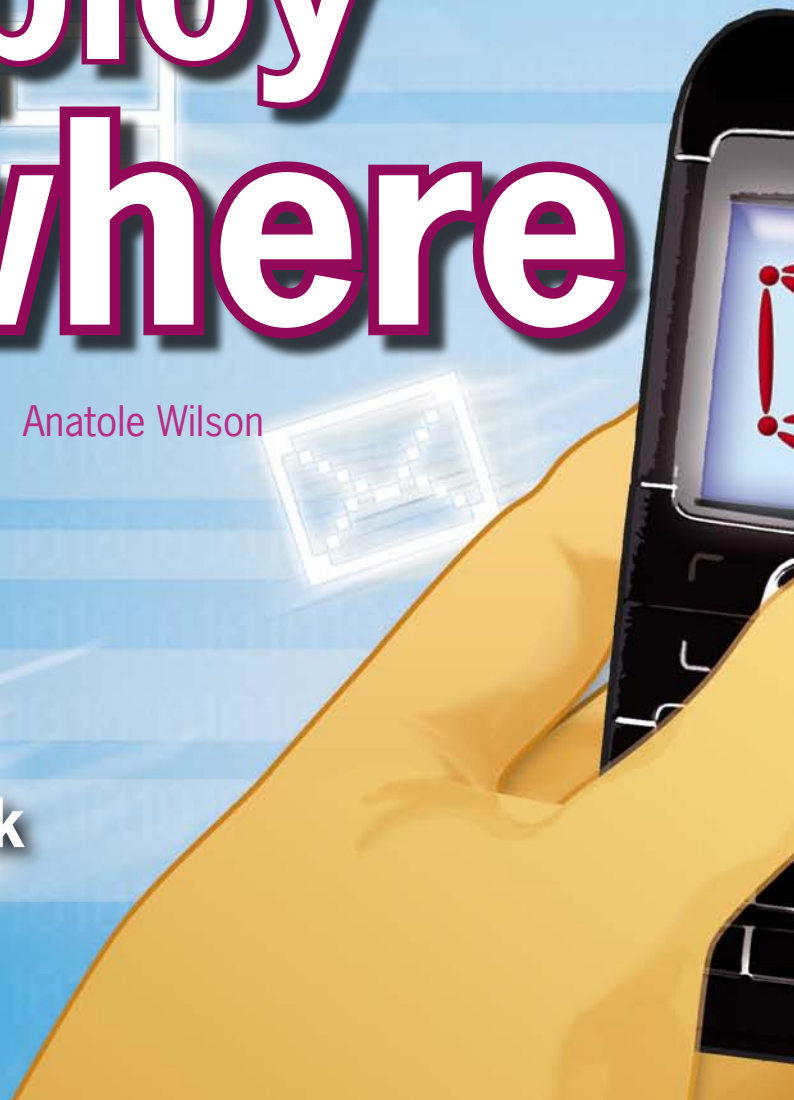
Write Once Deploy Anywhere

Anatole Wilson

Writing Applications for mobile devices and reducing device fragmentation with NetBeans Mobility Pack



Take a look around you in just about any location – your office, walking down the street, the movies, even your own household – and it won't take a marketing genius to tell you that if your application is useful to people on the go but hasn't been ported to phones or PDAs, you're missing a pretty large market, potentially *millions* of users. Of course, that same look around will tell you that your game or widget had better be able to work on mobile devices of all names and sizes, or you'll cut out a large portion of your potential customer base.

Sure, it sounds like a lot of work. But in this article, we'll show you how to use the NetBeans Mobility Pack 5.5 for CLDC/MIDP to create a simple application suitable for cell phones and then deploy it to two very different devices. Best of all, you'll do it using only a single set of code instead of creating and managing a separate code base for each device.

In addition, we'll take a quick look at the Mobility Pack 5.5.1 for CDC and show you how you can use it to develop applications for larger devices, like set-top boxes, PDAs, and the new generations of "smart" phones.

A brief overview of the Mobility Pack

With NetBeans 5.5, there are now two separate versions of Mobility Pack – one for each of the standard *configurations* that specify the minimum requirements for memory, Java language features, JVM support, and runtime libraries.

In Mobility Pack 6.0, the CLDC/MIDP and CDC Packs will be merged.

The Mobility Pack for Connected, Limited Device Configuration / Mobile Information Device Profile (CLDC/MIDP) serves developers who create applications targeted at smaller mobile devices with limited resources, such as mobile phones, PDAs, and two-way pagers.

The Mobile Information Device Profile is one of two profiles that works "on top" of CLDC (the other profile being the Information Module Profile). MIDP provides graphical interfaces for interactive applications and is the standard profile for mobile telephone development under the JSR 185, Java Technology for the Wireless Industry (JWTI) specification, and more recently also the Mobile Service Architecture (MSA) spec, JSR 248.

The Mobility Pack for CDC provides a development environment for developers who are creating applications for larger devices, such as set-top boxes, embedded devices, high-end or smart phones, and PDAs. The CDC Pack supports development for all the current CDC Profiles, including the Foundation and Personal profiles, as well as the AGUI toolkit. It also supports development for the Windows CE platform.

netbeans.org/kb/trails/mobility.html

The Mobile Applications Learning Trail guides you through a structured approach to learning about the Mobility Pack.

How the two Java ME configurations fit into the Java "family" is shown in **Figure 1**.

In the next section, we'll create a MIDP application, or *MIDlet*, using the Mobility Pack for CLDC/MIDP. Then we'll look at how you can create an application using the Mobility Pack for CDC. It's worth noting here that with version 6.0, the two Packs will be integrated into a single Mobility module.

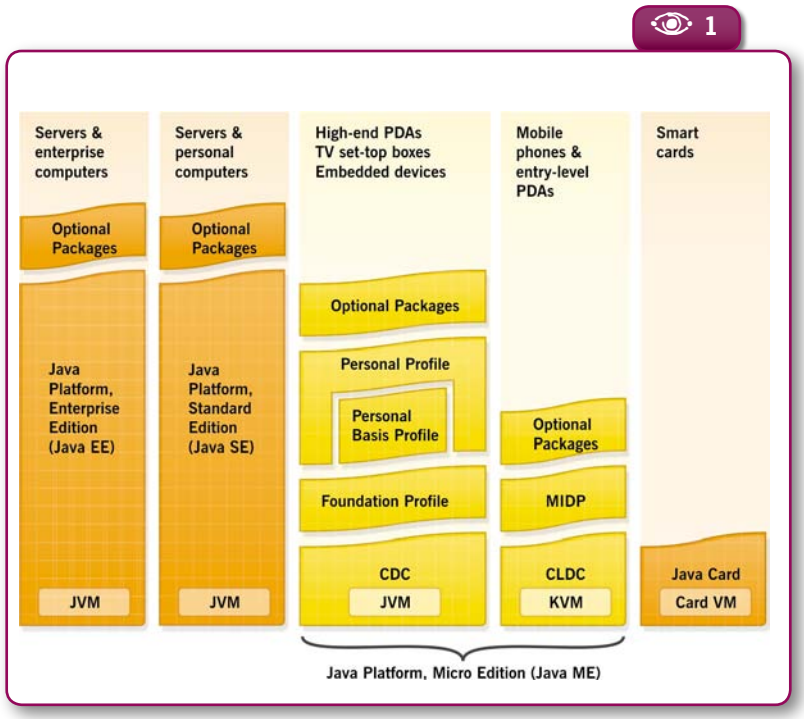


Figure 1 Java ME and the Java technology landscape.

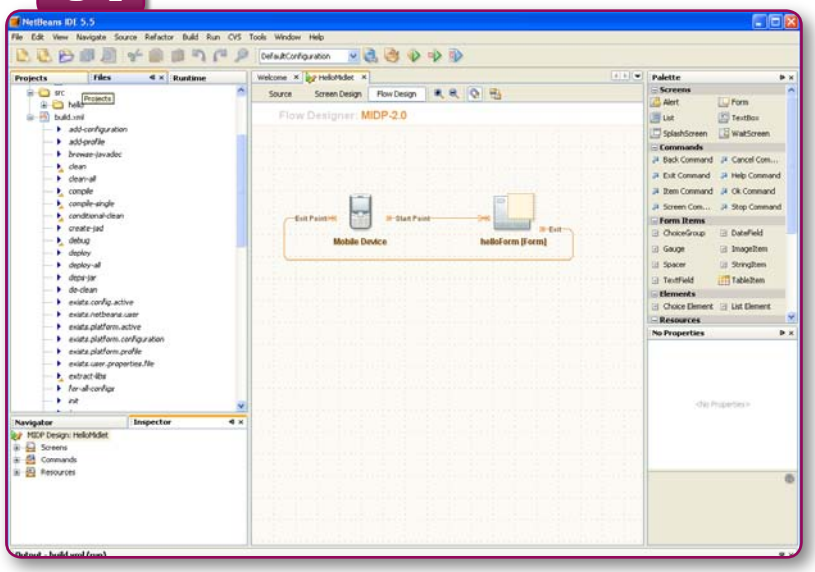


Figure 2 The Hello MIDlet Application .

Creating a MIDlet

We're going to start with a very simple "Hello World" MIDlet. Although this is about the simplest MIDlet you can create, it will let us demonstrate the Mobility Pack without too much focus on writing code.

To begin working with the Mobility Pack, you create a new project just as you would with any Java application, only selecting a Mobile project type.

In the NetBeans IDE, as you know, all Java development has to take place within a project. The IDE builds its project infrastructure directly on top of Apache Ant, and stores all the information about your project in an Ant script, a properties file, and a few XML configuration files. This means that you can build and run your projects outside of the IDE in exactly the same way as you would inside it. As you can see in the source tree shown in **Figure 2**, however, a MIDP project differs from a NetBeans/Java SE project in several ways.

When a MIDP project is created, the IDE creates folders to hold the files after they are compiled and preverified. (Preverification is a Java ME-specific building step that augments *.class* files with extra information that makes bytecode verification and classloading much simpler and faster on the resource-restricted JVMs of ME devices.) The preverified files are packaged into the distribution JAR of your project. If you use project

configurations for different device types (we'll get to that soon) or use obfuscation, the IDE creates folders to hold the different versions of the source files.

Now let's create the Hello MIDlet project. Choose *File>New Project* and in the first step of the wizard select the *Mobile* category and *Mobile Application* for the project type. Then click *Next*. Change the project name to "TestMIDlet." Click *Next* again to see the default Emulator Platform, which is the J2ME Wireless Toolkit. The default device is the DefaultColorPhone.

Click *Next* to see some other project configurations you can choose to set. The IDE checks all of the platform emulators you have installed and creates a configuration template for each device emulator available. Check the "MediaControlSkin_ template" box. Click *Finish* and you've just created your first working MIDlet, containing a simple form named **helloForm**, that displays the familiar "Hello, world!" message. Now Choose *Run>Run Main Project* to see the MIDlet in action. You'll see a device emulator like the one shown in **Figure 3**.

Figure 3
Hello MIDlet in action.

Designing flow and UI in the Visual Mobile Designer

The screenshot in **Figure 2** also shows you the flow of our MIDlet in the Visual Mobile Designer (VMD). You can click the *Source* button and edit the code manually, but the VMD is a very powerful tool for designing both the page flow and individual screens.

Let's look at the flow first. If you start with the mobile device on the screen, it's pretty easy to follow the arrows and understand the flow of the MIDlet. When a command is entered (the Start Point), the **helloForm** is opened. When the exit command is chosen from this form, you close your application (Exit Point) and return to the mobile device's application launcher.

Now, you might want to add a splash screen to display your logo or development credits before the application starts running. Let's add a simple one to this application. (To provide a graphic, we've added *mobileduke.png* to our *<project_source>/src* directory. The

Figure 4
A splash screen in the Screen Designer.

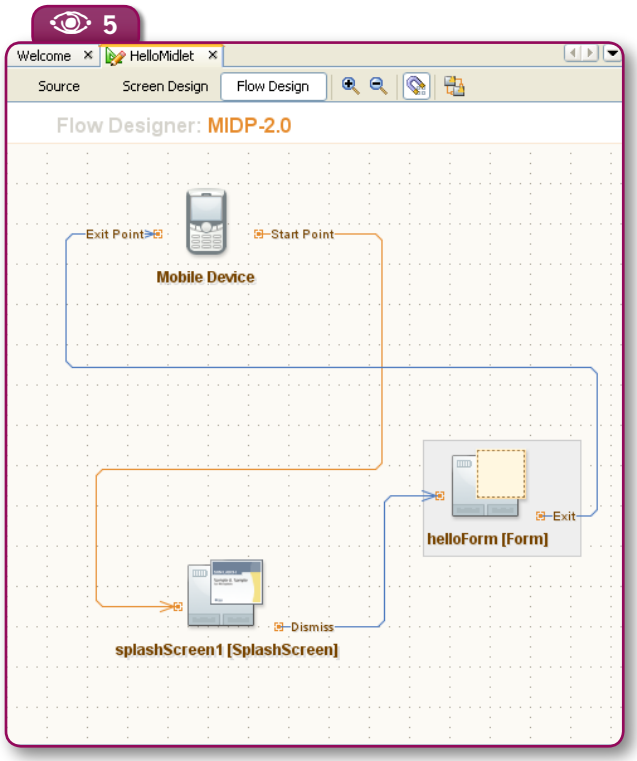


Figure 5 Application flow with a splash screen.

IDE automatically makes it available to us from the *Resources* folder in the Navigator window, on the lower left side of the IDE.) Drag a SplashScreen component from the Tools Palette on the right of the window on to the Flow Designer. Next, Drag the **image1[Image]** from the Navigator window and drag it on top of the splash screen.

Presto! Our splash screen has been created. You can look at it by double-clicking on the SplashScreen component, which takes you into the Screen Designer. The splash screen should look like **Figure 4**.

Now let's make the splash screen part of the application flow. Click the Flow Design button to go back to the Flow Designer. Grab the tip of the Start Point Arrow and drag it over to the **splashScreen1** component. Click on the orange square next to Dismiss and drag the arrow over the **helloForm** component. The flow should now look something like **Figure 5.**

Let's just make one more quick change to the program to illustrate the Screen Designer. Double click on the form component. In the Screen Designer, click on the "Hello, World!" text and change it to something else, say "Hello, Universe!"

Now go ahead and run the MIDlet again. Click the button under "Launch," and the emulator displays the splash screen as shown in **Figure 6**. Then you see the new message.

That was just a simple example of how you can quickly design the screens and flow of your application in the VMD. The SplashScreen component is one of three custom components, along with WaitScreen and TableItem, created by the NetBeans Mobility Pack team and added to the palette to make visual programming easier. You can also create your own custom components and add them to the palette.

Another important VMD feature is its support for Scalable Vector Graphics (SVG). SVG is an XML-based standard defined by the W3C and supported by MIDP 2.0 through JSR 226. The compact size and consistent appearance across different platforms and screen resolutions makes it an attractive graphics format for mobile developers. SVG also enables scripting and animation that allows users to interact with

Figure 6 Splash screen in the device emulator.

Mobility Pack for CLDC/MIDP documentation index
netbeans.org/kb/55/mobility.html

Figure 7 An interactive SVG-based menu.

the visual content. To use SVG, you'll need the Sun Java Wireless Toolkit 2.5, which is currently available as a module from NetBeans Update Center, and is bundled with the Mobility Pack starting with version 5.5.1.

In the VMD, you can add an external SVG editing tool, such as Hyperion or Ikivo, and use it to create your initial SVG graphic. Like we did in the example we just looked at, you can create a splash screen, an interactive menu, or a wait screen by dragging and dropping components into the Flow Designer. You can drop the graphic or animation on the component, and inspect the behavior of the graphic or animation as the application is run.

Figure 7 shows the VMD inspecting an animated SVG menu. You can use the VMD flow designer interface to link each menu item to a separate screen that would be called when the menu item is selected.

Reducing fragmentation with project configurations

One of the most difficult aspects of developing applications is *device fragmentation*. Mobile devices differ in a variety of attributes, such as screen size, color depth, and the proprietary or optional APIs they support. These differences often require special code or project settings for successful deployment. One solution is to create separate source code for each device you're programming to, which is almost guaranteed to be a logistics nightmare. We've already touched on the Mobility Pack solution for device fragmentation – *project configurations*.

Project configurations enable you to define the execution environment for each target device. With project configurations and code pre-processing, you can write an application and – using a single set of source code – customize, debug, and deploy a separate distribution JAR for each target device. If you need to customize your MIDlet for more devices, you add a new configuration for each device, modify the project properties, add some pre-processing code, then build and deploy the application. In most cases, you should create one configuration for each distribution JAR you plan to build for your project. For example, if you are planning to support three different screen sizes using two sets of vendor specific APIs, you should create six configurations.

Before we look at deploying our MIDlet to different devices, let's examine the three main concepts behind project configurations.

The Emulator Platform

An emulator platform simulates the execution of an application on one or more target devices. It enables you to understand the user

The Sun Java Wireless Toolkit 2.5

The Sun Java Wireless Toolkit 2.5 includes all of the advanced development features found in earlier versions, such as MIDlet signing, certificate management, integrated OTA emulation, push registry emulation, and more. New features include support for the Mobile Service Architecture (JSR-248) platform. Although this JSR does not define any new APIs, it does standardize many existing ones into a common API stack, to increase interoperability and make mobile development easier. There are also many new APIs supported, such as Security and Trust Services (JSR 177), Location (JSR 179), SIP (JSR 180), Content Handler (JSR 211), Scalable 2D Vector Graphics (JSR 226), Payment (JSR 229), and several others.

netbeans.org/kb/55/quickstart-mobilitycdc.html Mobility Pack for CDC Quick Start Guide

experience for an application on a particular device, and to test the portability of the application across different devices. As you have seen, the J2ME Wireless Toolkit 2.2, bundled with Mobility Pack 5.5 for CLDC/MIDP, provides several sample devices, like the DefaultColorPhone (and you can easily update to WTK 2.5).

It is important, however, to remember that an emulator can only approximate a device's performance. Environmental variables like processing speed or the strength of the wireless signal can affect performance on a real device, and should be taken into account.

A very important feature of the Mobility Pack is its ability to work with the emulators provided by major manufacturers, such as Nokia, Sony Ericsson, Siemens, and Motorola. Using the Java Platform Manager (see **Figure 8**), you can easily add any emulator that supports the Unified Emulator Interface (UEI) standards. Simply choose *Tools>Java Platform Manager*, select the J2ME Platform, and the wizard detects the emulator platforms installed on your system. Other emulators can also be added with a little more effort.

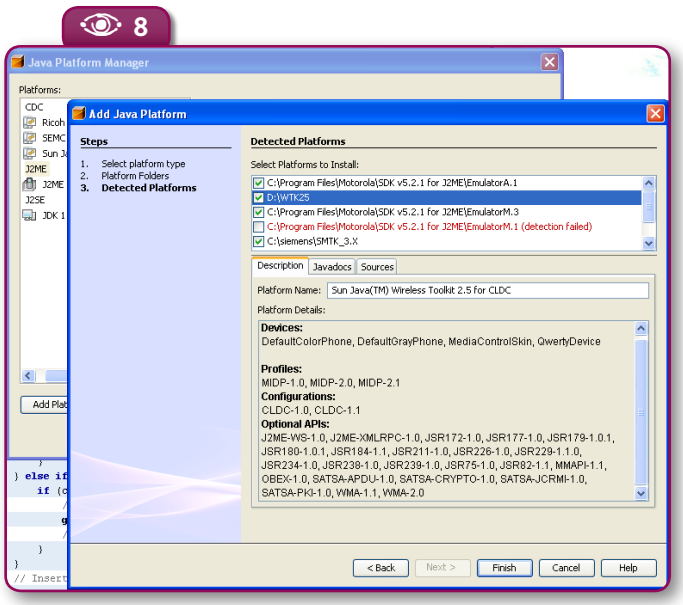


Figure 8
The Java Platform Manager.

Project properties

We can use the project properties to define many aspects of the program. As you can see in **Figure 9**, the property categories are on the left, and the properties for that category are on the right. A short list of things you can do in properties includes: defining the emulator platform, setting/checking Configuration and Profile versions and optional APIs the device supports; adding or removing the contents of the JAR and JAD files; setting the Push Registry; setting obfuscation and optimization levels; adding signing and security certificates; and setting deployment options. The Abilities shown in **Figure 9** list attributes that might be shared by different devices, and therefore might be shared when you are adding pre-processing code.

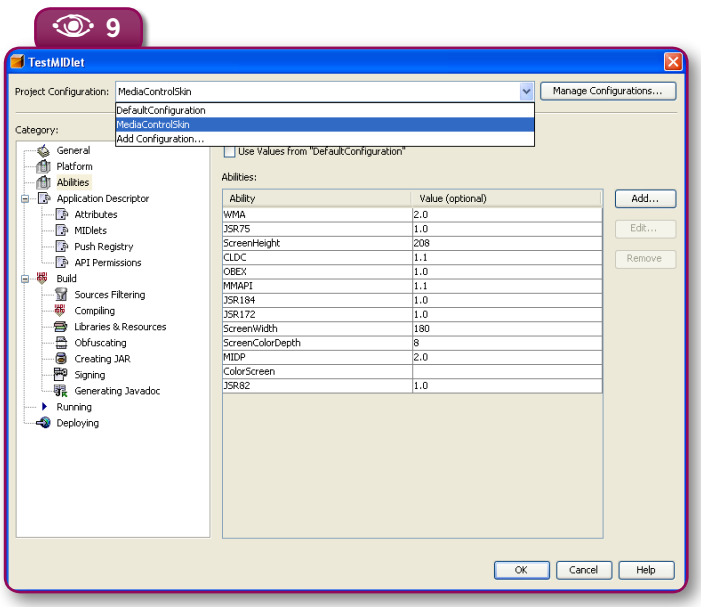


Figure 9
Abilities page.

Let's take a look at this using the two configurations we've created for our **TestMIDlet**. The first is the DefaultConfiguration we've been using so far. The second is the MediaControlSkin that you selected when first creating the project. Before we discuss project properties and pre-processing code, let's take a quick look at the MediaControlSkin device emulator.

To switch configurations, choose the

MediaControlSkin from the Configuration drop-down menu in the IDE toolbar. Then choose *Run>Run Main Project*. It's the same **TestMIDlet** as before, but this time the device emulator and its display are thinner – so we lose a little bit of the splash screen, as shown in **Figure 10**.

Preprocessing code

Preprocessing modifies the code in your source files before the code is parsed by the compiler. The preprocessor modifies the code according to preprocessor directives you insert into the code as code blocks with beginning and ending directives. These code blocks are marked visually in the Source Editor and are included (or excluded) when you build the JAR for a specific project configuration or ability. You can use these code blocks to create, manage, and track code that is specific to one or more project configurations or abilities.

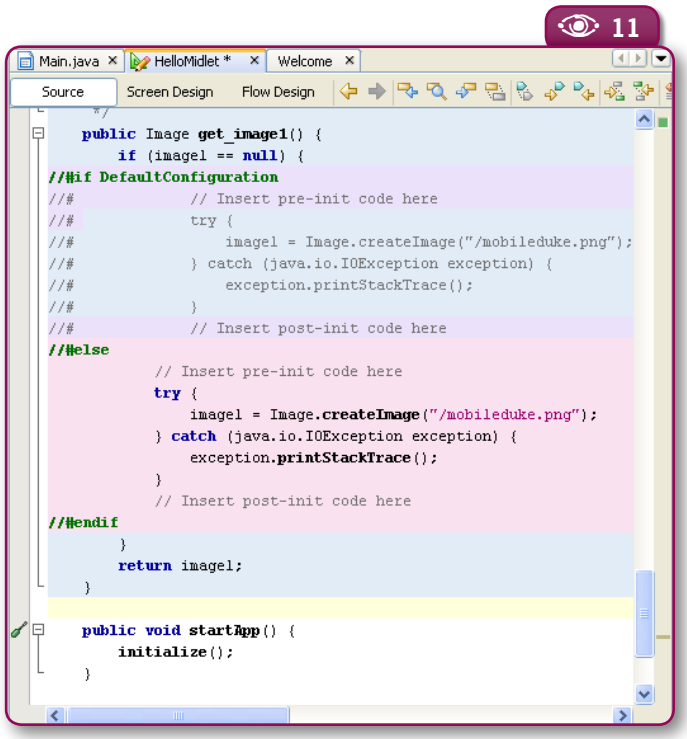


Figure 11 Preprocessor code.

Figure 10 The MediaControlSkin Emulator.

Now that we've defined the key concepts of project configurations, let's go back to our example and say now that we want to use a different graphic for the MediaControlSkin device. We already have our two project configurations defined, so what we need to do is add some pre-processing code so that the compiler knows it should use one image for DefaultConfiguration and another for MediaControlSkin.

Click the *Source* button to view the **TestMIDlet** source code. Then scroll down to the **get_image1()** method. Notice that some sections have a blue background – these are "guarded blocks" that are generated by the VMD, and cannot be edited. Highlight the code beginning with "//Insert pre-init code here and ending with "//Insert post-init code here." Right click on the selection and choose *Preprocessor Blocks>Create If/Else Block*. A menu with all the available configurations and abilities appears. Double click on DefaultConfiguration. Your code should look like **Figure 11**.

Notice that the **//#else** directive has a pink background. This code *can* be edited. Change the graphic name "/mobileduke.png" to the name of another graphic (such as "/veryproudduke.

netbeans.org/kb/55/mobilitycdc.html Mobility Pack for CDC documentation index

png"). Now, when you run our MIDlet using the DefaultConfiguration, the emulator will display the "mobile duke" image. When you run the MIDlet using the MediaControlSkin configuration, or any other configuration you add later, the emulator will display the second graphic. This is a very simple example of what's possible with preprocessor blocks, but it hopefully gives you a taste of what they can do.

Deploying to multiple devices

Now that you have an application that works with two devices, it's time to deploy it. The deployment property page shown in **Figure 12** shows the different deployment methods available.

Because deployment is set in the project Properties, you can define a different deployment for each configuration. When you have chosen your deployment method, choose *Build>Build All Main Project Configurations*. Then you'll have a JAR for each target device you're programming for.

Other features

Our example was purposefully kept simple to focus on the design, configuration, and deployment features of the Mobility Pack for CLDC. But there are two other important features we'd like to mention quickly before we move on to the CDC Pack.

The End-to-End Bridge technology is a set of two wizards that enable you to quickly modify your MIDlet to consume Web Services. The J2ME Web Service Client creates a client-side proxy that connects directly to Web Services that support the JSR-172 (J2ME Web Services) specification. The Mobile Client to Web Application Generator generates a servlet that connects to a web application that includes a Web Service client.

Another important feature is JMUnit testing support. The Mobilty Pack for CLDC provides built-in JMUnit support for generating and executing unit tests for MIDP/CLDC applications. You can generate and navigate to tests by selecting any class or package node in the Projects window and choosing from the *Tools>JUnit* menu.

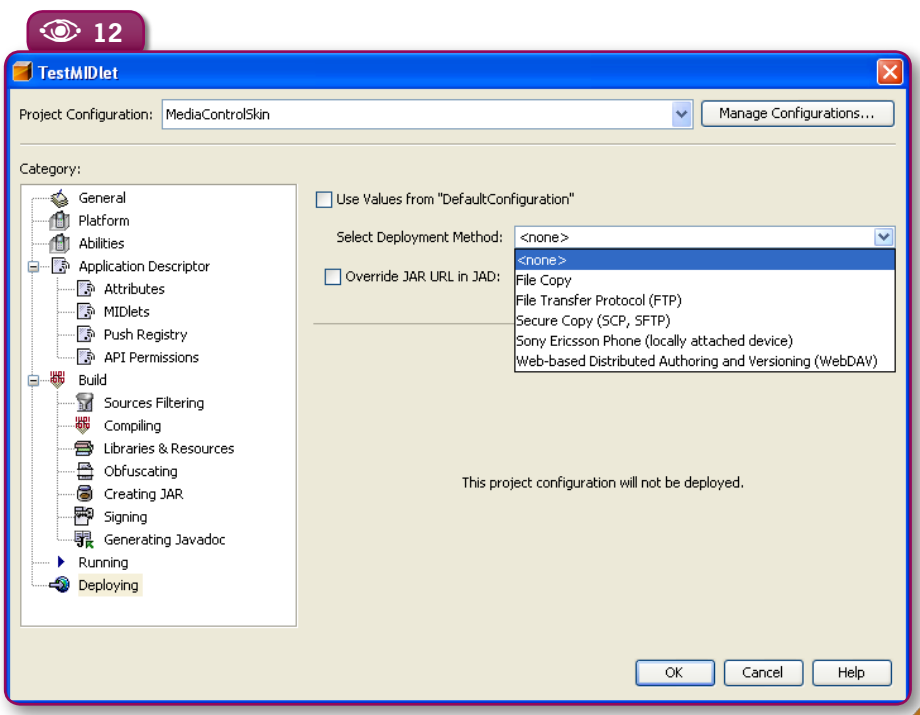


Figure 12 Deployment properties.

NetBeans Mobility Pack for CDC

The Mobility Pack for CDC (Connected Device Configuration) makes it possible to create, test, and deploy applications for several CDC platforms including the Sun Java CDC Toolkit, Sony Ericsson CDC Platform 1, Nokia S80, SavaJe and Ricoh MFP, as well as Windows CE using NSIcom's CrEme CDC virtual machine. Although it is not yet as complete as the MIDP/CLDC Pack, many new features will be added when Mobility Pack 6.0 is released.

You create CDC projects in the same manner as MIDP/CLDC projects, using the New Project Wizard. Before you begin, you will want to install the Sun Java Toolkit for CDC or an emulator platform from one of the growing list of manufacturers the Mobility Pack supports. You can find this list in the NetBeans Mobility Pack for CDC Quick Start Guide.

To get started on a project, Choose *File>New Project*. Choose the category *CDC*, project type *CDC Application*. The wizard will guide you through the rest of the steps for creating your Main Project.

Once you've created a project, you can use the Matisse GUI Builder with either the AGUI toolkit or the Personal Profile 1.0, in the same way you would use it for regular Java SE development. For Personal Profile GUI development, right click the Main.java form in the GUI Builder, and choose *Set Layout>Free Layout*. Then drag and drop components from the Palette window into the Design Area of the GUI Builder. You can also take advantage of JUnit testing and other key features of the NetBeans IDE when developing CDC applications in the Mobility Pack for CDC.

What's coming in NetBeans Mobility Pack 6.0

The NetBeans Mobility Pack has many dramatic changes coming up. One of the most significant changes, as we've mentioned before, is that the CLDC/MIDP and CDC Packs will be merged into a single UI, making it easier to create end-to-end applications. Other exciting new features include:

- New custom components to simplify programming, including a File Browser, an SMS Composer, a Login Screen and a Personal Information Manager (PIM) Browser
- VMD support for the MIDP 2.x Game API that allows creating tiled and animated layers for environment design, and support for animated character and sprites.
- Improved VMD UI, including support for non-visual components and a design analyzer.
- CDC support for project configurations and pre-processing blocks.

Conclusions

This article was intended to give you a hands-on sense of the capabilities of the Mobility Pack for CLDC/MIDP and CDC, and a running start on building your first mobile application. You can learn more about the Mobility Pack by reading the tutorials and articles on the NetBeans website, joining the NetBeans community of developers, and most importantly, by going out there and creating great mobile applications!

Anatole Wilson (*anatole.wilson@sun.com*) lives in Pittsburgh, Pennsylvania and has been the Senior Technical Writer for the Mobility Pack since its inception. He has worked for various high-tech companies, including IBM and Oracle, and, as a freelance writer, has written articles for publication in various magazines.