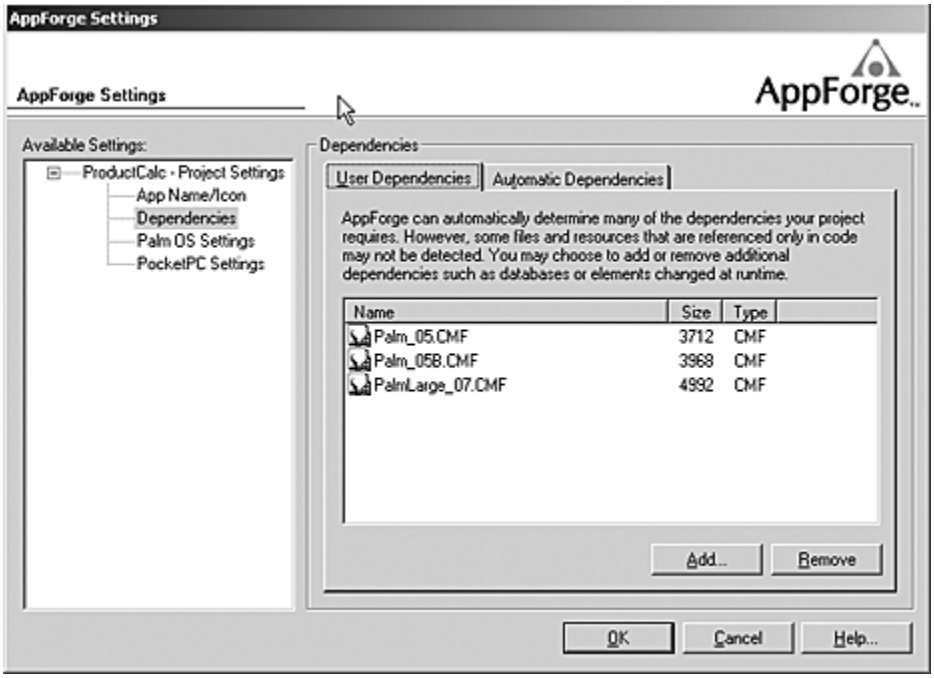

Figure 3-14. The AppForge Dependencies section of the AppForge Properties window

The next two items are the Palm OS and Pocket PC Settings. The Palm OS settings option simply contains the Creator ID information I mentioned earlier when saving a project.

Conclusion

In this chapter, I've given you a brief overview of how the AppForge compiler works and how to use it to build a simple application. I've also discussed the basic principles behind how to compile, install, and test your final application in the Palm environment.