F O R U M   N O K I A

# Nokia Developer's Suite for the Java™ 2 Platform, Micro Edition, Version 3.0 for Windows

## User's Guide

April 26, 2005

NOKIA

## Contents

## Disclaimer

### License

# Nokia Developer's Suite for J2ME™ User's Guide

Version 3.0 for Windows; April 26, 2005

# 1 Introduction

This User's Guide describes how to configure and work with Nokia Developer's Suite 3.0 for J2ME™ for Windows. Nokia Developer's Suite 3.0 for J2ME™ is a software package intended for developers creating MIDP and Personal Profile applications. To get most out of Nokia Developer's Suite 3.0 for J2ME™ and this guide, you should be familiar with Java programming [1], Mobile Information Device Profile (MIDP) [2], Personal Profile (PP) [3] and the Connected Limited Device Configuration (CLDC) [4] and Connected Device Configuration (CDD) [5].

## 1.1 Setting up Nokia Developer's Suite 3.0 for J2ME™

To install Nokia Developer's Suite 3.0 for J2ME™, run the installation software. See installation guide [7] for more information on installing the product. During installation, you can choose to install Nokia Developer's Suite 3.0 for J2ME™ as either a stand-alone program or integrated with Borland JBuilder®, Sun™ Java Studio / NetBeans or Eclipse platform's Java Development Tools (JDT).

Nokia Developer's Suite 3.0 for J2ME™ is designed for use with Java™ 2 SDK Standard edition 1.4.1 or later, earlier Java™ versions are not supported. Uploading of applications files using server deployment via FTP requires network access to the FTP server. For more information, see section 10.2.

In this User's Guide, Nokia Developer's Suite 3.0 for J2ME™ is used as a stand-alone application. Additionally, three chapters have been dedicated for its use with Borland JBuilder®, Sun™ Java Studio / NetBeans or Eclipse platform.

## 1.2 Product registration

The product comes with a 14-day trial period with full functionality. After that you need to register it. A Forum Nokia account is required for the registration. If you are not a Forum Nokia member, register (free of charge) at www.forum.nokia.com. After that you can register the product with your username and password.

If you continue to use the product after the trial period, a registration wizard will open. You can also register the product before the trial period expires by selecting *Help* | *Register now...* from the menu bar. Follow the wizard's instructions to continue using the product.

## 1.3 Product overview

Nokia Developer's Suite 3.0 for J2ME™ allows content creation for MIDP and Personal Profile. It provides you with tools for creating classes, creating application packages and signing MIDlet plus emulating and deploying MIDlets and Personal Profile applications among other things. Tools are loaded dynamically during start-up and you can customise the loaded tool set to suit your needs. All the tools in Nokia Developer's Suite 3.0 for J2ME™ are described in detail in the following chapters. In addition to content creation tools, utility tools exist for setting up the working environment. Working environment includes emulators, devices and working directories. See chapters 4, 5 and 16 for detailed instructions on using these utility tools.

To create application for MIDP, you need to use a MIDP capable SDK, for example Nokia Prototype SDK 2.0 for J2ME ™, which is installed during the Nokia Developer's Suite 3.0 for J2ME installation. And for Personal Profile development you need to use Personal Profile capable SDK, Series 80 Developer Platform 2.0 SDK for Symbian OS – For Personal Profile, which can be downloaded from Forum Nokia (www.forum.nokia.com/tools). In this context, using an emulator refers to setting the desired default emulator device. See chapter 5.1 for more information on setting the default emulator device.

The stand-alone installation of Nokia Developer's Suite 3.0 for J2ME™ does not include tools for editing, compiling or debugging the MIDlet and personal profile classes; you must use third-party tools and a Java™ SDK for these purposes. The other option is to install Nokia Developer's Suite 3.0 for J2ME™ integrated with an IDE such as Borland JBuilder®, Sun™ Java Studio / NetBeans or Eclipse platform, which provide the tools for editing, compiling and debugging MIDlets and Personal Profile applications (see the Installation and Configuration Guide [7], chapters 17, 18 and 19). The SDK Plug-in, which is delivered with Eclipse integration, enables to run and debug MIDlets with MIDP SDKs and Personal Profile applications with SDKs supporting Personal Profile.

## 1.4 Document overview

This document contains the following chapters:

- **Chapter 1** introduces Nokia Developer's Suite 3.0 for J2ME™
- **Chapter 2** takes a glance at the stand-alone window of the Nokia Developer's Suite 3.0 for J2ME™ toolkit
- **Chapter 3** familiarises you with the dynamic tool-loading concept
- **Chapter 4** explains how to change projects in the stand-alone version of Nokia Developer's Suite for J2ME™
- In **Chapter 5** you learn about SDK/emulator configuration
- From **Chapter 6** to **Chapter 15**, the use of each tool in the toolkit is described in detail
- **Chapter 16** shows how you can add new features to and update Nokia Developer's Suite for J2ME™

- **Chapter 17** provides information on using the Nokia Developer's Suite for J2ME™ tools with Borland® JBuilder®
- **Chapter 18** shows how to use Nokia Developer's Suite for J2ME™ tools with Sun™ ONE Studio 4
- **Chapter 19** shows how to use Nokia Developer's Suite for J2ME™ tools with Eclipse

## 2 Getting started

Launch the stand-alone program from the Start menu or run the shell script RunNDSforJ2ME.bat. When the program has been loaded you see the main window of the application (Figure 1).

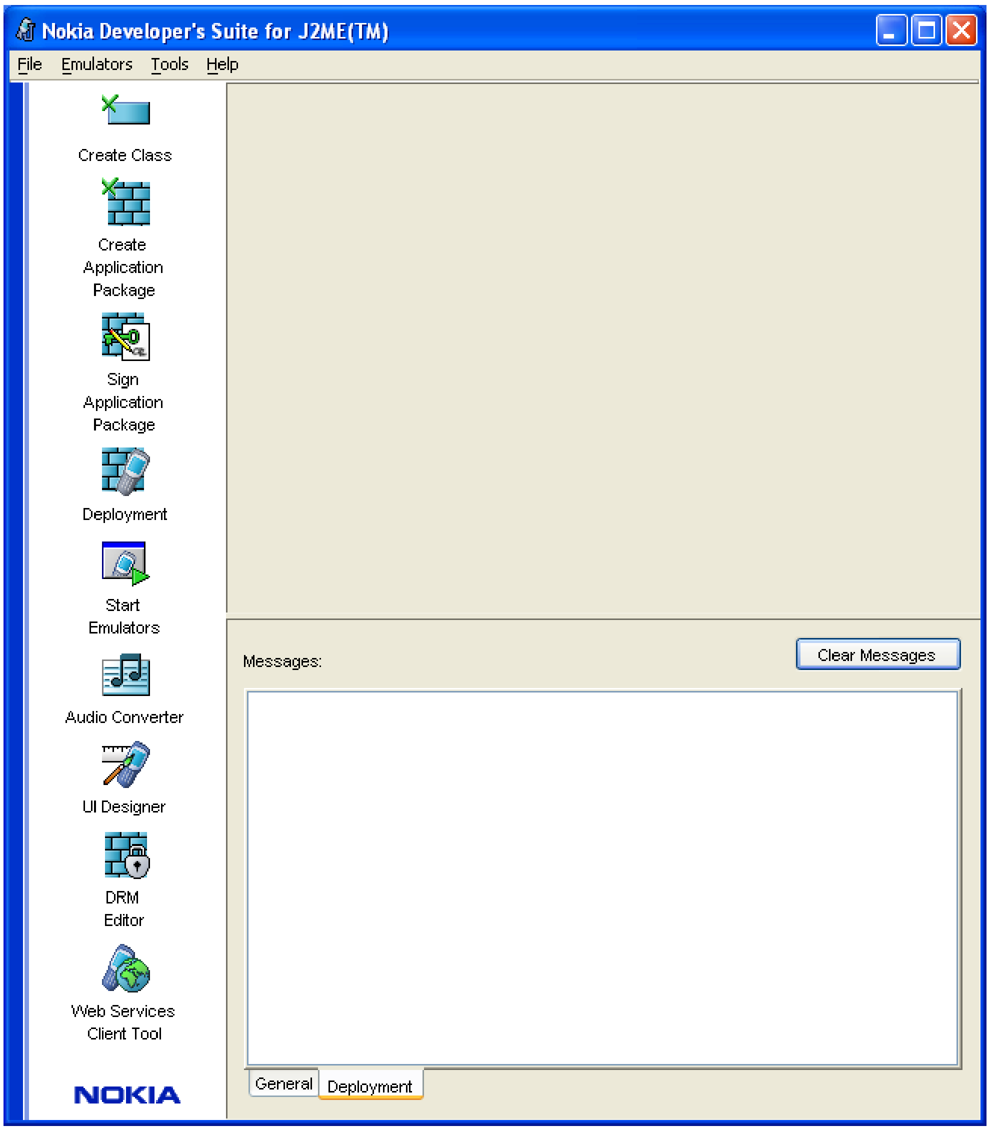


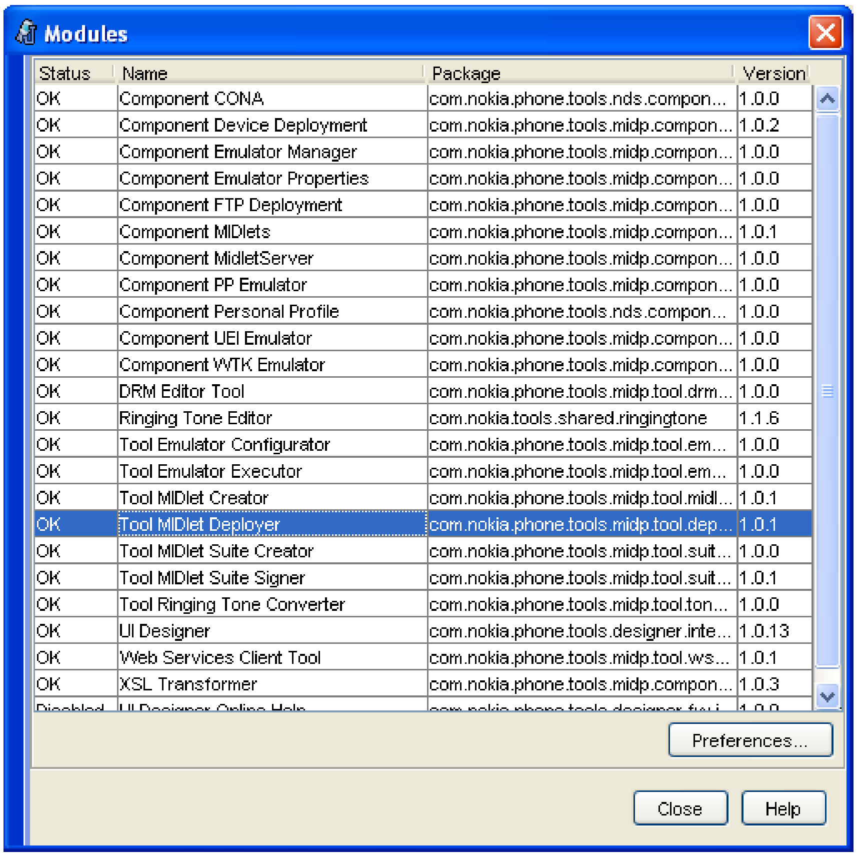
Figure 1: Stand-alone application's main window

# 3 Tool Management

When Nokia Developer’s Suite 3.0 for J2ME™ is loaded, it locates and dynamically loads modules from a specific directory. All Nokia Developer’s Suite 3.0 for J2ME™ tools on the stand-alone window’s left-hand pane and in the menus are loaded from these modules during start-up. Modules implement the actual functions of Nokia Developer’s Suite 3.0 for J2ME™. With Update Manager, you can add new modules or update old ones as additional modules come available. Typically module management happens behind the scenes without any input from the user. However, Nokia Developer’s Suite 3.0 for J2ME™ provides the user with the possibility to view the selection of modules and even affect their installation. You can safely skip this section, unless you want manual control over the loading of modules. But if you choose, you can access manual module management from the main menu’s *Help* | *Modules* item. This menu item opens the *Modules* dialog.

## 3.1 Modules Dialog

Modules dialog (Figure 2) shows all the modules found in the system with their current status.



| Status | Name | Package | Version |
|---|---|---|---|
| OK | Component CONA | com.nokia.phone.tools.nds.compon... | 1.0.0 |
| OK | Component Device Deployment | com.nokia.phone.tools.midp.compon... | 1.0.2 |
| OK | Component Emulator Manager | com.nokia.phone.tools.midp.compon... | 1.0.0 |
| OK | Component Emulator Properties | com.nokia.phone.tools.midp.compon... | 1.0.0 |
| OK | Component FTP Deployment | com.nokia.phone.tools.midp.compon... | 1.0.0 |
| OK | Component MIDlets | com.nokia.phone.tools.midp.compon... | 1.0.1 |
| OK | Component MidletServer | com.nokia.phone.tools.midp.compon... | 1.0.0 |
| OK | Component PP Emulator | com.nokia.phone.tools.midp.compon... | 1.0.0 |
| OK | Component Personal Profile | com.nokia.phone.tools.nds.compon... | 1.0.0 |
| OK | Component UEI Emulator | com.nokia.phone.tools.midp.compon... | 1.0.0 |
| OK | Component WTK Emulator | com.nokia.phone.tools.midp.compon... | 1.0.0 |
| OK | DRM Editor Tool | com.nokia.phone.tools.midp.tool.drm... | 1.0.0 |
| OK | Ringing Tone Editor | com.nokia.tools.shared.ringingtone | 1.1.6 |
| OK | Tool Emulator Configurator | com.nokia.phone.tools.midp.tool.em... | 1.0.0 |
| OK | Tool Emulator Executor | com.nokia.phone.tools.midp.tool.em... | 1.0.0 |
| OK | Tool MIDlet Creator | com.nokia.phone.tools.midp.tool.midl... | 1.0.1 |
| OK | Tool MIDlet Deployer | com.nokia.phone.tools.midp.tool.dep... | 1.0.1 |
| OK | Tool MIDlet Suite Creator | com.nokia.phone.tools.midp.tool.suit... | 1.0.0 |
| OK | Tool MIDlet Suite Signer | com.nokia.phone.tools.midp.tool.suit... | 1.0.1 |
| OK | Tool Ringing Tone Converter | com.nokia.phone.tools.midp.tool.ton... | 1.0.0 |
| OK | UI Designer | com.nokia.phone.tools.designer.inte... | 1.0.13 |
| OK | Web Services Client Tool | com.nokia.phone.tools.midp.tool.ws... | 1.0.1 |
| OK | XSL Transformer | com.nokia.phone.tools.midp.compon... | 1.0.3 |
| Disabled | UI Designer Online Help | com.nokia.phone.tools.designer.fw.i... | 1.0.0 |

Figure 2: Module status dialog

For each module the following information is displayed:

### Status

Reveals the current status of the module:

| | |
|---|---|
| OK | The module is loaded. |
| Disabled | The user has disabled the module. |
| Obsolete | A more recent version of the module was available. Hence, this version of the module was not used. |
| Error | The module could not be loaded. There are a number of possible reasons for this. For example, the module depends on another module, which is not available for some reason, such as the user has disabled it. |

**Name**

Internal name of the module. This is not necessarily the same as that shown, for example, in a menu.

**Package**

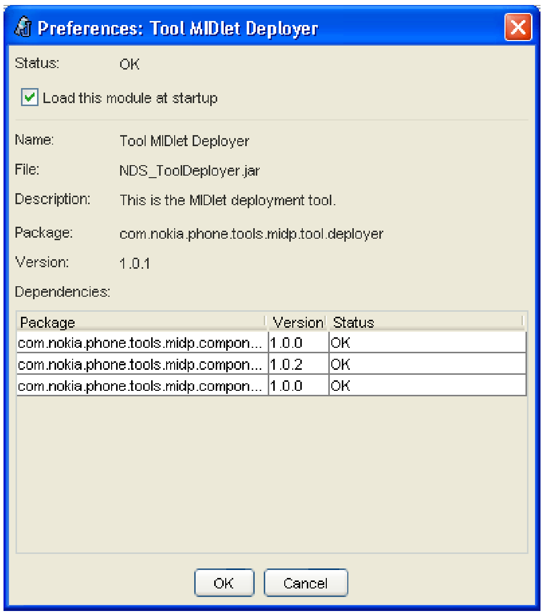
If two modules have the same package information, they are bound to be different versions of the same module.

**Version**

Version number of the module.

When launched from IDE, the table in the dialog contains an additional Toolbar column, where the user can select which tools are present in the Nokia Developer's Suite 3.0 for J2ME™ toolbar of IDE.

The table rows can be sorted by clicking the column headers. When a table row is selected, double-click the row or press **Preferences...** button to open the *Module Preferences* dialog (Figure 3).



Status: OK

✓ Load this module at startup

Name: Tool MIDlet Deployer

File: NDS_ToolDeployer.jar

Description: This is the MIDlet deployment tool.

Package: com.nokia.phone.tools.midp.tool.deployer

Version: 1.0.1

Dependencies:

| Package | Version | Status |
|---|---|---|
| com.nokia.phone.tools.midp.compon... | 1.0.0 | OK |
| com.nokia.phone.tools.midp.compon... | 1.0.2 | OK |
| com.nokia.phone.tools.midp.compon... | 1.0.0 | OK |

Figure 3: Module preferences dialog

## 3.2 Module Preferences Dialog

Module Preferences dialog contains further information about the selected module. In addition, users can enable/disable the module for the next time the application is launched. Some of the information is the same as in the Modules dialog but naturally there are additions as well:

**Status**

The status information is augmented with more details if available.

**File**

File name of Java archive file containing the module. The file is located in system's modules directory.

**Description**

Brief description of the module's functionality and its origins.

**Dependencies**

List of packages (modules) that are needed by this module. The contents of Status fields can help in resolving different module problems:

| | |
|---|---|
| OK | Module is loaded. |
| Not available | The module that corresponds to the needed package is either not installed or it could not be loaded (due to failure in its own dependencies, for example). |
| Superseded by *version* | A newer version of the needed package is found in the system. Since the modules are assumed to be backward compatible the newer versions are always used. However, nothing guarantees that the newer version can be successfully loaded. |
| Not resolved | One of the other dependencies could not be loaded, thus making it unnecessary to resolve dependencies any further. |

# 4 Preferences

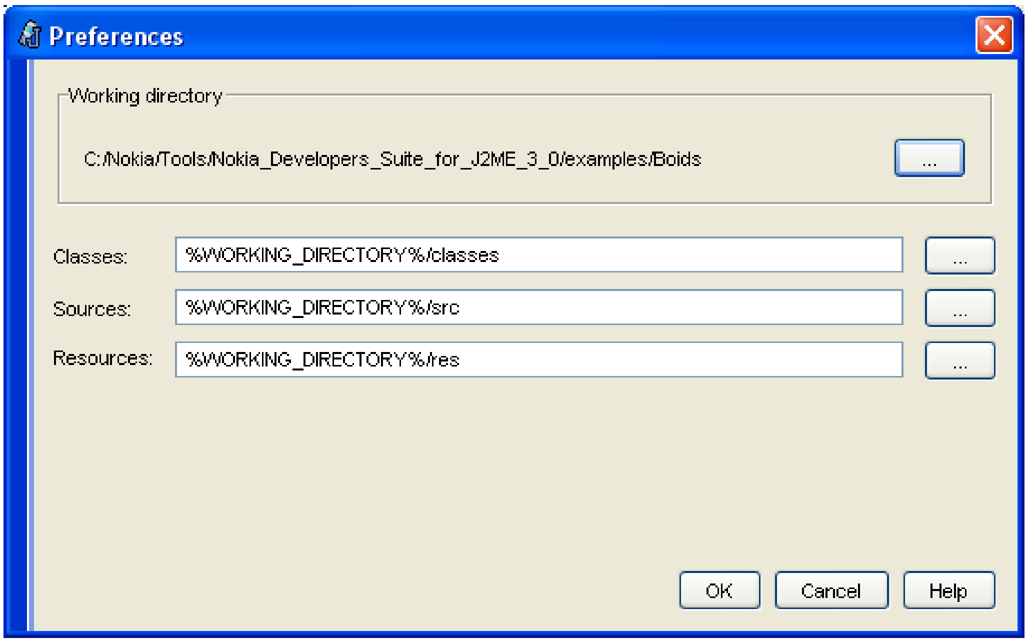
One of the first things you should do after installing the software is to set the preferences.

**Note : Preferences are only available when running Nokia Developer's Suite 3.0 for J2ME™ as a stand-alone application.**

If you are using the product integrated with an IDE, you can skip this section. Preferences control the "project" currently in use with Nokia Developer's Suite 3.0 for J2ME™. The stand-alone version of Nokia Developer's Suite 3.0 for J2ME™ does not maintain any project-related information as such, it only assumes that all files found from the project directories are project files. During the installation of Nokia Developer's Suite 3.0 for J2ME™ example application (Boids) directories are set as the default project directories. To create your own MIDP and Personal Profile applications you should change the directories. Use the Preferences dialog box to configure the following project directories:

- **User class directory (Classes) -** the directory where compiled project class files are located
- **Source directory (Sources) -** the directory where source files are located
- **Resource directory (Resources) -** the directory where resource files are located

To open the dialog (Figure 4), select *File* | *Preferences*.



Working directory: C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME_3_0/examples/Boids

Classes: %WORKING_DIRECTORY%/classes

Sources: %WORKING_DIRECTORY%/src

Resources: %WORKING_DIRECTORY%/res

Figure 4: Preferences dialog

An application package (suite) contains classes used by the MIDlets or Personal Profile applications and resources, such as picture files. The default directory structure for a MIDP or Personal Profile application project in the stand-alone version of Nokia Developer's Suite 3.0 for J2ME™ is as follows:

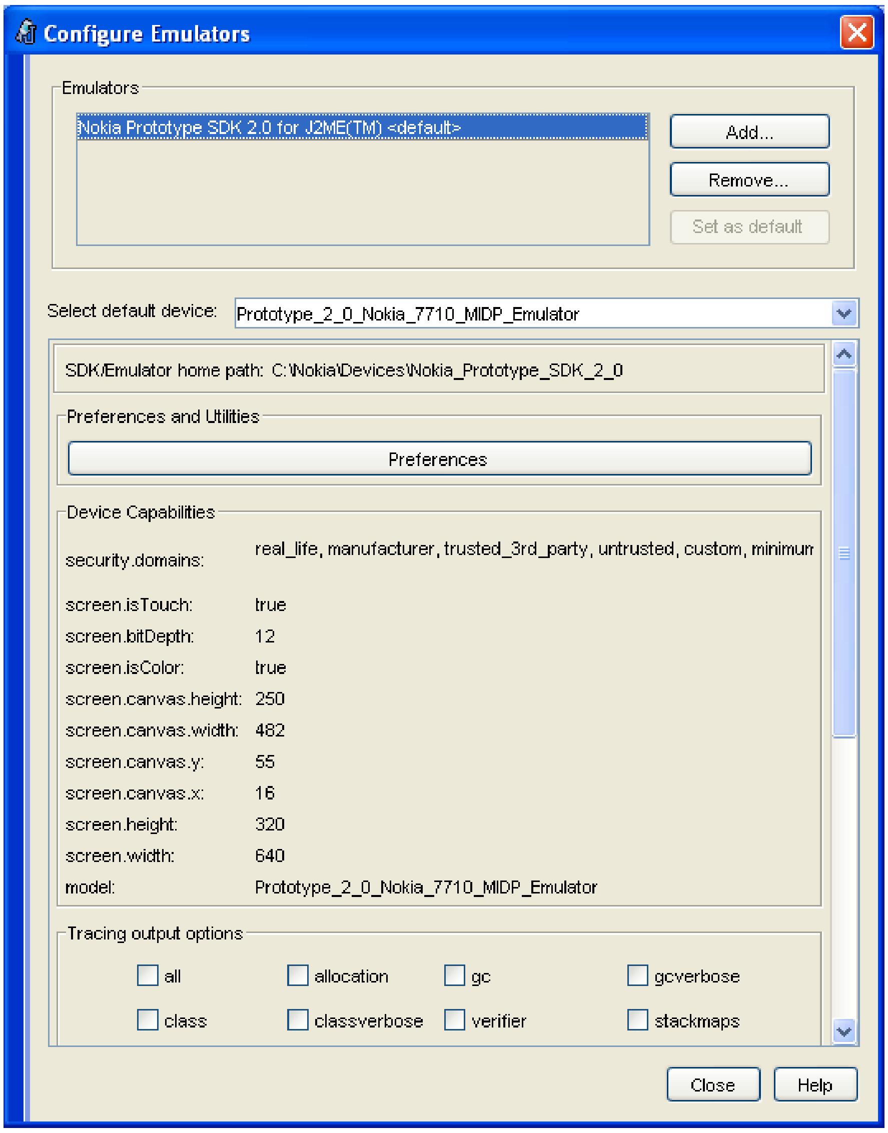
| | |
|---|---|
| \[root] | Working directory. Application Packages created by the tool will be saved here by default. If the class, source and resource directories are manually set to exist under different root directories, the working directory is considered to be the one above the class directory. |
| \classes | Repository for compiled classes. |
| \src | Repository for source code. |
| \res | Repository for resource files (resources may also be located under \root\classes). |

## 4.1 Setting the working directory

When you browse for a working directory with the *Working directory* browse button, the program will automatically set the class directory, source directory and resource directory to their default values according to the table above.

# 5 SDK/Emulator Configuration

During Nokia Developer's Suite 3.0 for J2ME™ installation one Nokia SDK is installed and configured automatically for use; Nokia Prototype SDK 2.0 for J2ME.This SDK contains four emulators (Prototype 2.0 S40 MIDP Emulator, Prototype 2.0 S60 MIDP Emulator, Prototype 2.0 S80 MIDP Emulator and Prototype 2.0 Nokia 7710 MIDP Emulator). You can configure the properties of the emulators with the *Configure Emulators* tool (Figure 5). The tool is also used for adding or removing new emulators manually to Nokia Developer's Suite 3.0 for J2ME™.



Nokia Prototype SDK 2.0 for J2ME(TM) <default>

Select default device: Prototype_2_0_Nokia_7710_MIDP_Emulator

SDK/Emulator home path: C:\Nokia\Devices\Nokia_Prototype_SDK_2_0

Device Capabilities

| | |
|---|---|
| security.domains: | real_life, manufacturer, trusted_3rd_party, untrusted, custom, minimum |
| screen.isTouch: | true |
| screen.bitDepth: | 12 |
| screen.isColor: | true |
| screen.canvas.height: | 250 |
| screen.canvas.width: | 482 |
| screen.canvas.y: | 55 |
| screen.canvas.x: | 16 |
| screen.height: | 320 |
| screen.width: | 640 |
| model: | Prototype_2_0_Nokia_7710_MIDP_Emulator |

Figure 5: Configure Emulators dialog

Use the *Update Manager* (see Chapter 16) to discover and download new emulator releases. When you have downloaded and installed a new emulator, it can be used and configured in the Nokia Developer's Suite 3.0 for J2ME™ *Configure Emulators* tool like any other SDK/emulator.

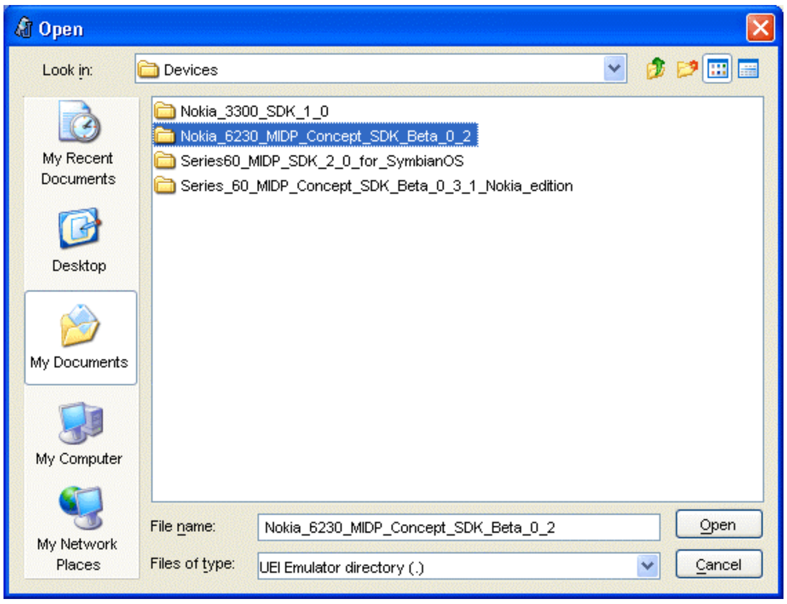
The *Configure Emulators* tool is started from the menu:

*Emulators* | *Configure Emulators...*

Use the *Configure Emulators* dialog box to do the following:

- Choose the default emulator
- Configure emulator properties
- Add emulators
- Remove emulators

You can add an emulator with **Add**. In the file dialog (Figure 6), select a valid emulator root directory or configuration file to add to Nokia Developer's Suite 3.0 for J2ME™.



Nokia_3300_SDK_1_0
Nokia_6230_MIDP_Concept_SDK_Beta_0_2
Series60_MIDP_SDK_2_0_for_SymbianOS
Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition

File name: Nokia_6230_MIDP_Concept_SDK_Beta_0_2
Files of type: UEI Emulator directory (.)

Figure 6: Browsing for an emulator root directory

When an emulator has been added to Nokia Developer's Suite 3.0 for J2ME™, the SDK/emulator can be selected from the list titled *Emulators* (see Figure 5). The properties of the selected emulator device are displayed in the tool. Each emulator may contain one or more devices. If there is more than one device you can configure each one individually. Change currently selected device from the *Select default device* combo box. If this combo box is not displayed, the emulator contains just one device.

**Note : Some Nokia SDKs can have options like traces, debugging and heap size configured via the SDKs' preferences dialog. For these SDKs you should use the SDKs' preferences dialog and not the Configure Emulator window in Nokia Developer's Suite window.**

An emulator can be removed with **Remove**. Note that this only removes the emulator from Nokia Developer's Suite 3.0 for J2ME™ tool set. The actual uninstallation of the emulator must be done with the emulator's own uninstallation procedure. If the SDK/emulator has not been uninstalled it can be reloaded to Nokia Developer's Suite 3.0 for J2ME™ with **Add...** button.

## 5.1 Default emulator device

The default emulator device, that is, the device that provides the MIDP/PP, CLDC/CDC and optional APIs and is used for pre-verifying the classes, is the current device that is selected for the default SDK marked with the tag <*default*>. If you want to change the default device, select the preferred SDK/emulator and set it as the default SDK/Emulator with the **Set as default** button. If the SDK contains only one device, it is selected automatically as the default emulator device. If there is more than one device in the SDK, select the default emulator device from the combo box labelled *Select default device*.

# 6 Create Class

Use the *Create Class* tool to create class source files for your application. Before running this tool the default emulator must have been set with the *Configure Emulators* tool. *Create Class* tool locates the available MIDP/PP and CLDC/CDC API classes from the default emulator device.

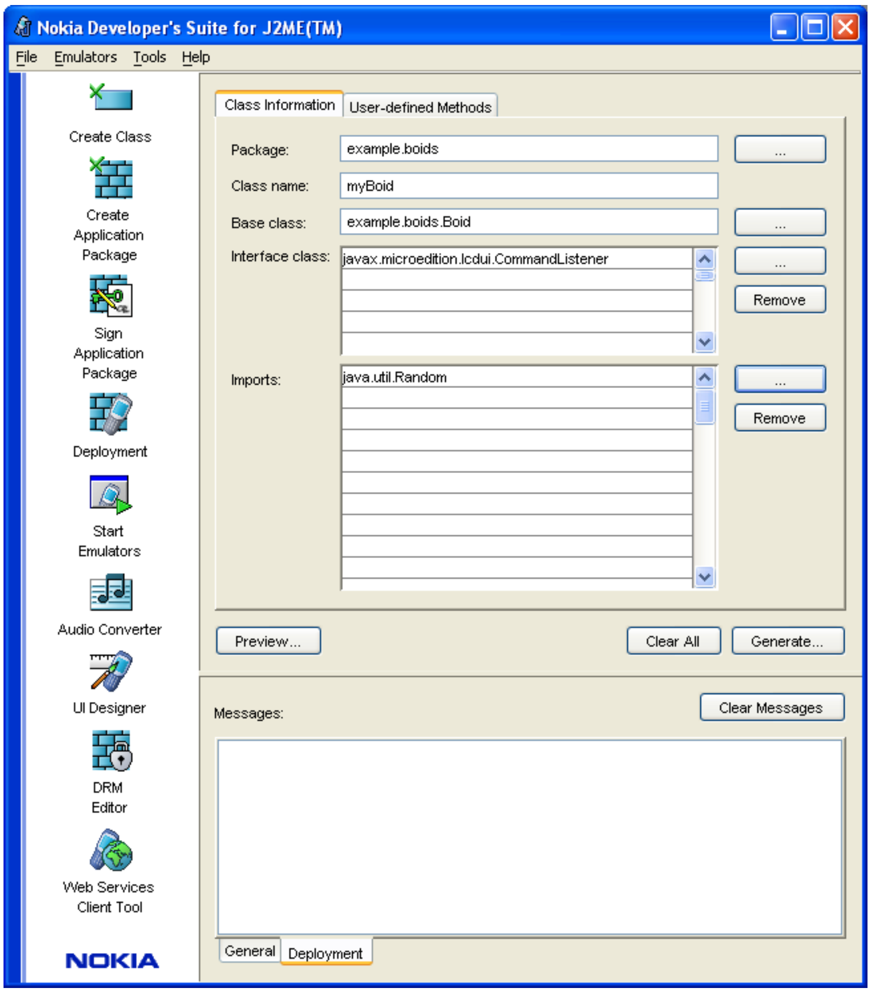
The tool is started from the main menu. To start the class creation tool select:

*File* | *New class*

or press the *Create Class* button in the main window of the stand-alone application.

## 6.1 Class Information tab

In the Class Information tab, enter basic information about the class (Figure 7).



Package: example.boids

Class name: myBoid

Base class: example.boids.Boid

Interface class: javax.microedition.lcdui.CommandListener

Imports: java.util.Random

Messages:

Figure 7: Class Information tab

Firstly, enter the package name of the class in the *Package* text field or browse for it with the browse button. Then enter the name for the class in the *Class name* text field. In the *Base class* text field, enter the base class that will be extended by your class. If you are going to create e.g. a MIDlet main class, then the base class must be **javax.microedition.midlet.MIDlet** or its subclass. The base class can be typed in the text field or browsed for with the browse button on the right side of the text field. When browsing for the base class, the browser displays all found API classes from the default SDK, as well as any classes found in the user class path (Figure 8). User class path is defined in the *File* | *Preferences* dialog (see Chapter 4) or by the IDE integration.

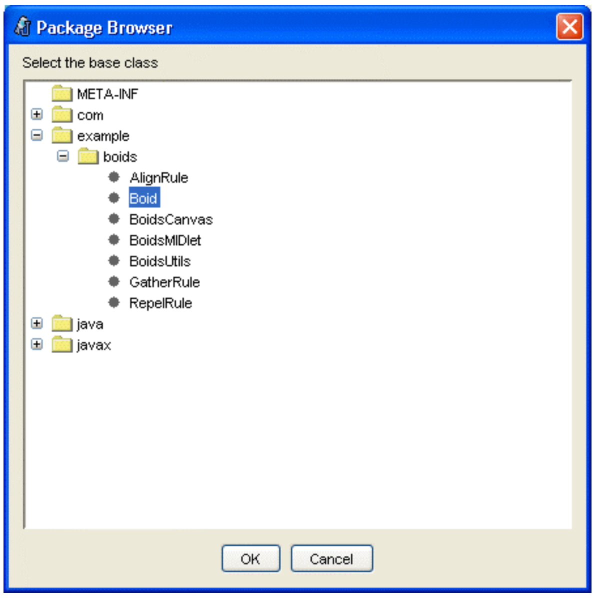


Figure 8: Browsing for a base class

In the *Interface* class table, the interface classes that are to be implemented by your class are displayed. For example, if you make a class for your MIDlet application that handles commands from the device, you should then implement **javax.microedition.lcdui.CommandListener**. Interfaces can be browsed for with the browse button on the right side of the panel (Figure 7). When browsing for interface classes, you can select multiple nodes. The browser displays all interface classes found from the default SDK, as well as the project's previously-created and compiled interface classes found in the user class path (Figure 9). The selected classes in the *Interface* class table can be removed with the *Remove* button on the right side of the table.

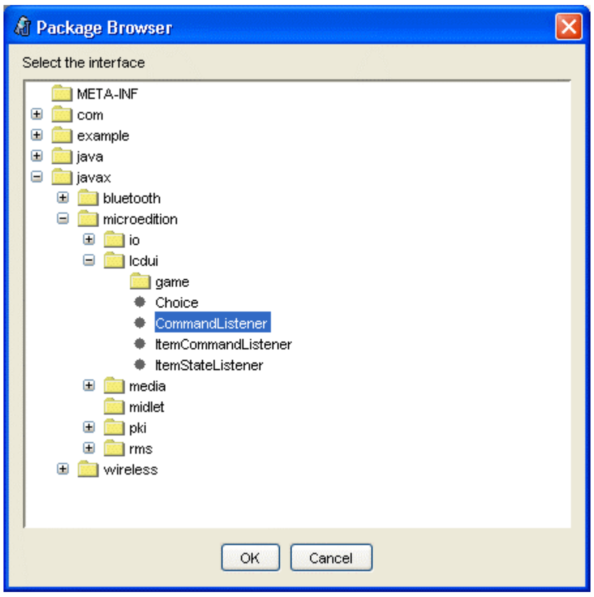
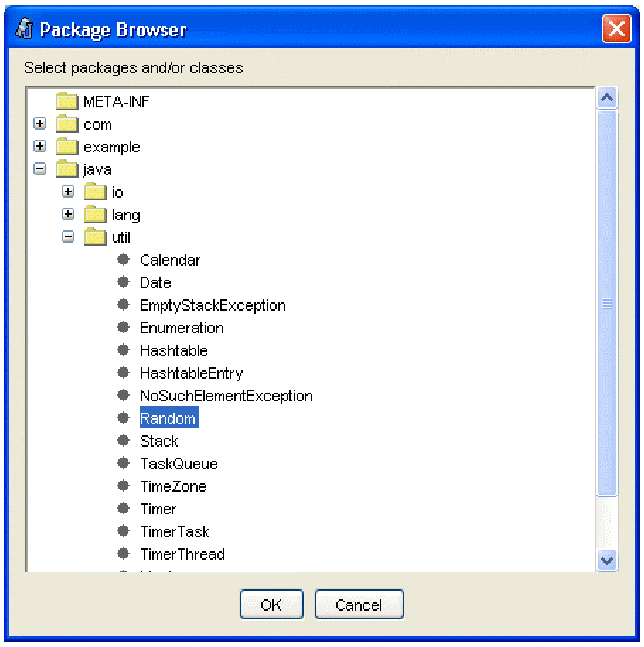


Figure 9: Interface class browsing

If you need to use other classes or packages with your class, add them to the *Imports* table. The browse button on the right side of the table will open a dialog that displays all API classes and packages, as well as the project's own previously created and compiled classes or packages (Figure 10). When browsing for imports, more than one node can be selected. The selected classes and packages in the *Imports* table can be removed with the *Remove* button on the right side of the table.



META-INF
com
example
java
io
lang
util
Calendar
Date
EmptyStackException
Enumeration
Hashtable
HashtableEntry
NoSuchElementException
Random
Stack
TaskQueue
TimeZone
Timer
TimerTask
TimerThread

Figure 10: Import selection

## 6.2 User-defined Methods tab

Under the *User-defined Methods* tab, you can create your own methods using a graphical user interface. Firstly, select a modifier for the method by clicking *_modifier_* tag in the method skeleton. A popup list of the selectable modifiers is displayed (Figure 11). When you have selected the modifier, it is displayed in blue in place of the *_modifier_* tag.

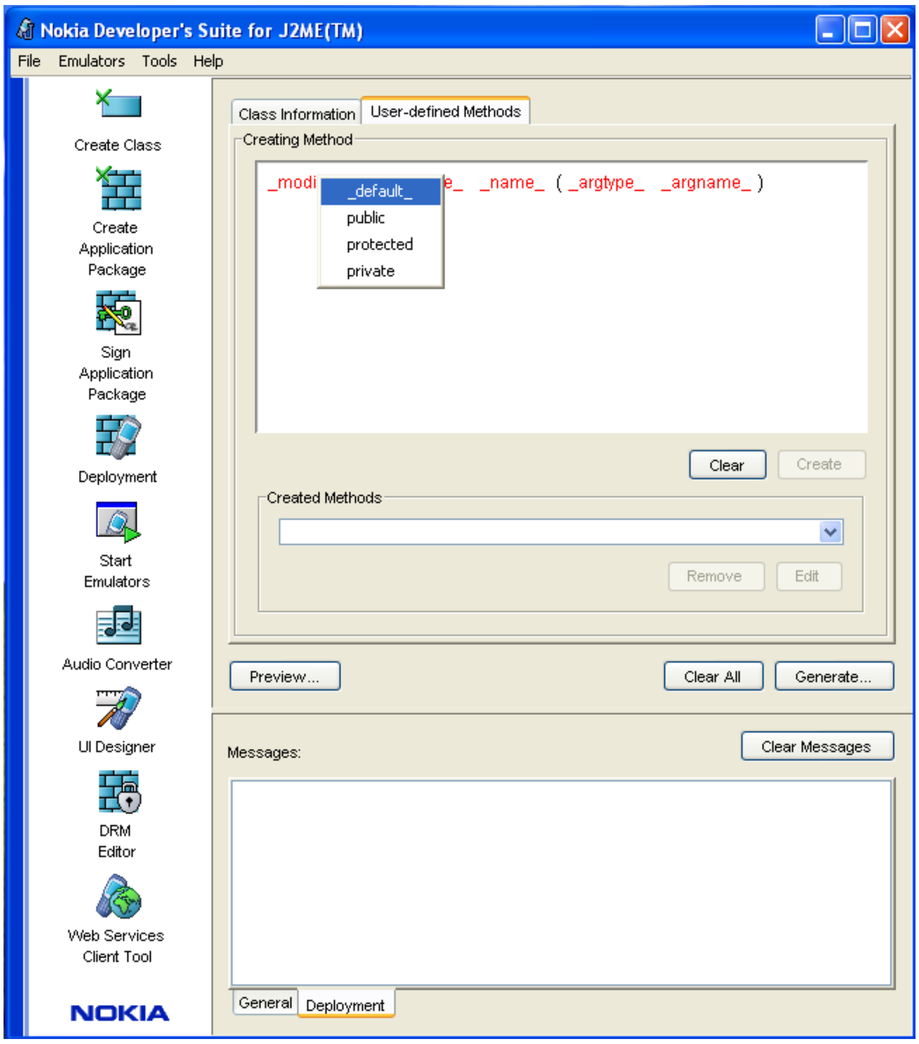


Figure 11: Setting method modifier

Set return type for the method in the same manner. When you click *_returntype_* tag, a list of selectable types is displayed. If the type you want is not displayed in the list, then select *Others...* at the end of the list. This selection gives you a browser tool similar to the one used while browsing for a base class. When you have selected a return type for your method, the return type is displayed in blue in place of the *_returntype_* tag.

When the return type is defined, name your method by clicking on *_name_* tag. A text field where you can type the name is displayed. Accept method name with Enter. The name you give is displayed in blue in place of the *_name_* tag.

When the modifier, the return type and the name of the method are defined, **Create** button is enabled. If you do not need to define arguments for your method, create it now. To define an argument for the method, set the argument type by clicking *_argtype_* tag in the method skeleton. A popup list with all the selectable types is displayed. If the type you want is not displayed in the list, select *Others...* at the end of the list. This is done in the same way as when selecting the return type of the method. The selected type is displayed in blue in place of the *_argtype_* tag. Clicking *_argname_* tag in the method skeleton sets the name for the argument. A text field where you can define the name for this argument is displayed. Accept argument name with Enter. The given argument name is displayed in blue in place of *_argname_* tag. When you have set the name for the argument, *_argtype_* and *_argname_* tags for the next argument are displayed.

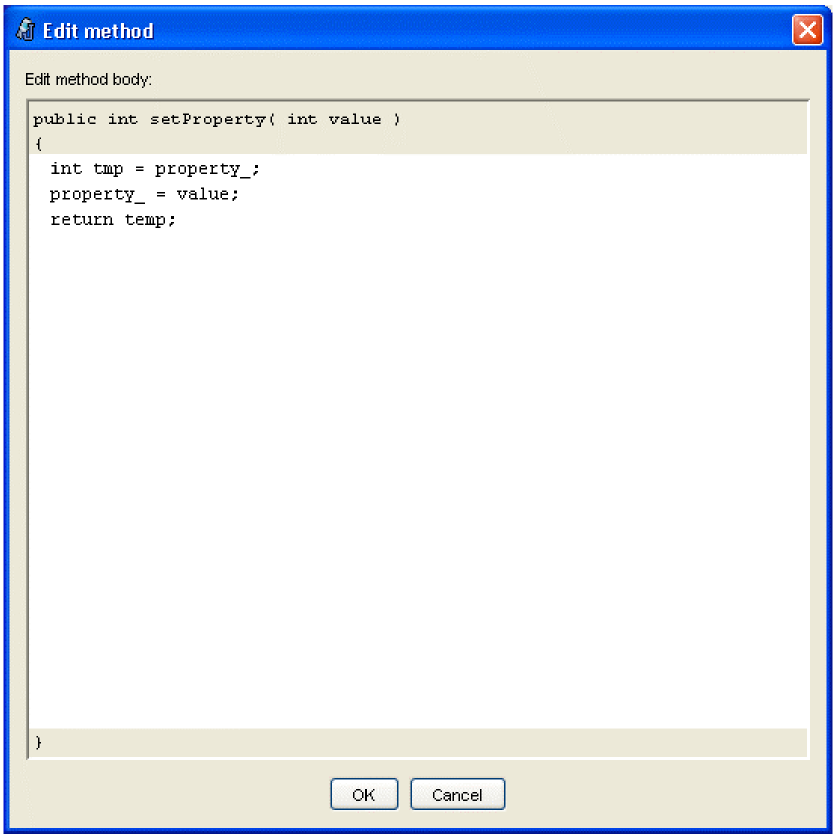
You can change all the previously set parts of the method skeleton by clicking the part and by selecting or writing a new value for that part. If the **Create** button is disabled, one or more parts have invalid values. Giving them valid values will enable the button.

When your method definition is ready, create the method with **Create** button. The arguments in blue are used. Remember that if there are two arguments with the same name, **Create** button will remain disabled. If you want to reset the method before it has been created, do so with **Clear** button.

The created method is displayed in the *Created methods* list. To remove a created method from the list, select the method and press **Remove** button.

To edit the body of the created method, select the method from the *Created Methods* list and click on **Edit** button. This opens a window displaying the method contents (Figure 12). In the edit window, the method skeleton generated by the tool is visible. Now you can add your own contribution to the method. When you press **OK** button, the modifications you have made are saved. **Cancel** button prevents the modifications from being saved. Note that you cannot modify the modifier, return type, method name or the arguments in this window. Only the content of the method body may be modified.

When enough information has been entered in the *Class information* tab, **Preview** and **Generate** buttons are enabled below the tabbed pane. With the **Preview** button, you can instantly preview the class source code. **Generate** button generates and saves the generated source code. The default location for saving the source code is under the source path defined in *Preferences* (see Chapter 4) or by the IDE integration.



```
public int setProperty( int value )
{
  int tmp = property_;
  property_ = value;
  return temp;
}
```

Figure 12: Editing a user-defined method

# 7 Create Application Package

A MIDP Application Package (MIDlet Suite) consists of two files, the JAR package that contains all the classes and other files needed in your MIDP Application, plus the Java Application Descriptor (JAD) that describes the MIDlet and its JAR package.

To use the classes created with the *Create Class* tool in creating a MIDP Application Package, you must compile them. Nokia Developer's Suite 3.0 for J2ME™ does not provide tools for compiling classes. Instead, you must use Java™ 2 SDK from the command line or from within a third-party Java IDE together with an emulator class library to compile the classes. The following command compiles an application using the command line tools of JDK 1.4.1_02 in the working directory of the project and the Nokia Prototype 2.0 SDK class libraries.

```
javac -target 1.1 –d classes –classpath classes –bootclasspath
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\cldcapi11.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\bluetooth.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\fca.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\location.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\m3g.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\mma.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\nokiaui.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\pim.zip;
C:\Nokia\Devices\Nokia_Prototype_2_0\lib\ext\wma.zip src\com\mycompany\myapp\*.java
```

For more information on compiling classes, see J2SE™ documentation [1]. The above command only works with a Nokia Prototype SDK that has been installed in its default installation directory. It is important to remember to add all mobile API classes supported by the SDK to the boot class path when compiling. Once the project's classes have been compiled, create the Application Package. Use the *Create Application Package* tool to do this.

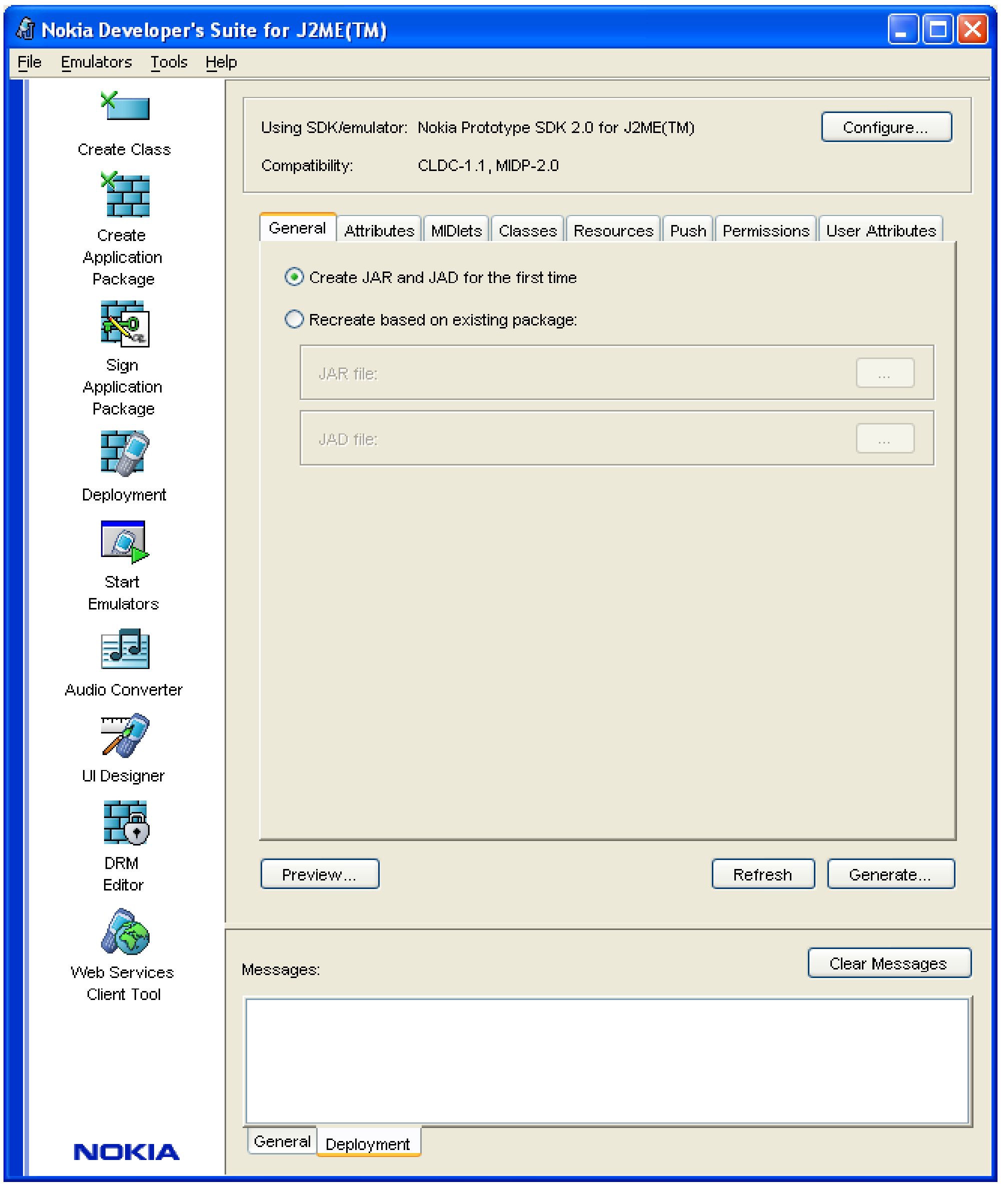
The Create Application Package tool also provides a way to create Connected Device Configuration (CDC) – Personal Profile (PP) / Foundation Profile (FP) based application packages. Depending on your SDK/emulator selection, you can create either MIDP or Personal Profile application packages.

The tool is started from the main menu. To start the tool, select *File* | *New Application Package* or click the *Create Application Package* button in the main window of the stand-alone application.

## 7.1 MIDP Application Package

### 7.1.1 General options

Select a MIDP SDK/emulator to start creating a MIDP Application Package. The default emulator used for loading and pre-verifying classes is displayed on the *General* tab. Select whether you want to create a new pair of JAR and JAD files or recreate JAR and JAD files from the contents of an existing package (Figure 13).



Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)

Compatibility: CLDC-1.1, MIDP-2.0

(●) Create JAR and JAD for the first time

( ) Recreate based on existing package:

Figure 13: General options for creating an Application Package

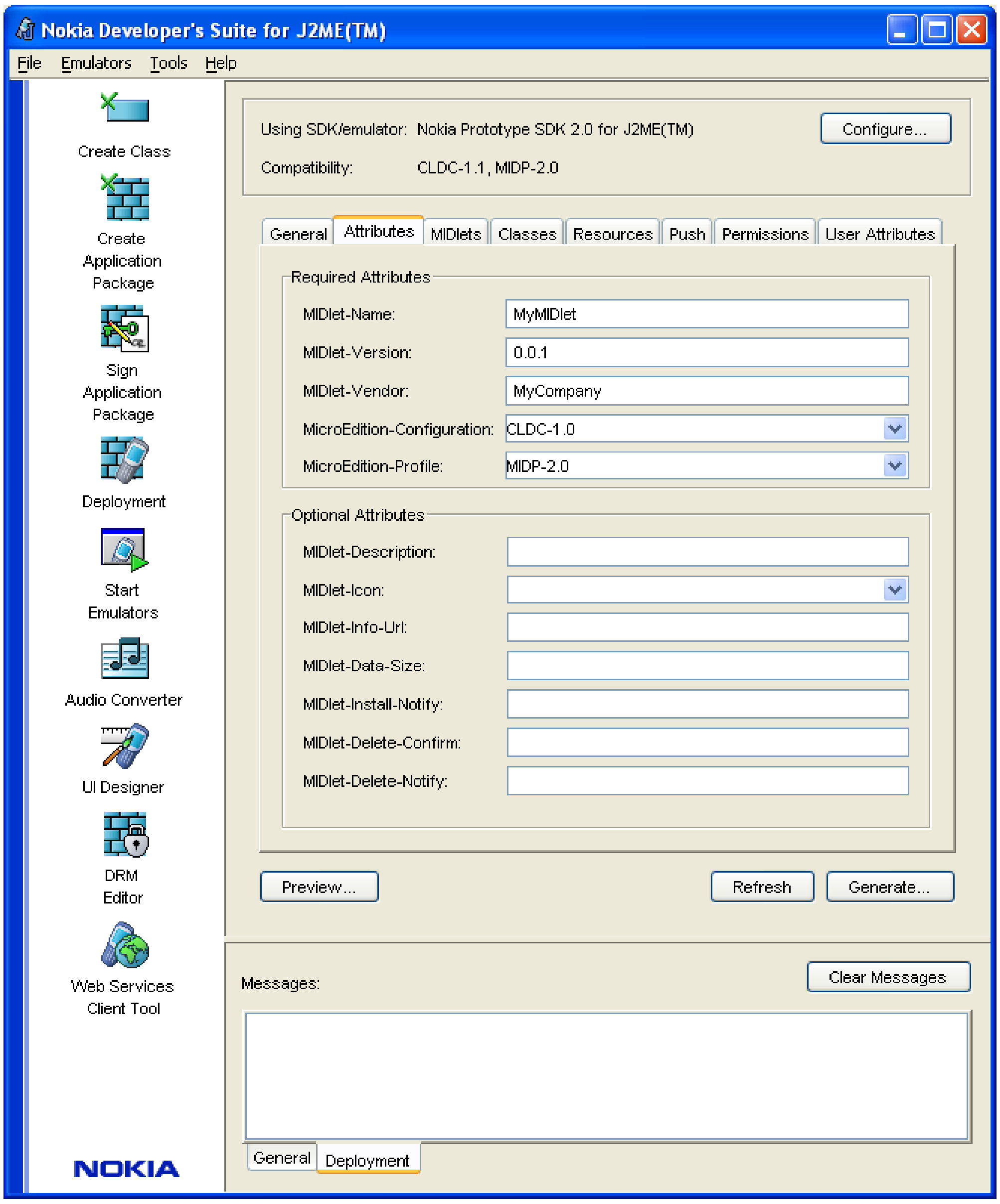
The default emulator device that is used for pre-verifying MIDlet Suite classes and for creating the package is displayed on this tab, with information about the MIDP version it supports. The number of tabs visible depends on whether the emulator includes MIDP 2.0 or MIDP 1.0. If you select a MIDP 2.0 SDK, Nokia Developer's Suite 3.0 for J2ME™ will create a MIDP 2.0 compatible Application Package by default. A default SDK/Emulator that is only MIDP 1.0 capable allows Nokia Developer's Suite 3.0 for J2ME™ to display options that are relevant to that profile only, and also create a MIDP 1.0 compatible Application Package. If you wish to change the default emulator, click **Configure**.

If you want to create a new application package from scratch, select **Create JAR and JAD file for the first time** option. The classes found in the user class path will be used in creating the MIDlet Suite.

If you want to modify previously created or otherwise already existing application package, select the **Recreate based on existing package** radio button. When recreating MIDP 2.0 MIDlets, you should use a MIDP 2.0 compatible emulator. Firstly, select the MIDlet Suite that you want to recreate. The MIDlet Suite is selected by browsing with the browse buttons inside the JAR and JAD file boxes.

### 7.1.2 MIDlet attributes

The required and optional MIDlet attributes are defined on the *MIDlet Attributes* tab (Figure 14). All the required attributes must be set. The required attributes are assigned default values when the tool is opened. The current default emulator assigns the *MicroEdition-Profile* and *MicroEdition-Configuration* attributes. If you wish to create a package for some other profile or configuration version, you can edit the profile and configuration using the drop-down menus.


Nokia Developer's Suite for J2ME(TM)
File Emulators Tools Help
Create Class
Create Application Package
Sign Application Package
Deployment
Start Emulators
Audio Converter
UI Designer
DRM Editor
Web Services Client Tool
NOKIA
Configure...
General Attributes MIDlets Classes Resources Push Permissions User Attributes
Preview... Refresh Generate...
Clear Messages
General Deployment

Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)

Compatibility: CLDC-1.1, MIDP-2.0

Required Attributes

| | |
|---|---|
| MIDlet-Name: | MyMIDlet |
| MIDlet-Version: | 0.0.1 |
| MIDlet-Vendor: | MyCompany |
| MicroEdition-Configuration: | CLDC-1.0 |
| MicroEdition-Profile: | MIDP-2.0 |

Optional Attributes

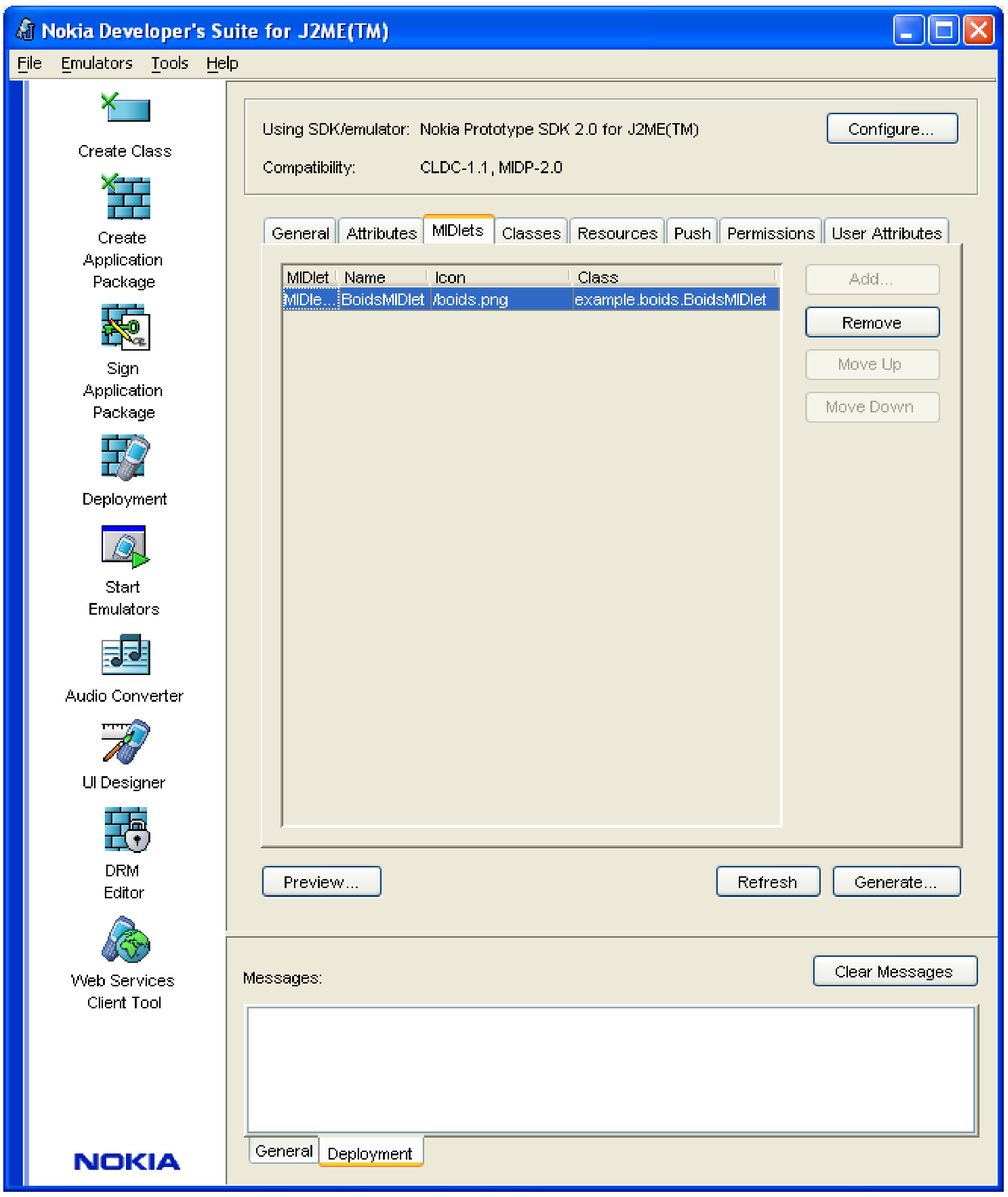
| | |
|---|---|
| MIDlet-Description: | |
| MIDlet-Icon: | |
| MIDlet-Info-Url: | |
| MIDlet-Data-Size: | |
| MIDlet-Install-Notify: | |
| MIDlet-Delete-Confirm: | |
| MIDlet-Delete-Notify: | |

Messages:

Figure 14: MIDlet Attributes

### 7.1.3 MIDlets in the Application Package

*MIDlets* tab displays all the MIDlets in your project in a table (Figure 15). In the *Name* column of the *MIDlets* table, you can change the MIDlet's name. The default name is the MIDlet class name. In the *Icon* column of the *MIDlets* table, you can select an icon for the MIDlet. This icon is displayed in the MIDlet selection screen of the device. The selectable icons are jpg, png and gif files found in the project's class or resource directories. The *Class* column displays the actual class name of the MIDlet. This cannot be changed.


Nokia Developer's Suite for J2ME(TM)
File Emulators Tools Help
Create Class
Create Application Package
Sign Application Package
Deployment
Start Emulators
Audio Converter
UI Designer
DRM Editor
Web Services Client Tool
NOKIA
Configure...
General Attributes MIDlets Classes Resources Push Permissions User Attributes
Add... Remove Move Up Move Down
Preview... Refresh Generate...
Messages:
Clear Messages
General Deployment

Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)

Compatibility: CLDC-1.1, MIDP-2.0

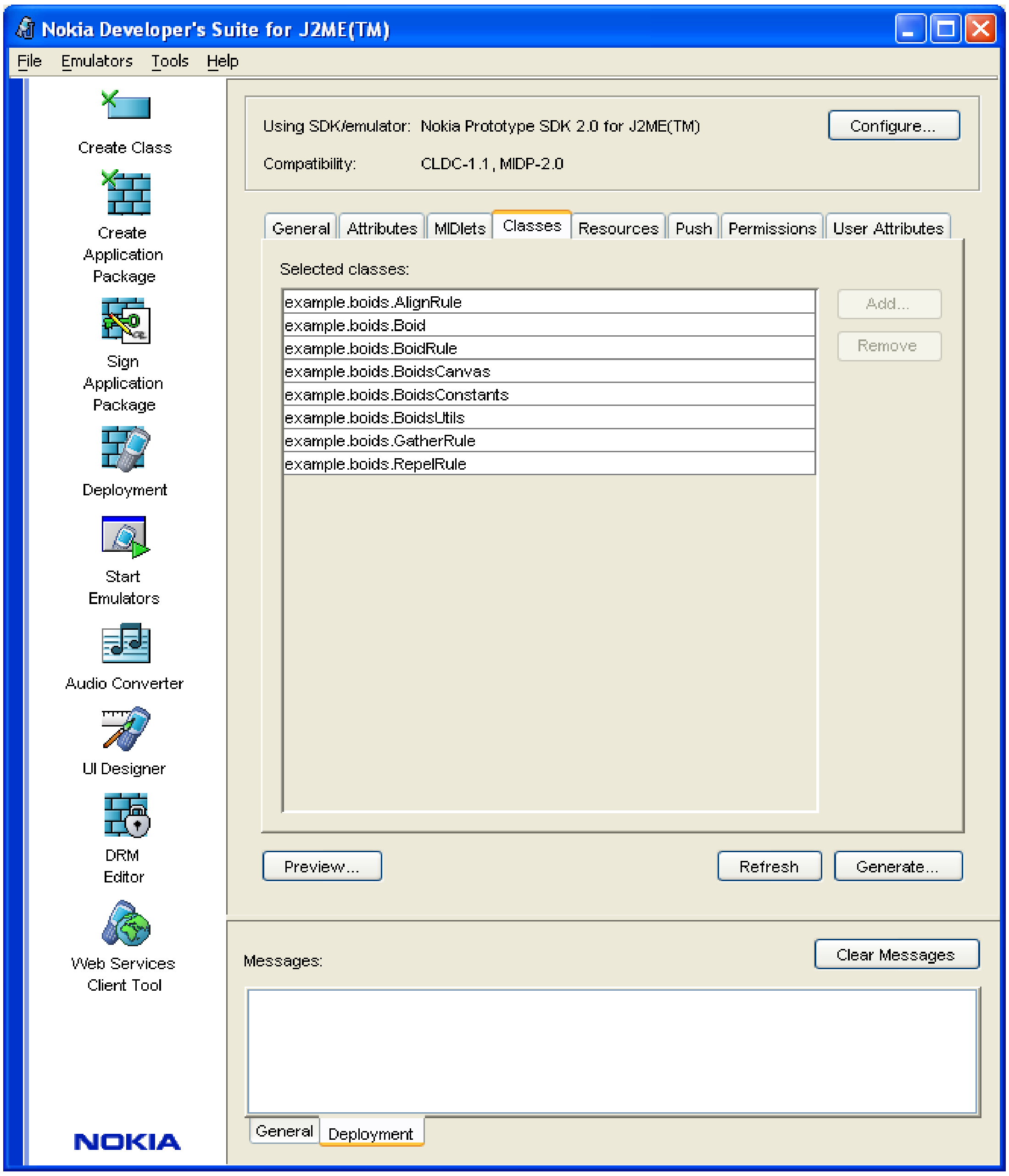
| MIDlet | Name | Icon | Class |
|---|---|---|---|
| MIDle... | BoidsMIDlet | /boids.png | example.boids.BoidsMIDlet |

Figure 15: MIDlets

To remove a MIDlet from your application, select a row in the MIDlets table and then press **Remove**. Select **Add...** to add a MIDlet to the table. **Move up** and **Move down** move the selected row up and down in the table. This affects the MIDlet-n attribute of the MIDlet in the manifest of the MIDlet Suite's JAR package.

### 7.1.4 Other classes and resources in the Application Package

If you have other classes in the project in addition to **javax.microedition.midlet.MIDlet** classes, these additional classes are displayed in a table on the *Classes* tab (Figure 16). To remove classes from your application, select rows in the table and then click **Remove**. To add classes to the table, click **Add...**.



Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)

Compatibility: CLDC-1.1, MIDP-2.0

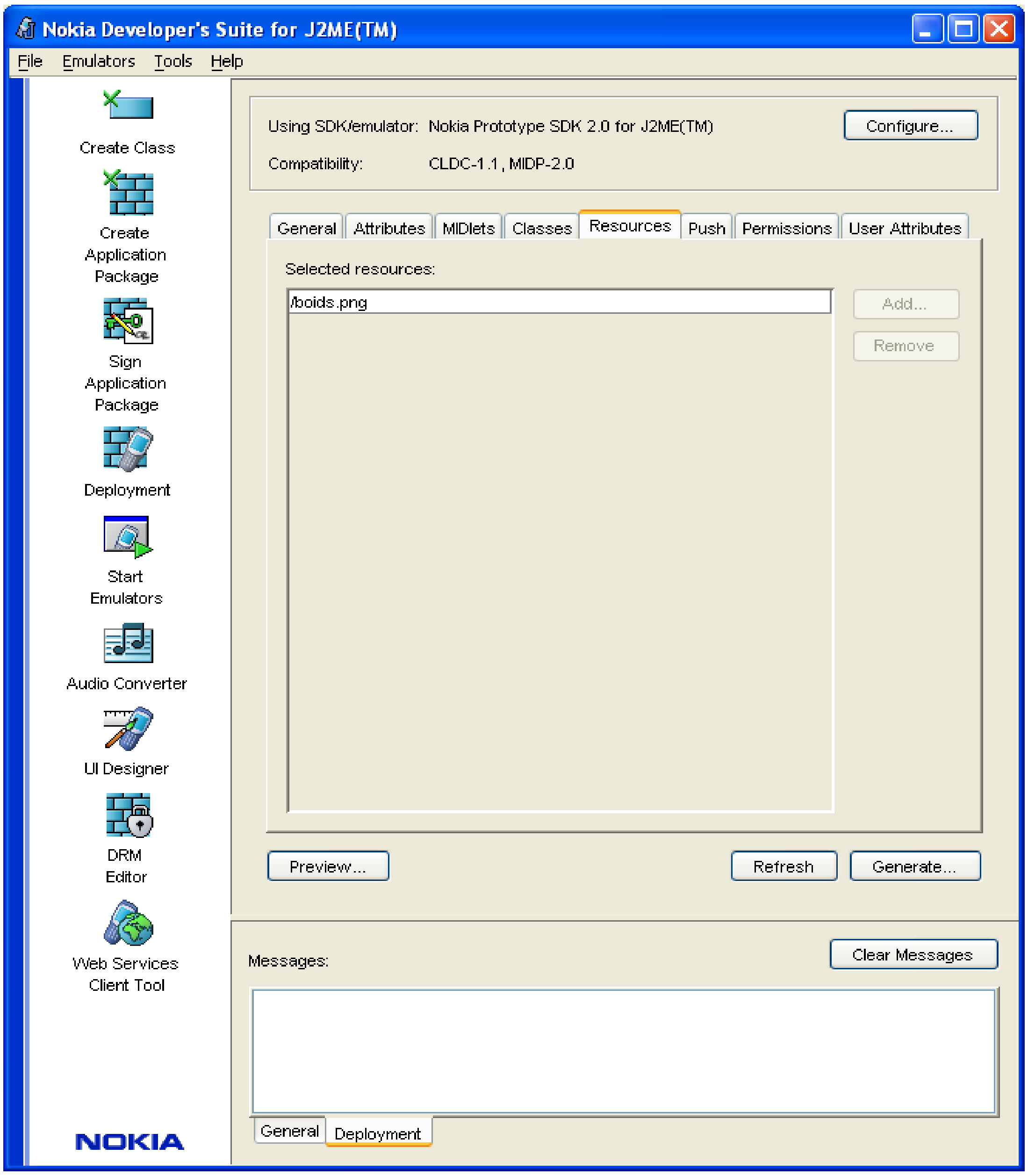
Selected classes:

example.boids.AlignRule
example.boids.Boid
example.boids.BoidRule
example.boids.BoidsCanvas
example.boids.BoidsConstants
example.boids.BoidsUtils
example.boids.GatherRule
example.boids.RepelRule

Messages:

Figure 16: Other classes

If you have resources in your project, for example, picture or text files to be used in your application, they are displayed in a table on the *Resources* tab (Figure 17). To remove resource files from your application package, select rows in the table and then click **Remove**. Click **Add...** to add resources found in the class or resource path to the table.



Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)

Compatibility: CLDC-1.1, MIDP-2.0

Selected resources:

/boids.png

Messages:

Figure 17: Resource files

### 7.1.5 Push registry settings (MIDP 2.0)

**Note : The Push tab is only available when creating a MIDP 2.0 package.**

The push registry allows MIDlets to set themselves up to be launched automatically without user initiation. The push registry manages network and timer initiated MIDlet activation. It allows an inbound network connection or a timer-based event to launch a MIDlet. For example, you can write an application that utilises network activation to wake up and process received email. Or you can use timer-based activation to schedule your MIDlet to synchronise data with a server and then go to sleep. For more information on the push registry, see the MIDP 2.0 specification [6]. On the *Push* tab (Figure 18) you can create a list of inbound connections for MIDlets. The Push attributes contain the following information:

- **Key** – The Push registration attribute name. A MIDlet suite can have multiple push registrations.

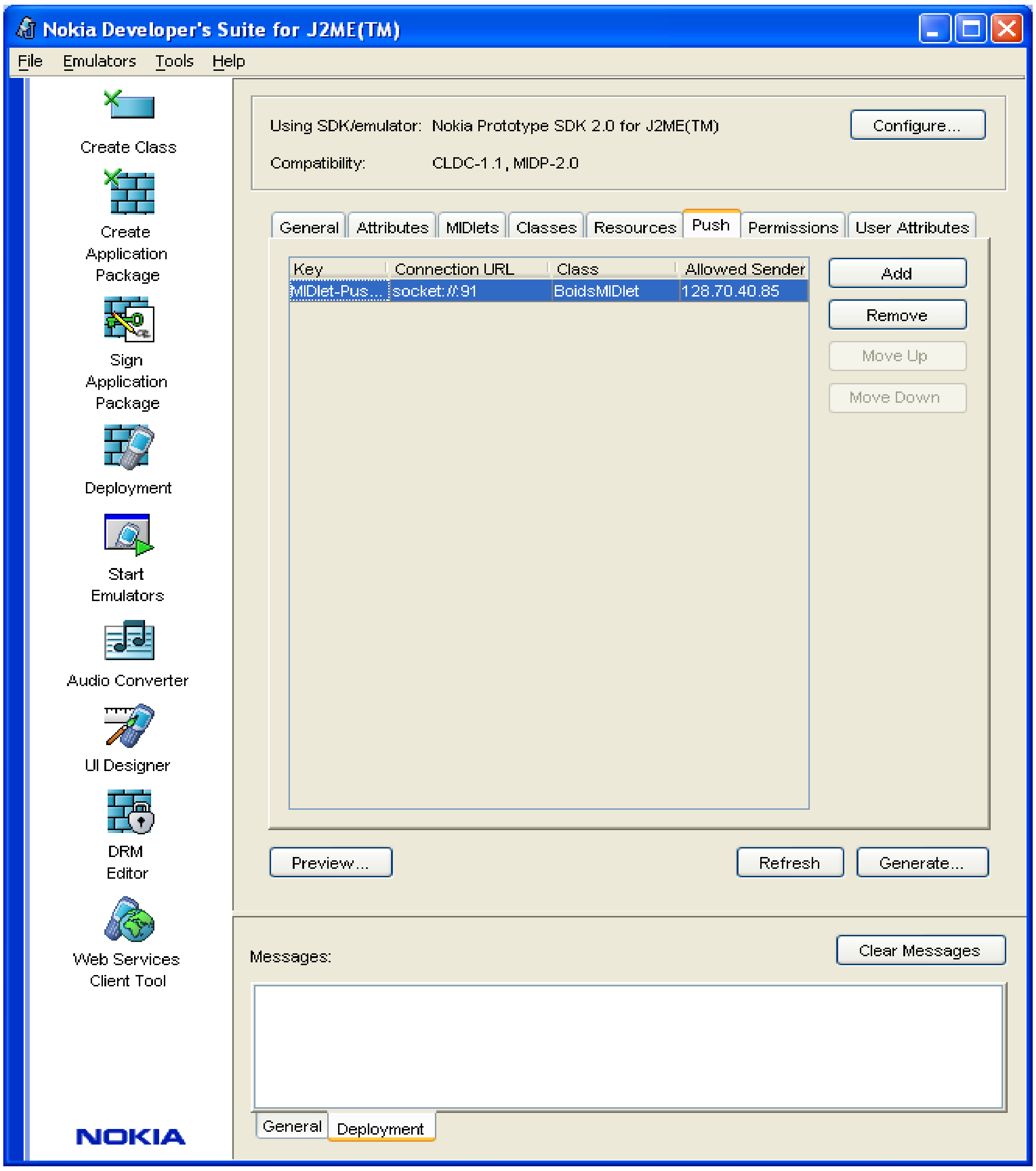
  Each key (registration) designation is unique and in the form MIDlet-Push-.

- **Connection URL** – The connection string used with the Connector.open() method.

- **Class** – The MIDlet class name for the MIDlet responsible for the connection.

- **Allowed Sender** – A filter to restrict which senders can launch the registered MIDlet.

Also, make sure the MIDlet has permission to access **javax.microedition.io.PushRegistry**. For instructions on how to set permissions, see below.



Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)

Compatibility: CLDC-1.1, MIDP-2.0

| Key | Connection URL | Class | Allowed Sender |
|---|---|---|---|
| MIDlet-Pus... | socket://:91 | BoidsMIDlet | 128.70.40.85 |

Messages:

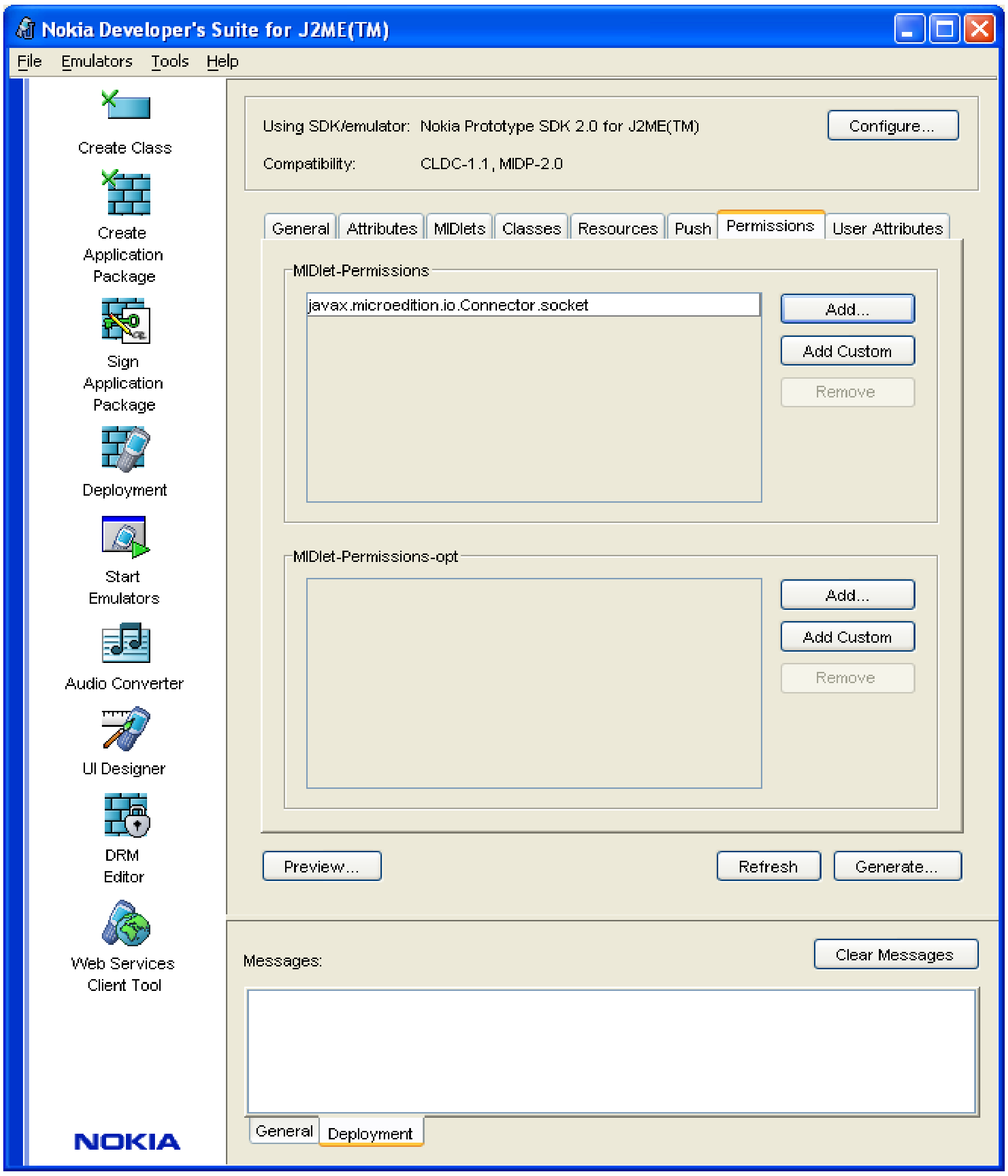
Figure 18: Push registry

### 7.1.6 Permissions for MIDlets (MIDP 2.0)

**Note : The Permissions tab is only available when creating a MIDP 2.0 package.**

The MIDP 2.0 specification [6] defines an open-ended system of permissions. To make any type of network connection, a MIDlet must have the appropriate permission. For example, a MIDlet that uses HTTP to talk to a server must have permission to open an HTTP connection. The permissions defined in MIDP 2.0 correspond to network protocols, but the architecture allows optional APIs to define their own permissions.

If your MIDlet needs to access certain protected APIs, you can select the needed permissions and optional permissions from the *Permissions* tab (Figure 19). For the required and optional permissions, **Add...** opens a list of permissions, from which you can choose the required ones. **Add Custom** adds an empty line to the list of permissions, which you can edit freely.



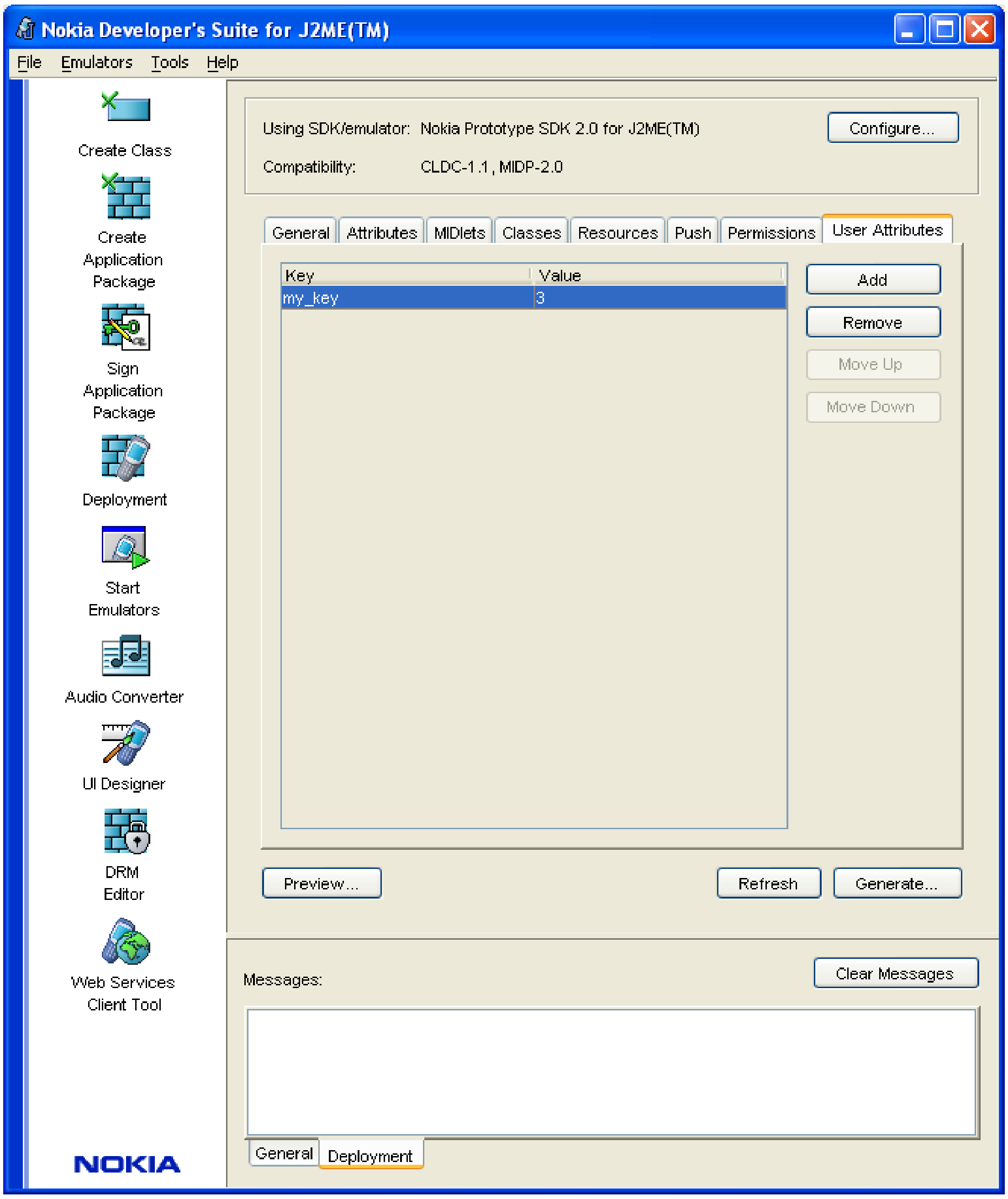
Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)
Compatibility: CLDC-1.1, MIDP-2.0

MIDlet-Permissions: javax.microedition.io.Connector.socket
MIDlet-Permissions-opt:
Messages:

Figure 19: MIDlet permissions

### 7.1.7 User attributes

In the *User Attributes* tab (Figure 20), set visible your own attributes for all the classes in the application. Name-value pairs define the attributes. To remove an attribute, select a row in the table and then click **Remove**. To insert a new row in the table, click **Add**. **Move up** and **Move down** buttons allow you to change the order of the attributes.



Using SDK/emulator: Nokia Prototype SDK 2.0 for J2ME(TM)

Compatibility: CLDC-1.1, MIDP-2.0

| Key | Value |
|---|---|
| my_key | 3 |

Messages:

Figure 20: Setting user attributes for a MIDlet Suite

## 7.2 Personal Profile Application Package

In addition to the MIDP application package creation, the Create Application Package tool also allows you to create Personal Profile (PP) application packages. You can specify mandatory .jar manifest tags required by the Personal Profile emulator and the included class files.

## 7.2.1 Personal Profile Attributes

To create a Personal Profile Application Package, configure the tool to use a Personal Profile emulator. For the PP application package you need to define at least the following required attributes on the *Manifest Attributes* tab (Figure 21):

- **JAR File Name** – The JAR file that is being created.

  Note: You need to define a full JAR file path, not only a relational path.

- **Main Class** – An executable class.

- **Ppro App Name** – A personal profile application name.

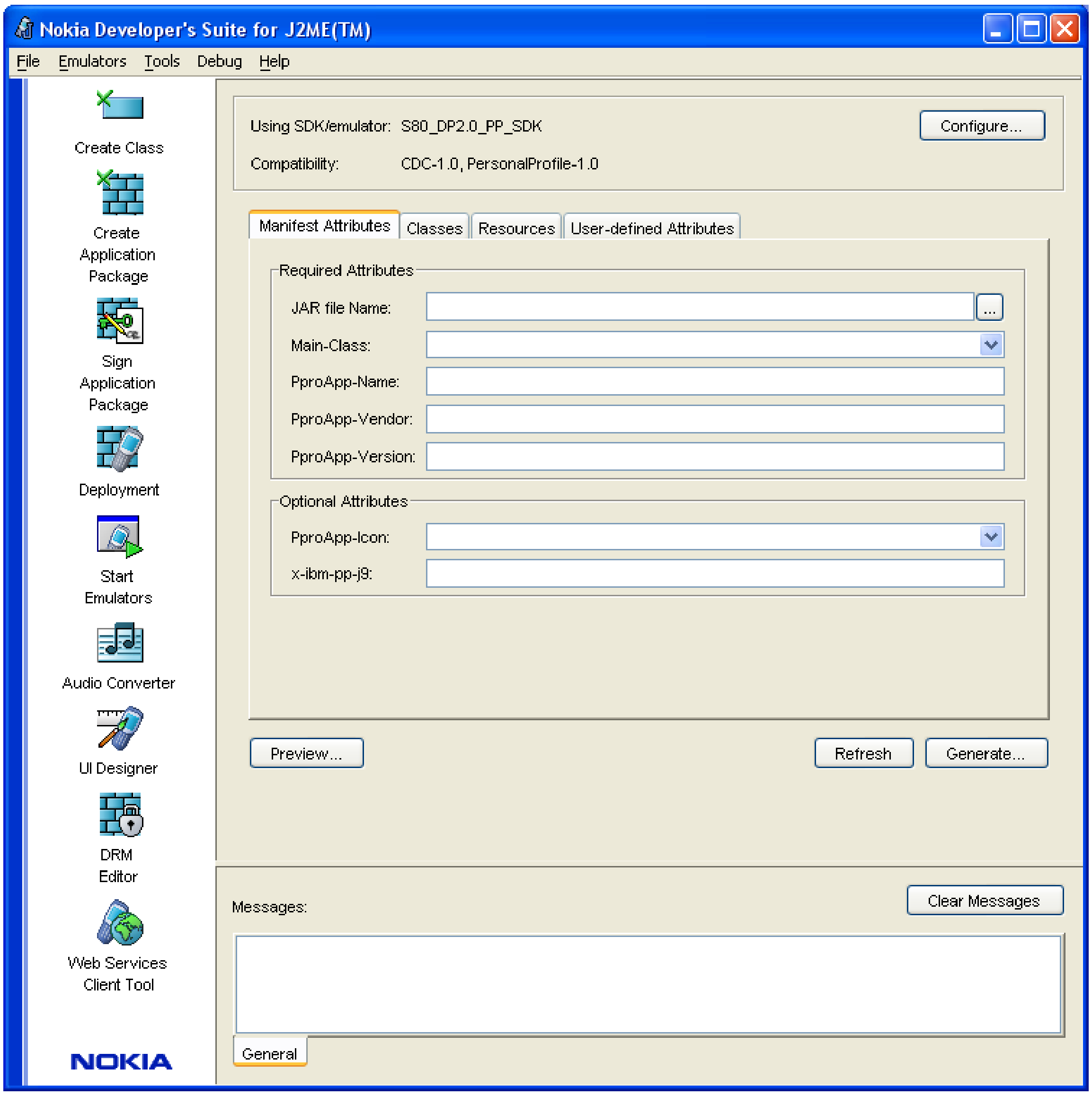
- **Ppro App Vendor** – A personal profile application vendor, e.g., “Nokia.”

- **Ppro App Version** – A personal profile application version.

Optionally you can also define:

- **Ppro App Icon** – An icon for the application.

- **x-ibm-pp-j9** – A parameter for the IBM virtual machine.



Using SDK/emulator: S80_DP2.0_PP_SDK

Compatibility: CDC-1.0, PersonalProfile-1.0

Required Attributes

| | |
|---|---|
| JAR file Name: | |
| Main-Class: | |
| PproApp-Name: | |
| PproApp-Vendor: | |
| PproApp-Version: | |

Optional Attributes

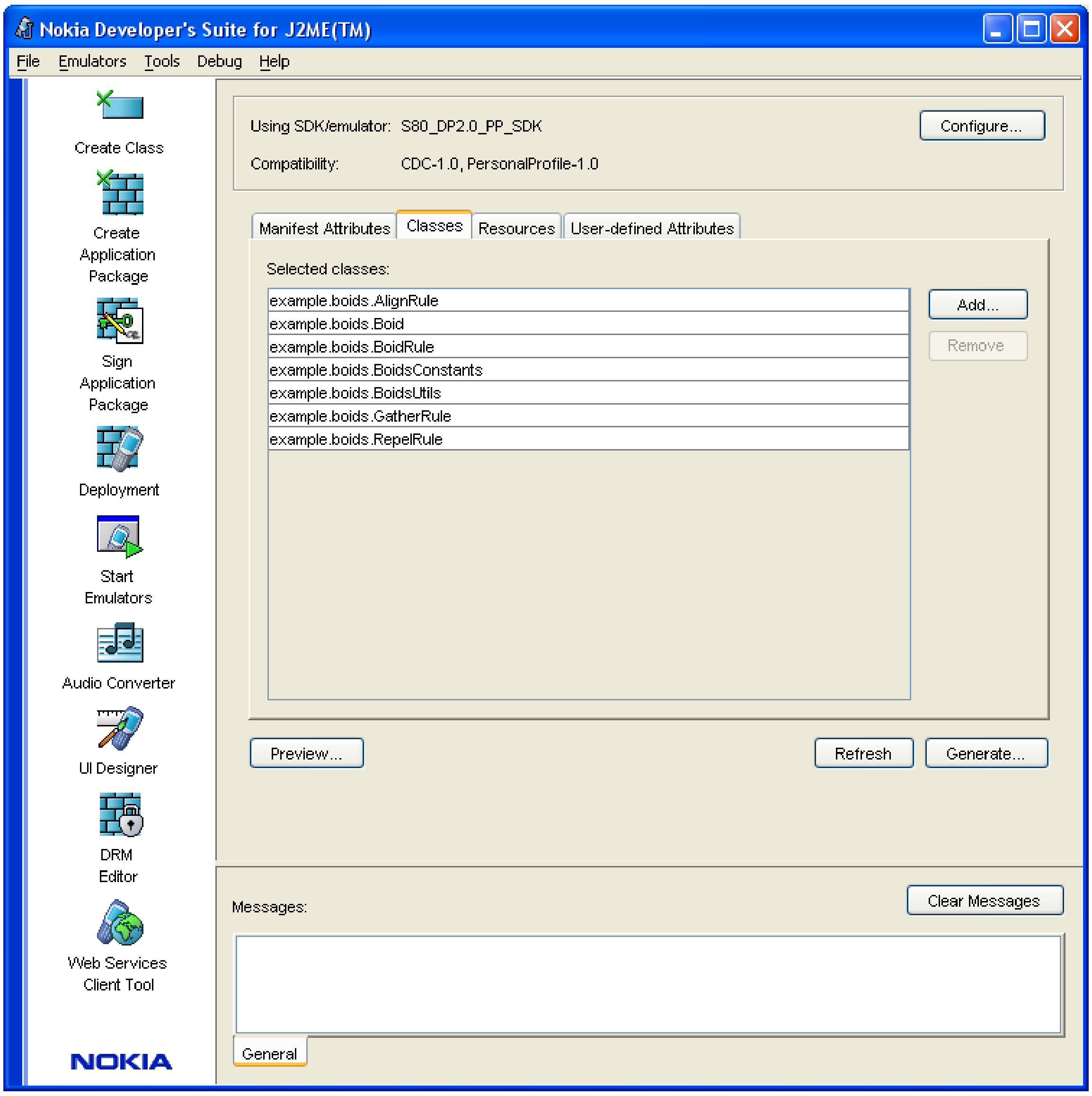
| | |
|---|---|
| PproApp-Icon: | |
| x-ibm-pp-j9: | |

Messages:

Figure 21: Setting attributes for a Personal Profile application package

### 7.2.2 Personal Profile Classes

Select classes for the application package on the *Classes* tab (Figure 22). Remove any unnecessary classes and add new required ones with the **Add** and **Remove** buttons.



Using SDK/emulator: S80_DP2.0_PP_SDK

Compatibility: CDC-1.0, PersonalProfile-1.0

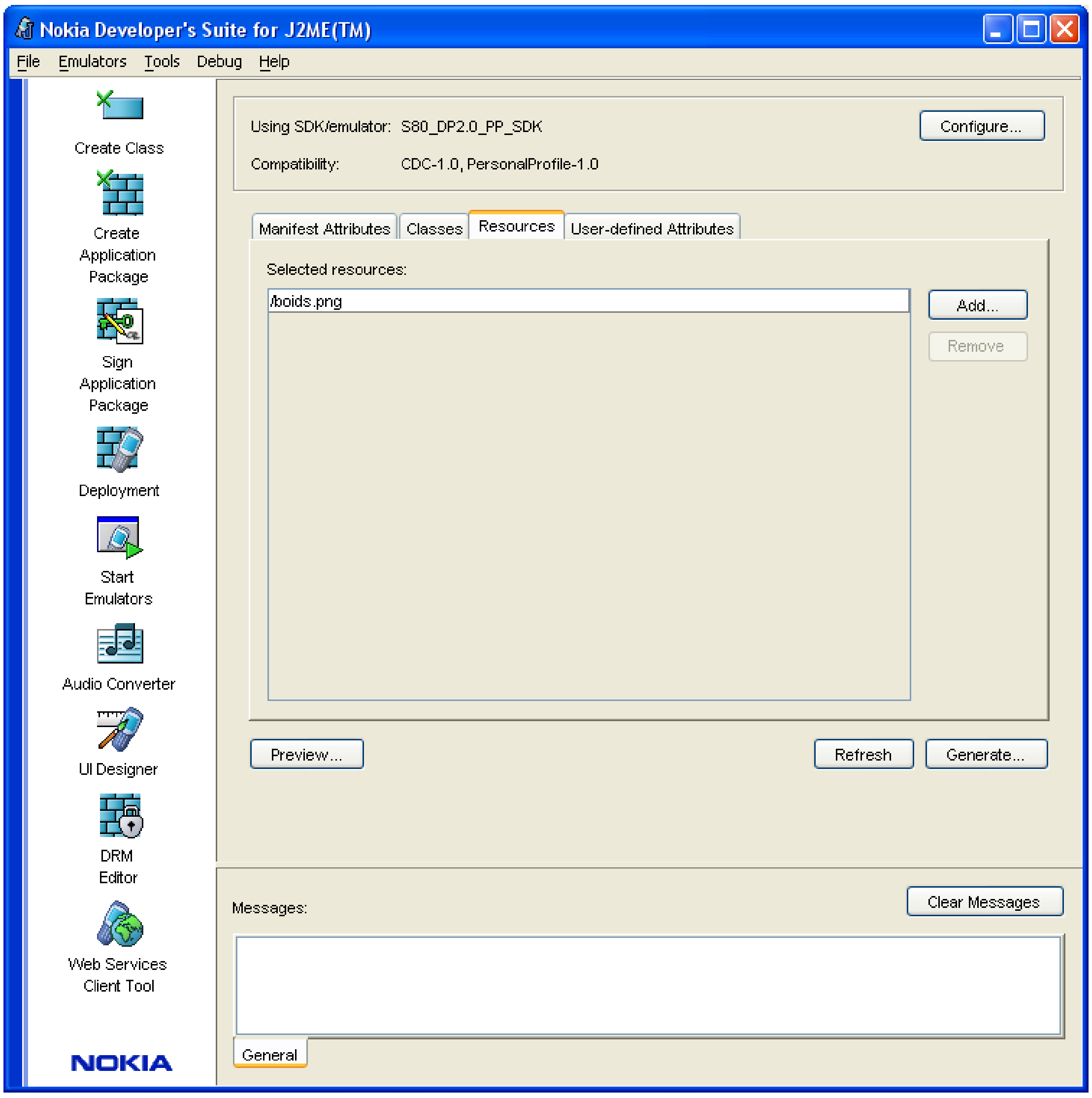
Selected classes:

```
example.boids.AlignRule
example.boids.Boid
example.boids.BoidRule
example.boids.BoidsConstants
example.boids.BoidsUtils
example.boids.GatherRule
example.boids.RepelRule
```

Figure 22: Setting classes for a Personal Profile application

### 7.2.3 Personal Profile Resources

Select other resources for the application package on the *Resources* tab (Figure 23). Remove any unnecessary resources and add new required ones with the **Add** and **Remove** buttons.



Using SDK/emulator: S80_DP2.0_PP_SDK

Compatibility: CDC-1.0, PersonalProfile-1.0

Selected resources:

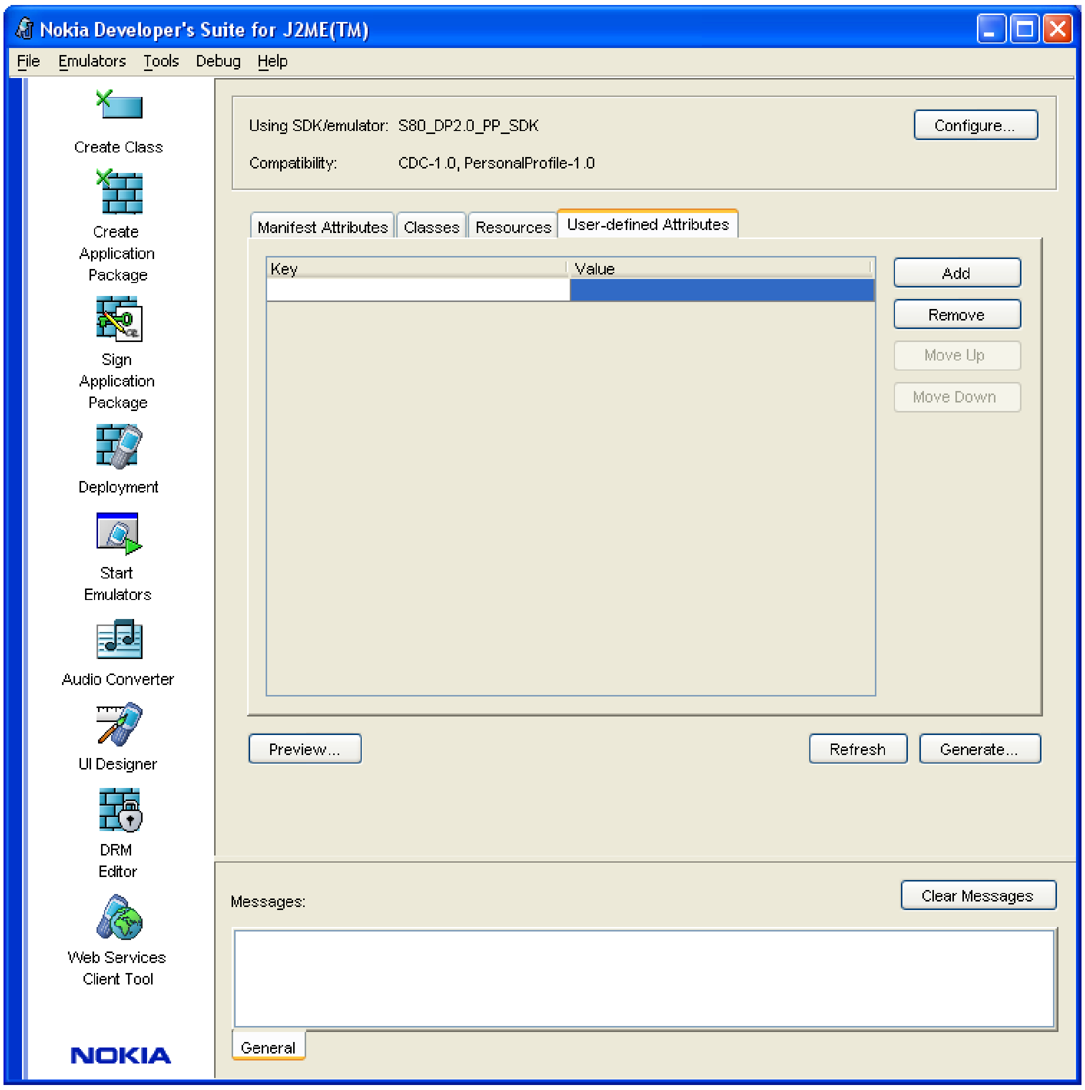
/boids.png

Messages:

Figure 23: Setting resources for a Personal Profile application

### 7.2.4 Personal Profile User Attributes

On the *User Attributes* tab (Figure 24), you can define attributes for all classes in the application package. Key-value pairs define the attributes. To remove an attribute, select a row in the table and then click **Remove**. To insert a new attribute, click **Add**. The **Move Up** and **Move Down** buttons allow you to change the order of the attributes.



Using SDK/emulator: S80_DP2.0_PP_SDK

Compatibility: CDC-1.0, PersonalProfile-1.0

Figure 24: Setting user attributes for a Personal Profile application

## 7.3 Creating the Application Package

If you compile, copy or delete class and resource files in any of the project directories while the tool is open, you should click the **Refresh** button to refresh file information in the tool. Otherwise the tool might not detect added or removed classes and resources until you actually generate the MIDP or Personal Profile Application Package.

When all the required attributes are set, the **Preview** and **Generate** buttons are enabled.

Clicking **Preview** displays the contents of the manifest for the JAR package of the MIDlet Application Package. The contents of the JAD file are also displayed with MIDP applications. MIDlet-Jar-Size and MIDlet-Jar-URL attributes are not displayed, as these are not determined until the files are actually created.

When generating a MIDlet Application Package, the preverifier of the default emulator preverifies the selected classes. The **Generate** button generates and saves the generated JAR and JAD files. The default location for saving the files is the project root directory or, if running Nokia Developer's Suite 3.0 for J2ME™ in stand-alone mode, the working directory defined in *Preferences* (see Chapter 4). The result of the pre-verification and packaging the files is displayed in a message window.

# 8 Running Applications on Emulators

*Start Emulators* tool allows you to emulate MIDlets and Personal Profile applications. Each emulator contains one or more devices that can be used for emulation. A device can be run in one of two modes. More common method for development is running an application directly from the local file system. The second is to run the device using Over the Air (OTA) provisioning. When running applications from the file system, emulation is started with a MIDlet class, an Application Descriptor (JAD) or JAR file, please see SDK's documentation for more help. When using OTA, the application is loaded via HTTP to the device. Nokia Developer's Suite 3.0 for J2ME™ includes a local HTTP server to test OTA provisioning with files located in the local file system.

**Note : All emulators/devices do not support OTA provisioning mode.**

The tool is started from the main menu. Select:

*Emulators | Start Emulators*

or press the **Start Emulators** button in the main window of the stand-alone application.

## 8.1 Running Applications from the Local File System

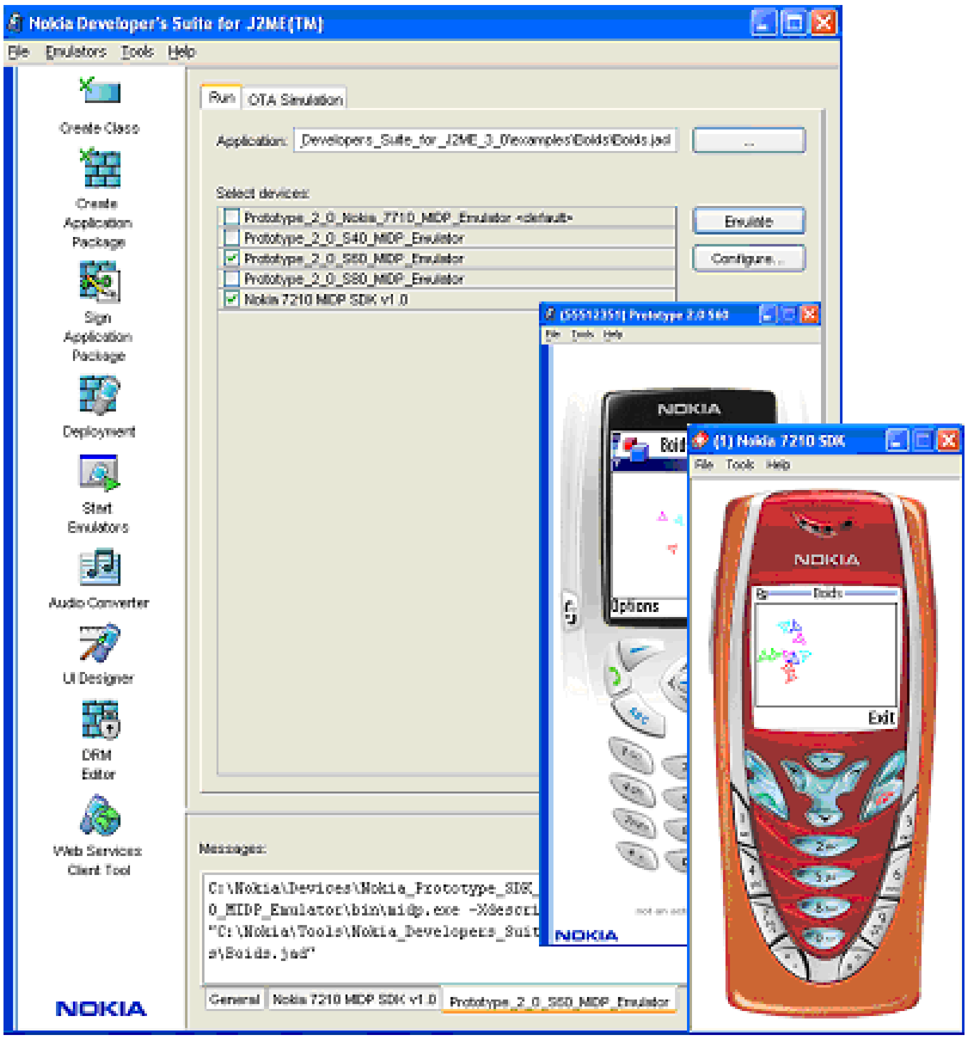
Browse for a JAD file, a MIDlet class file or a JAR file in the *Application* field or type a file location. If you use a MIDlet class file, the file must be in the user class directory or one of its subdirectories.

**Note : To start an application from a class file, the class path must match the working directory specified in Preferences. You can check and change the working directory by selecting *File | Preferences* from the menu bar. The defined *Classes* directory is used to run applications.**

Each emulator may contain one or more devices. All available devices are listed in the *Select Devices* list. If you specified a JAD or a MIDlet class file in the *Application* field, MIDP emulators are enabled and the emulators that support Personal Profile are disabled. If you selected a JAR file, Personal Profile emulators are enabled and MIDP emulators are disabled. From the list of devices, select the ones that will be launched. If you wish to add new devices or configure device parameters, click the **Configure...** button.

Emulators execute the application file specified in the text field when you click **Emulate**. If you use a class, a warning will appear if any of the selected devices does not support this feature. All devices are executed in their own processes so they do not share the same instance of the application. Figure 25 shows two emulators running an application.

Messages from the devices are displayed in a message window. Emulators are stopped by their own operations.



Application: Developers_Suite_for_J2ME_3_0\examples\Boids\Boids.jad

Select devices:

- [ ] Prototype_2_0_Nokia_7710_MIDP_Emulator <default>
- [ ] Prototype_2_0_S40_MIDP_Emulator
- [✓] Prototype_2_0_S60_MIDP_Emulator
- [ ] Prototype_2_0_S80_MIDP_Emulator
- [✓] Nokia 7210 MIDP SDK v1.0

Messages:

```
C:\Nokia\Devices\Nokia_Prototype_SDK_
0_MIDP_Emulator\bin\midp.exe -Xdescri
"C:\Nokia\Tools\Nokia_Developers_Suit
s\Boids.jad"
```

General | Nokia 7210 MIDP SDK v1.0 | Prototype_2_0_S60_MIDP_Emulator

Figure 25: Two emulators running

## 8.2 Over The Air (OTA) provisioning

Some emulators and devices support OTA provisioning emulation. Using features provided in *OTA Simulation* tab (Figure 26), you can emulate OTA-related functions.

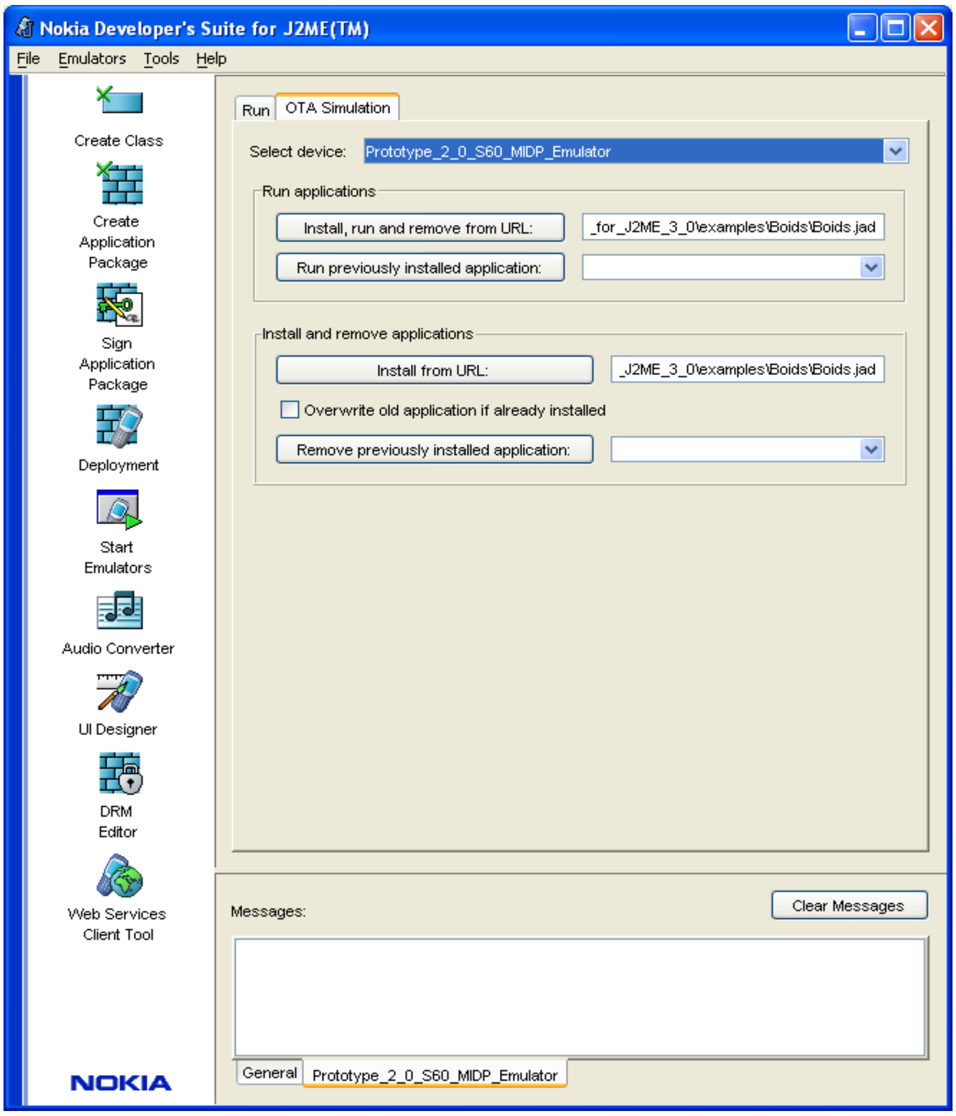

Nokia Developer's Suite for J2ME(TM)
File Emulators Tools Help
Create Class
Create Application Package
Sign Application Package
Deployment
Start Emulators
Audio Converter
UI Designer
DRM Editor
Web Services Client Tool
NOKIA
Run
OTA Simulation
Select device:
Prototype_2_0_S60_MIDP_Emulator
Run applications
Install, run and remove from URL:
_for_J2ME_3_0\examples\Boids\Boids.jad
Run previously installed application:
Install and remove applications
Install from URL:
_J2ME_3_0\examples\Boids\Boids.jad
Overwrite old application if already installed
Remove previously installed application:
Messages:
Clear Messages
General
Prototype_2_0_S60_MIDP_Emulator

Figure 26: OTA provisioning

Devices, which support OTA, are listed in the *Select device* combo box. If a device does not support a particular feature of OTA functionality, it will stay disabled. When you select a device from the list, Nokia Developer's Suite 3.0 for J2ME™ will query the device if it has applications installed. In this tool any URL can be replaced with absolute path to a file in the local file system since Nokia Developer's Suite 3.0 for J2ME™ offers a HTTP server that will send the files to the emulator.

This tool supports the following OTA provisioning functions:

- **Install, run and remove from URL -** This allows you to run an application in so-called transient mode, in which an application is sequentially installed, run and removed from a device.
- **Run previously installed application -** Select a previously installed application from the combo box and press this button to run it.
- **Install from URL -** Installs the application to the device using OTA provisioning.

- **Overwrite old application if already installed -** Aa application with the same storage name can be discovered during installation of an application. This option allows the device to forcefully remove the old application prior to installing the new application. If this option is not selected installer is not allowed to overwrite the old application.
- **Remove previously installed application -** Select a previously installed application from the combo box and press this button to remove it.

# 9 Sign Application Package

Use the *Sign Application Package* tool to sign a MIDlet Application Package (Figure 27). Signing a MIDlet allows the user to authenticate the sender of the application through the use of certificates and ensures the integrity of the application with public/private key security features. The signature of the JAR is created with the signer's private key. The signature and public key certificate are added to the JAD as attributes. The device uses them to verify the signature and to complete the authentication using a root certificate bound to a protection domain on the device. For a thorough discussion of the MIDP 2.0 security model, see the MIDP 2.0 specification [6].

Choose *File* | *Sign Application Package*

or press the **Sign Application Package** button in the main window to start using this tool.

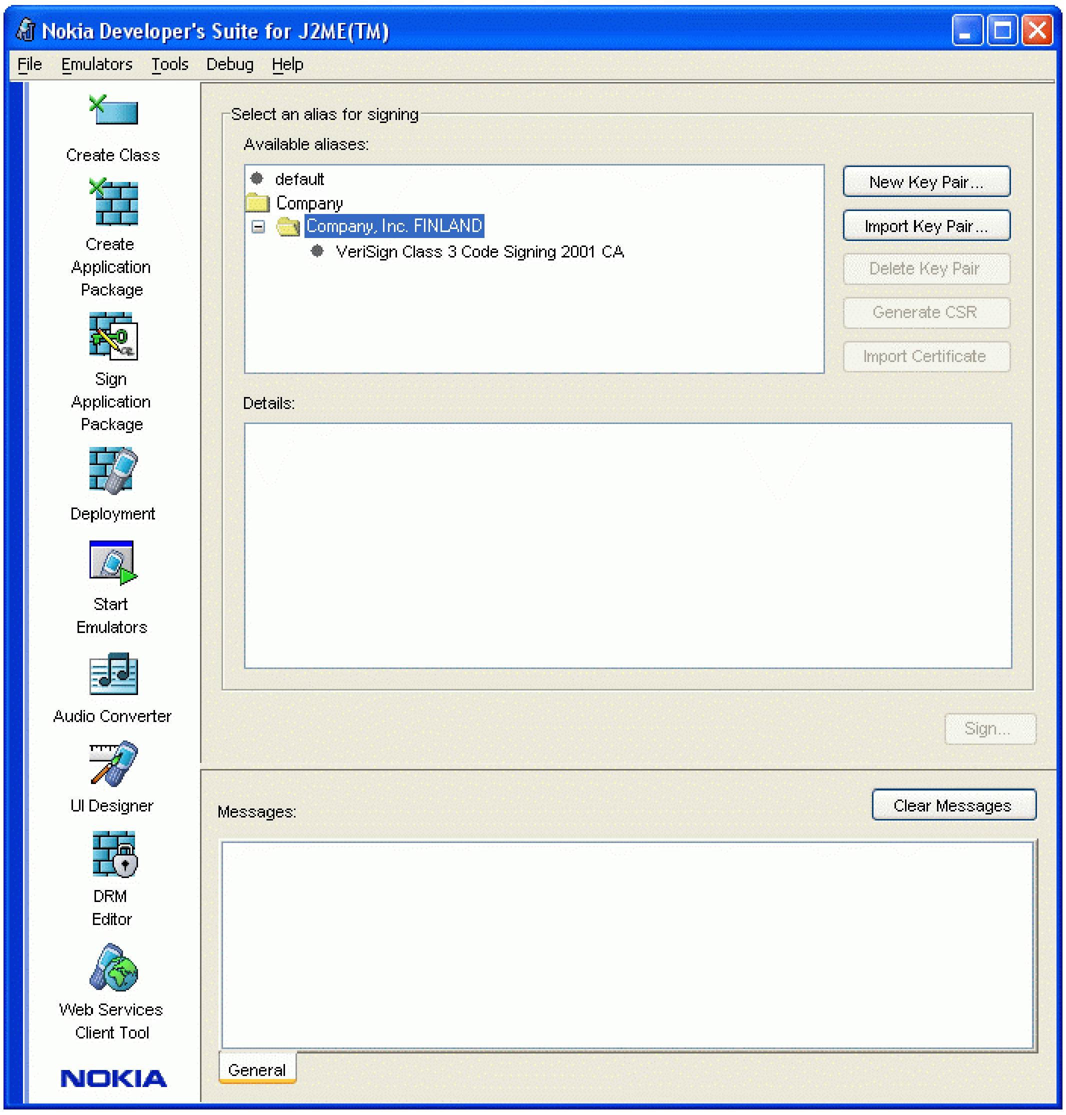


Figure 27: Sign Application Package tool

## 9.1 Signing

After you have created a MIDlet application package, you can sign it using a public/private key pair and an associated public key certificate. Each key pair and the associated public key certificate are identified with a name, an alias. In the *Available aliases* tree, you'll find the key pairs currently stored in Nokia Developer's Suite 3.0 for J2ME™ key store, to whom the certificate has been granted and an issuer of the certificate. In the *Details* text area below, you can see detailed information about the currently selected key pair and the certificate. There are also five buttons for managing the keys and public key certificates in the key store, which are described later.

To actually sign an application, select an alias for the key pair you wish to use and press **Sign...**. This will bring up a file dialog, from which you must select the MIDlet application package's JAD file. The private key and the public key certificate corresponding to the selected alias are used in signing the MIDlet. The tool will notify you when it has successfully signed the application package.

You can sign the application package with several keys. If you sign the application package with different keys, each signed application package is saved.

**Note : The default key is a sample dummy key, Nokia SDKs and devices don't have certificates that would allow you to use the default key for verifying MIDlets.**

## 9.2 Key pair management

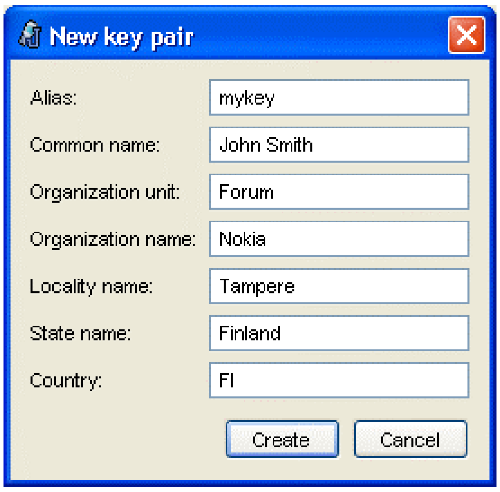
A key pair in Nokia Developer's Suite 3.0 for J2ME™ is a pair of public and private keys that can be used in signing MIDlets. A key pair is always associated with a public key certificate. The following functions manage key pairs.

### 9.2.1 Creating a new key pair

If you need to create a key pair, press **New Key Pair...** button. In the dialog box that opens (Figure 28), specify the *Distinguished Name* for the key pair with the following fields:

- **Common name** – full name of person or object, e.g., "John Smith"
- **Organisation unit -** small organisation (e.g, department or division) name, e.g., "Forum"
- **Organisation name -** large organisation name, e.g., "Nokia"
- **Locality name -** locality (city) name, e.g., "Tampere"
- **State -** state or province name, e.g., "Finland"
- **Country -** two-letter country code, e.g., "FI"

The tool then creates a public and a private key that are referenced by the given alias. A self-signed public key certificate for the key pair is also generated and all information is then stored to the key store.



| | |
|---|---|
| Alias: | mykey |
| Common name: | John Smith |
| Organization unit: | Forum |
| Organization name: | Nokia |
| Locality name: | Tampere |
| State name: | Finland |
| Country: | FI |

Figure 28: New key pair dialog

### 9.2.2 Importing a key pair

If you want to use an existing key pair for signing press **Import Key Pair…** button. Locate the key store from which you want to import a key pair and select it in the dialog. You are asked to provide a password for the key store in question. Next, you see a combo box, which displays the aliases in the key store. Choose the alias for the key pair you want to import and press **OK**. Provide the password for the alias if you are asked for it. You can now see the imported key pair in the tree of available aliases. The alias is also selected after importing and its details are shown in the text area below the tree structure.

**Note : Imported keys used for signing should use RSA as their encryption algorithm.**

### 9.2.3 Deleting a key pair

To delete a key pair or several key pairs from the key store select the aliases and press **Delete Key Pair**. You are asked to confirm the deletion before the key pairs are actually removed.

## 9.3 Public key certificate management

In Nokia Developer's Suite 3.0 for J2ME™, a key pair and its alias are always associated with a public key certificate. Successful signing of a MIDlet requires a public key certificate that can be validated to one of the protection domain root certificates on the device. A public key is used to verify the signature in the MIDlet Application Package and it is provided to the device as a RSA X.509 certificate included in the application descriptor (JAD). This public key certificate can be obtained from a Certificate Authority.

### 9.3.1 Generating a Certificate Signing Request (CSR)

To obtain a certificate from a Certificate Authority (CA), you must generate a request for the certificate. To do this, press the Generate CSR button. You can then select the file to which the CSR will be written. After writing the file you also have the option to copy the CSR contents to clipboard, to be used in, for example, e-mailing the request to the CA.

### 9.3.2 Importing the public key certificate

When you have sent the CSR to a CA, you receive a signed RSA X.509v3 certificate for your public key from the CA. You can now import it to the corresponding alias with **Import Certificate** feature. If the public keys in the key store and certificate do not match, an error message is displayed. In case the certificate is imported successfully, you are also notified. You are now ready to use the key pair and its public key certificate for signing MIDlet Application Packages.

# 10 Deployment

Use the *Deployment* tool to deploy applications to an actual device or server. The tool is started from the main menu. Select:

*File | Deployment*

or click the **Deployment** button in the main window of the stand-alone application.

**Note : Nokia PC Suite 6.5 enables communication between Nokia handsets and a PC. Make sure that you have Nokia PC Suite 6.5 installed on your PC. The PC Suite installer is delivered in the Nokia Developer's Suite 3.0 for J2ME™ delivery archive (nds_jme_v3_0.zip) or you can download it from the Nokia website at www.nokia.com/pcsuite.**

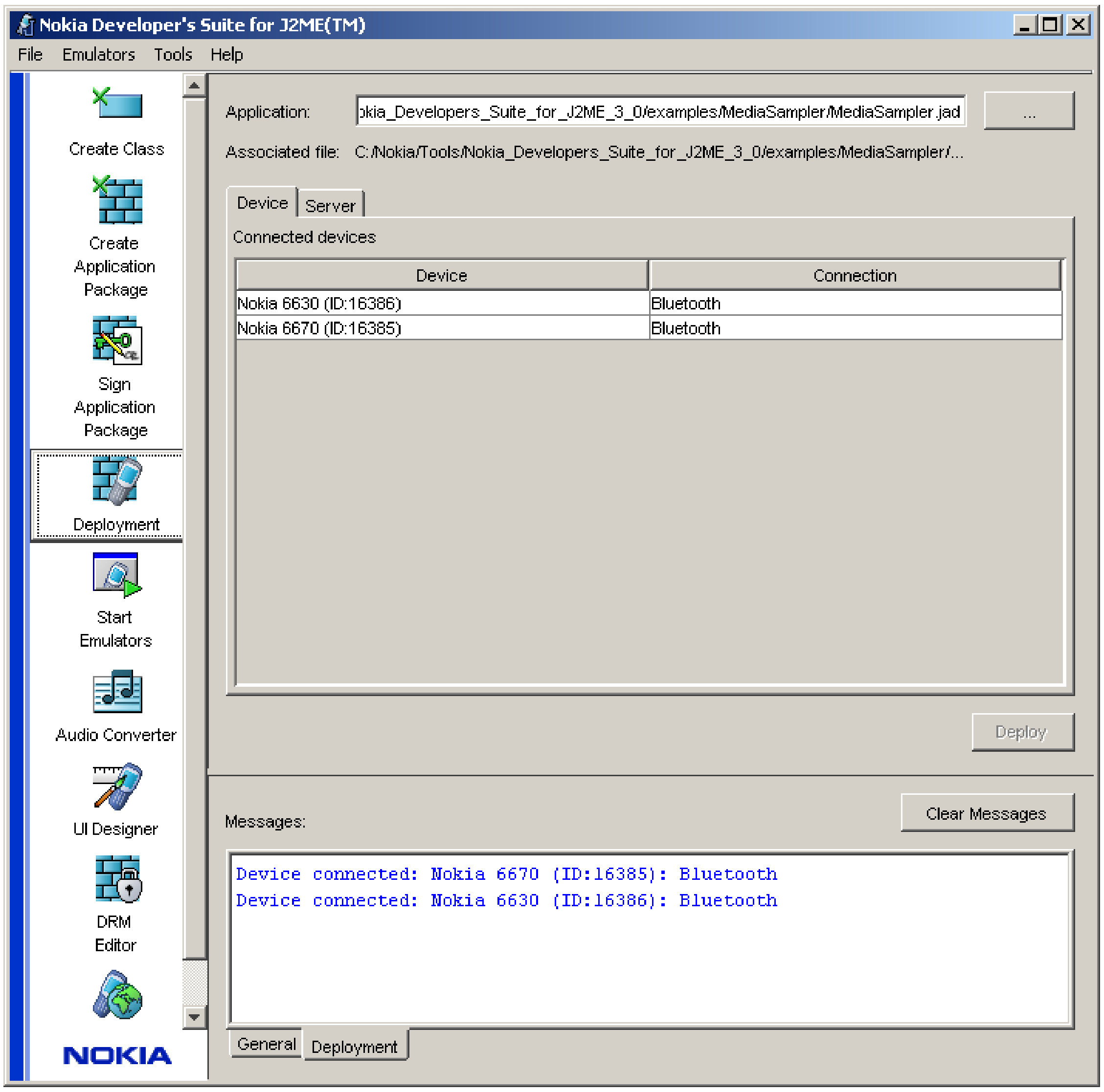
## 10.1 Device deployment

**Note : Before you can deploy applications to a real device via Bluetooth, make sure that you have configured and created a Bluetooth connection between your PC/laptop and the actual device.**

To deploy an application to a device, select a JAD, JAR or a SIS file using the browse button or type the file location directly into the text field (Figure 29). To deploy an application package consisting of a JAD file and a JAR file, both files must be found in the local file system. Particularly, JAD file's MIDlet-Jar-URL attribute must point to the correct location where the JAR file can be found. If the JAR is not found, an error message is displayed.

Having selected the application file(s), connect the device. The list of devices shows all devices connected to the computer using any of the supported media. This list is updated automatically when you connect or disconnect a device. Nokia Developer's Suite 3.0 for J2ME™ supports connections using Bluetooth, IrDA, USB and RS-232 serial cable. Consult your Nokia devices user's guides for more information on the media that your devices support.

Once you have selected a device, you can deploy the application files by clicking the **Deploy** button. Deployment status is shown in the message area.



Application: okia_Developers_Suite_for_J2ME_3_0/examples/MediaSampler/MediaSampler.jad

Associated file: C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME_3_0/examples/MediaSampler/...

Connected devices

| Device | Connection |
|---|---|
| Nokia 6630 (ID:16386) | Bluetooth |
| Nokia 6670 (ID:16385) | Bluetooth |

Messages:

```
Device connected: Nokia 6670 (ID:16385): Bluetooth
Device connected: Nokia 6630 (ID:16386): Bluetooth
```

Figure 29: Device deployment

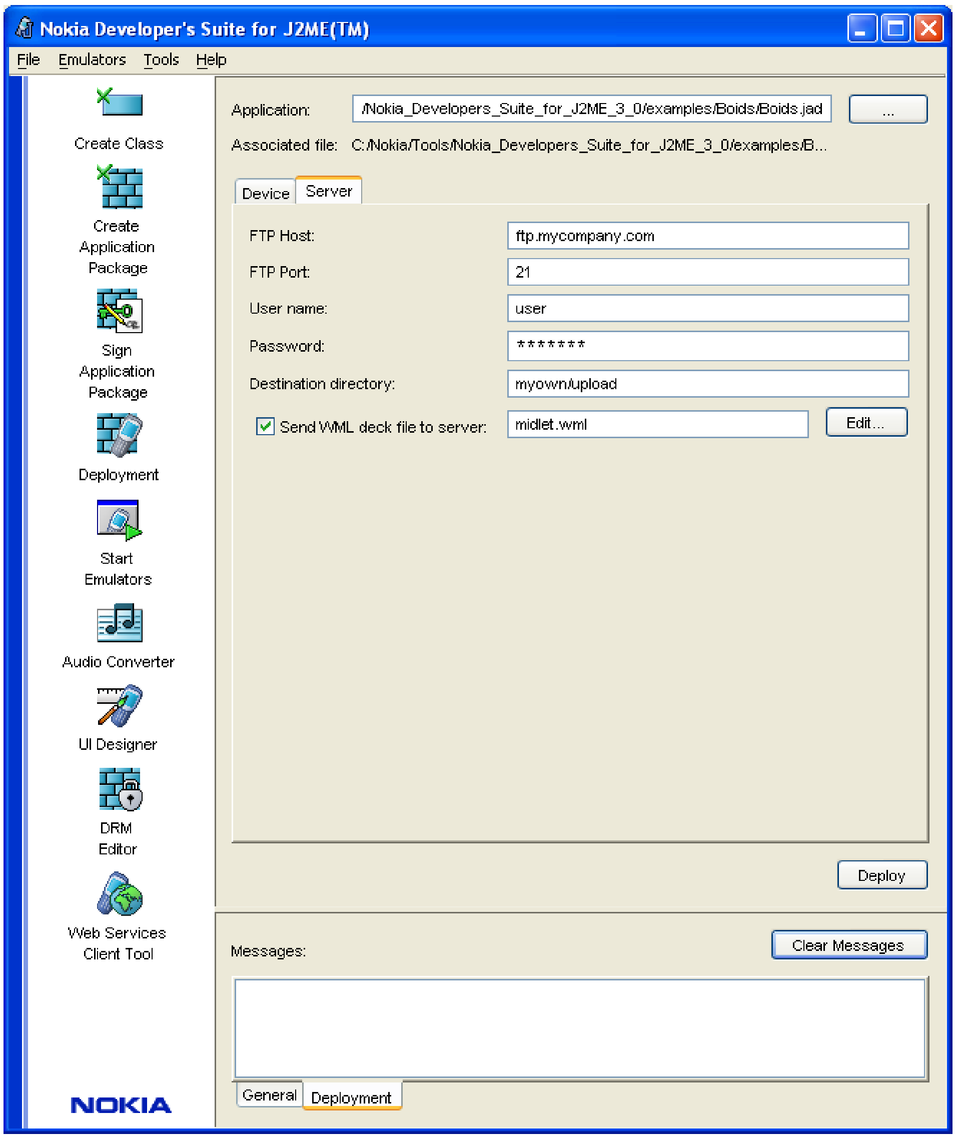
## 10.2 Server deployment

Server deployment opens a connection to the server via Passive FTP and sends the files of an Application Package (JAD and JAR) to the designated directory on the server (Figure 30). There is an option to send a default WML deck to the server in addition to the JAR and JAD files. The default WML deck simply acts as a link to the deployed MIDlet. When you are running the server so that you can browse the WML deck with your device, a link to the MIDlet you deployed becomes visible and it can be downloaded over the air.

To enable FTP deployment, enter the following information: *FTP host*, *FTP port* to be used (normally 21), *username* for logging into the FTP server, *password* to use when logging into the FTP server and the *directory* where files will be uploaded.

Check the *Send WML file to the server* option if you want a WML deck uploaded with the JAR and JAD files. Edit the contents of the WML deck with the **Edit...** button.

When all information is set, click **Deploy** to deploy the files. Deployment status is shown in the message area.



Application: /Nokia_Developers_Suite_for_J2ME_3_0/examples/Boids/Boids.jad

Associated file: C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME_3_0/examples/B...

FTP Host: ftp.mycompany.com

FTP Port: 21

User name: user

Password: *******

Destination directory: myown/upload

[✓] Send WML deck file to server: midlet.wml

Figure 30: Server deployment

To retrieve the application from the server to the device, you will need to point to the WML page using the browser on your device. Figure 31 demonstrates this procedure. You must have an access point to your server, provided by your operator or by yourself. For more information, contact your operator and visit the Forum Nokia website at www.forum.nokia.com.

**Note : Some devices may require a manual removal of the deployed files before you can deploy new files with the same names.**

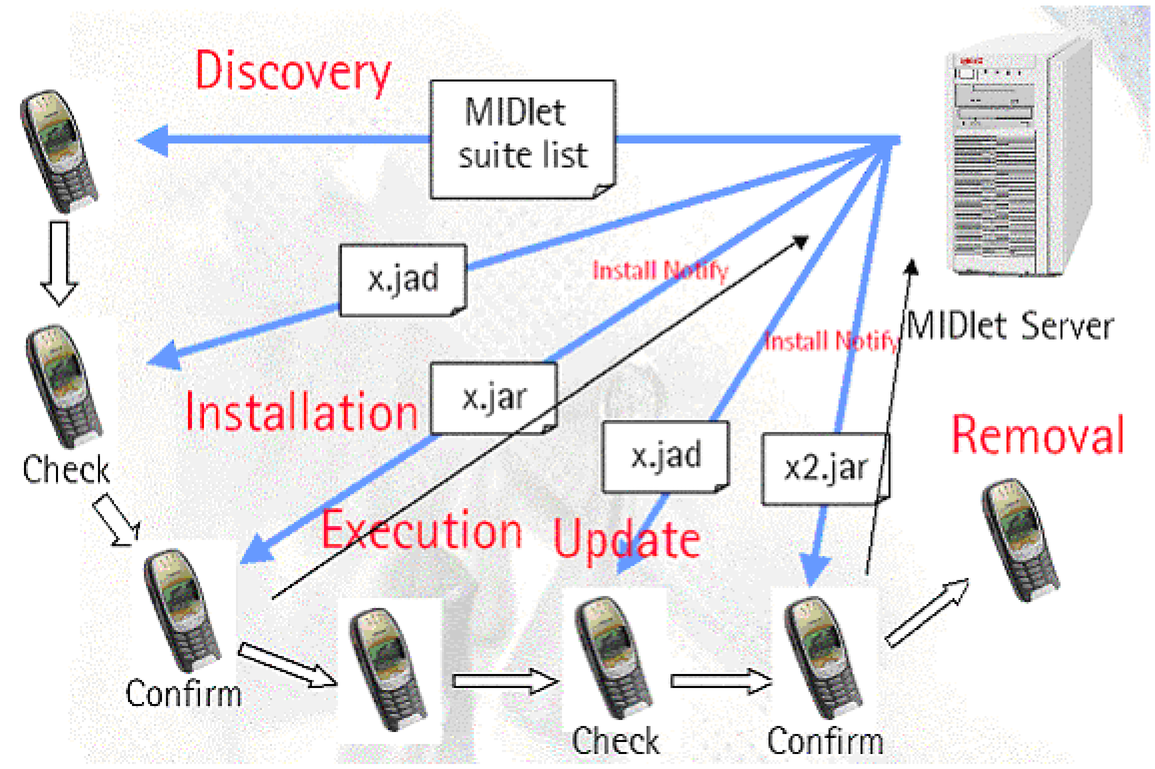


Figure 31: Server to device procedure

# 11 Audio Converter

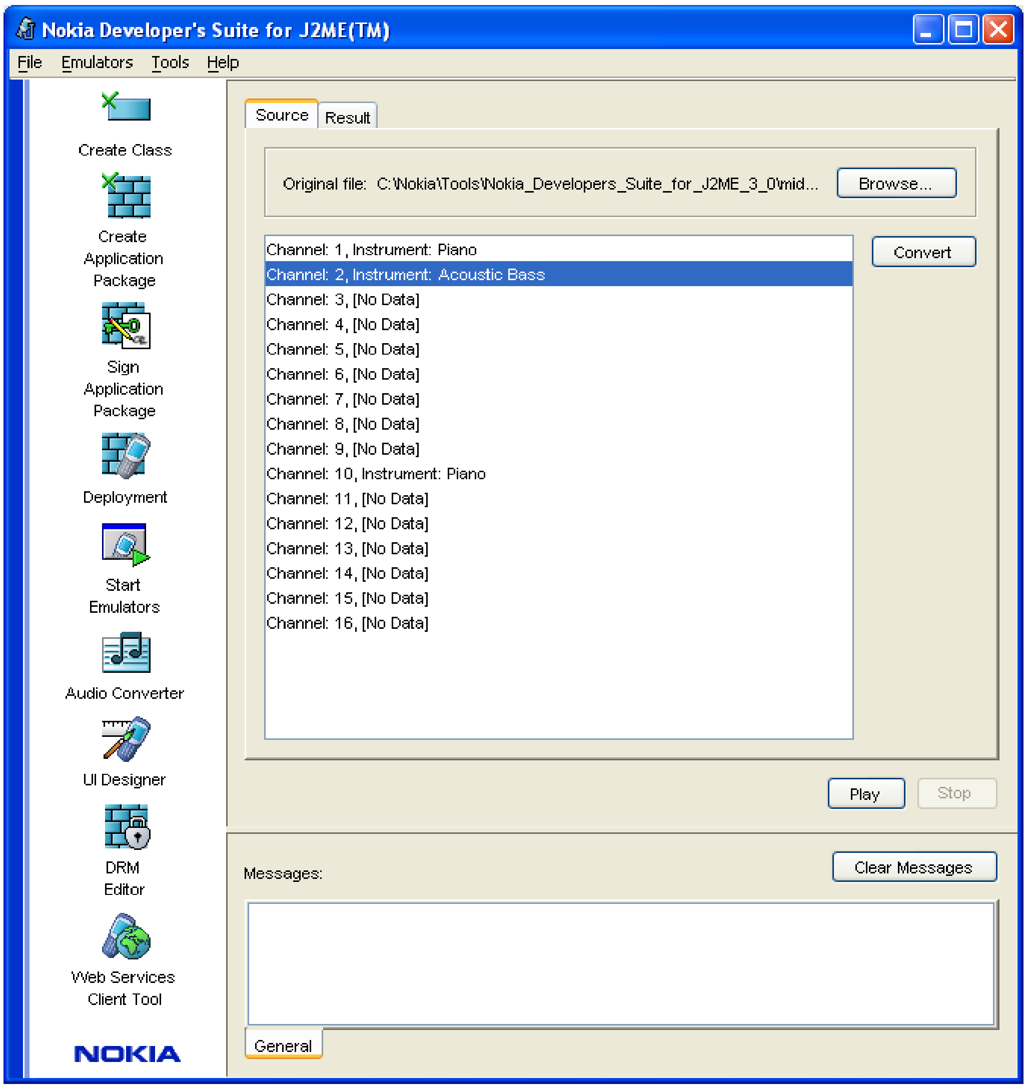
Use the *Audio Converter* tool to convert MIDI or Ringing tone XML files to Ringing tone audio resources. The tool is started from the main menu. Select:

*File | Audio Converter*

or press **Audio Converter** button in the main window of the stand-alone application.

## 11.1 Converting MIDI files

Open a MIDI file with **Open**. A list of channels appears in the window (Figure 32). You can now play the MIDI file with **Play**. It is not possible to convert more than one MIDI channel at any time. Select the channel to be converted from the list and listen to it by pressing **Play**. To convert a channel, **Stop** the audio from playing, select a channel and press **Convert**. The results of the conversion can be seen and listened to under *Result* tab.



Original file: C:\Nokia\Tools\Nokia_Developers_Suite_for_J2ME_3_0\mid...

```
Channel: 1, Instrument: Piano
Channel: 2, Instrument: Acoustic Bass
Channel: 3, [No Data]
Channel: 4, [No Data]
Channel: 5, [No Data]
Channel: 6, [No Data]
Channel: 7, [No Data]
Channel: 8, [No Data]
Channel: 9, [No Data]
Channel: 10, Instrument: Piano
Channel: 11, [No Data]
Channel: 12, [No Data]
Channel: 13, [No Data]
Channel: 14, [No Data]
Channel: 15, [No Data]
Channel: 16, [No Data]
```

Messages:

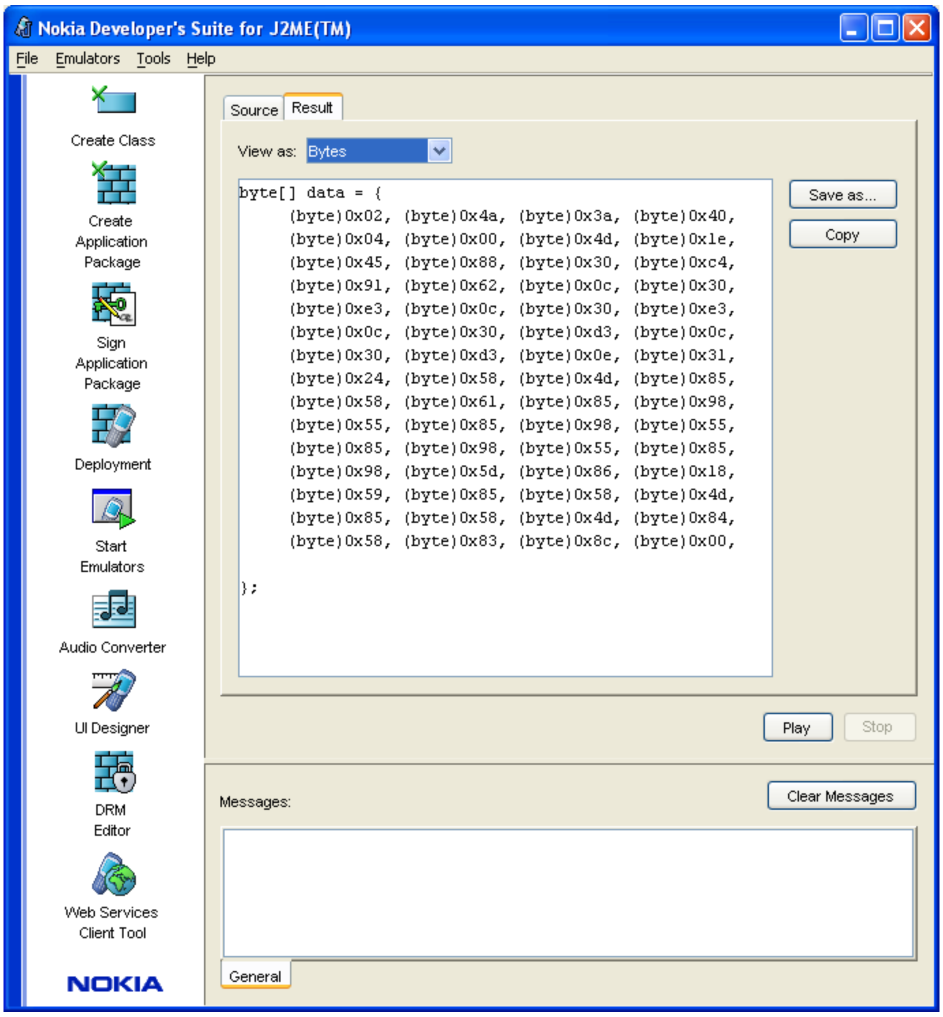
Figure 32: Converting a MIDI channel

## 11.2 Converting XML files

Open an XML file with **Open**. The contents of the file appear in the window. Play the XML file with **Play**. You can also edit the contents of the file and play it several times to listen to the changes. To convert the XML contents, **Stop** the audio from playing and press **Convert**. The results of the conversion can be seen and listened to under *Result* tab.

## 11.3 Results

The results of the conversion can be seen under the *Result* tab (Figure 33). Select the format you wish to view from the View as combo box. The choices are *OTA* (Over the Air), *Bytes* (Java source) and *Ringing Tone XML*. Obviously, if you converted from XML, the resulting XML will look and sound exactly the same. Listen to the results of the conversion with **Play**. To save the results to a file in any of the three formats, press **Save as...** button. To copy the displayed content to clipboard press **Copy** button.



View as: Bytes

```
byte[] data = {
      (byte)0x02, (byte)0x4a, (byte)0x3a, (byte)0x40,
      (byte)0x04, (byte)0x00, (byte)0x4d, (byte)0x1e,
      (byte)0x45, (byte)0x88, (byte)0x30, (byte)0xc4,
      (byte)0x91, (byte)0x62, (byte)0x0c, (byte)0x30,
      (byte)0xe3, (byte)0x0c, (byte)0x30, (byte)0xe3,
      (byte)0x0c, (byte)0x30, (byte)0xd3, (byte)0x0c,
      (byte)0x30, (byte)0xd3, (byte)0x0e, (byte)0x31,
      (byte)0x24, (byte)0x58, (byte)0x4d, (byte)0x85,
      (byte)0x58, (byte)0x61, (byte)0x85, (byte)0x98,
      (byte)0x55, (byte)0x85, (byte)0x98, (byte)0x55,
      (byte)0x85, (byte)0x98, (byte)0x55, (byte)0x85,
      (byte)0x98, (byte)0x5d, (byte)0x86, (byte)0x18,
      (byte)0x59, (byte)0x85, (byte)0x58, (byte)0x4d,
      (byte)0x85, (byte)0x58, (byte)0x4d, (byte)0x84,
      (byte)0x58, (byte)0x83, (byte)0x8c, (byte)0x00,

};
```

Figure 33: Results of the conversion

## 12 UI Designer

Use the UI Designer tool to design MIDP User Interface layouts, tiled layers for games and screen flows between UI components. The tool can be used with an integrated IDE or with a standalone Nokia Developer's Suite for J2ME.

**Note : Your default emulator selection affects the use of the UI Designer. UI Designer is a MIDP-based tool, so you cannot use it properly if your default emulator is a Personal Profile SDK. To change the default emulator, select *Emulators* | *Configure Emulators...* from the menu bar and select a new default emulator.**

Integration with Borland JBuilder: Open a Form or a LayerManager class and access the UI Designer view through the *Nokia UI Designer* tab in the JBuilder application window.

Integration with Sun Java Studio or NetBeans: Select a Form or a LayerManager class and then select *Open with Nokia UI Designer* from the drop-down list.

**Note : The UI Designer requires that JDK 1.4.1 or later has been defined as the JDK in the Sun Java Studio IDE configuration file ..\me\bin\ide.cfg**

Standalone Nokia Developer's Suite for J2ME ™: Choose *Tools* | *UI Designer* or click the **Designer** button in the Nokia Developer's Suite for J2ME ™ main window.

The layout of the UI Designer view depends on the type of class that you are editing. When the class is inherited from javax.microedition.lcdui.Form, the Designer uses the Form class mode, and when the class is inherited from javax.microedition.lcdui.game.LayerManager, it uses the Tiled Layer mode. If .flw file is selected then Nokia Screen Flow designer is used.

**Note : Please note that Nokia UI Designer requires javac from JDK 1.4.1 or later**

## 12.1 MIDP User Interface Layout Design

### 12.1.1 UI Designer Layout in the Form Class Mode

*Component palette* shows the available components you can place on a form. You can create additional tabs to the component palette by right clicking the palette and selecting **Customize...**

The actual *UI Designer* pane presenting the layout of the Form is located in the middle. Once you have added items to the form, you can resize them and change their positions on the form with your mouse.

*Resource tree* is displayed in the left pane. The tree represents the components and resource files that have been placed on the form.

Component *Properties* information is displayed in the right pane.

Select the desired UI series from the drop-down list box located below the UI Designer pane. The UI mode is changed when you click the **Apply** button. You can also select the *Viewport* mode to see how the items fits in the screen of the series you have selected. Changing the UI series or showing the viewport does not affect source code.

When working with the standalone version of Nokia Developer's Suite for J2ME ™, you will also have toolbar buttons for creating a new class, opening an existing class, saving, as well as undoing and redoing the last action you performed.

### 12.1.2 Creating a New Form Class

To create a new form class in the standalone version of Nokia Developer's Suite for J2ME ™, click the **New** icon. A form creation dialog appears in the window. Browse for a suitable base directory in the file system, enter a package and a name for the class and select its type (javax.microedition.lcdui.Form). Click **Create** to continue.

In Borland JBuilder, Sun Java Studio and NetBeans, you can create a Form class with the options offered by the respective IDE.

Form properties are displayed in the Properties pane.

| | |
|---|---|
| name | |
| title | Type a name that will be displayed on top of the form class. |

Nokia UI Designer generates the code for creating the desired layout within an init method. The init method is *jbInit* for Borland JBuilder and *ndsInit* for standalone Nokia Developer's Suite for J2ME™ and for Sun Java Studio/NetBeans.

### 12.1.3 Opening an Existing Form Class

To open an existing form class with the standalone Nokia Developer's Suite for J2ME™, click **Open** on the UI Designer's toolbar. Browse for the class in the file system and click **Open** to continue.

In Borland JBuilder: Open a Form class and access the UI Designer view through the *Nokia UI Designer* tab in the JBuilder application window.

In Sun Java Studio or NetBeans: Select a Form class, build the class and then select the *Nokia UI Designer* from a drop-down list. An init method is created at this point if it is does not exist.

### 12.1.4 Adding Components to the Form

Once you have created or opened the form class, you can start adding components to it. You can add components to the form by first selecting an icon and then clicking the form. Component icons are located in the toolbar above the UI Designer form creation pane.

Creates a choice group.

Creates a date field.

Creates a gauge field presenting per cents.

Enables adding images to the form.

Creates an empty space to the form.

Creates a StringItem to the form.

Creates a text field.

Creates a ticker to the form.

Creates a command for an item.

### 12.1.5 Setting Component Properties

Component Properties are displayed in the right pane when the component is selected from the component tree or from the UI Designer area. You can modify component values based on the following principles. Modifications are updated in the form immediately after you exit the field.

**Note : Please note that the default component layout is based on MIDP 1.0 constraints. MIDP 2.0 enables you to modify layouts. A component-specific layout is modified in the dialog that is opened from the layout field of the component's property editor.**

**Note : MIDlet Suite has a default image that cannot be changed in UI Designer. *MIDlet_Icon* can be changed while creating an application package.**

Choice Group

| | |
|---|---|
| name | |
| items | Add choice items in the dialog opened from the right corner of the items field; Enter names for choices, select images and select the desired font settings. This is a UI designer-specific editor for modifying choice group items. |
| choiceType | Select desired choice type: exclusive, multiple or popup |
| fitPolicy | Select how the item name is displayed: default, wrap on (text in several lines if needed) or wrap off (only one line of text even if the name is longer) |
| label | Enter a label for the choice group |
| layout | Select the default layout or modify it in the dialog that opens |
| preferredSize | Enter size (width, height) |

Date Field

| | |
|---|---|
| name | |
| date | Browse for the desired date or enter it in the dialog that is opened from the button in the right corner of the date field |
| inputMode | Select how the date is displayed from the list: date, time or both |

| | |
|---|---|
| label | Enter a label for the date field |
| layout | Select the default layout or modify it in the dialog that opens |
| preferredSize | Enter size (width, height) |

**Gauge**

| | |
|---|---|
| name | |
| interactive | Select a true or false value for interactivity |
| maxValue | Enter maximum value |
| value | Enter value |
| label | Enter a label for the gauge |
| layout | Select the default layout or modify it in the dialog that opens |
| preferredSize | Enter size (width, height) |

**Image**

| | |
|---|---|
| name | |
| appearanceMode | Select how images are displayed: plain, hyperlink or button |
| altText | Enter the text that is shown when the image cannot be displayed |
| image | Browse for an image in the file system. You can use images in PNG, JPG, GIF or BMP format. |
| label | Enter a label for the image |
| layout | Select the default layout or modify it in the dialog that opens |
| preferredSize | Enter size (width, height) |

**Space Area**

| | |
|---|---|
| name | |
| label | Only null is accepted |
| layout | Select the default layout or modify it in the dialog that opens |

| | |
|---|---|
| preferredSize | Enter size (width, height) |

String Item

| | |
|---|---|
| name | |
| appearanceMode | Select how images are displayed: plain, hyperlink or button. Note that the hyperlink or button is shown as a plain string item if no command has been added for it. |
| font | Select face, size and style for the font |
| text | Enter text |
| label | Enter a label for the String Item |
| layout | Select the default layout or modify it in the dialog that opens |
| preferredSize | Enter size (width, height) |

Text Field

| | |
|---|---|
| name | |
| constraints | Select the constraints for the text field: any, e-mail address, numeric, phone number, URL or decimal |
| maxSize | Enter maximum size (number of letters) |
| string | Enter the text that is displayed on the text field |
| label | Enter a label for the Text Field |
| layout | Select the default layout or modify it in the dialog that opens |
| preferredSize | Enter size (width, height) |

Ticker

| | |
|---|---|
| name | |
| string | Enter ticker content. |

Command

Commands can be added to items or to the form; after this they are shown in the Resource tree. The commands are not visualised during the design stage but they are available when you test your design on the emulator or on an actual device.

| | |
|---|---|
| name | |
| label | Enter a label for the Command |
| longLabel | Enter a long label for the Command |
| commandType | Select a command type: screen, back, cancel, ok, help, stop, exit, item |
| priority | Enter a priority number |

### 12.1.6 Creating a Custom Component

MIDP 2.0 enables you to create custom components in order to add functionalities to MIDlets.

Using JavaBeans, you can create and combine reusable software building blocks. A custom item component that conforms to the JavaBeans architecture can be added to the *Component palette*.

Refer to MIDP 2.0 documentation for the javax.microedition.lcdui.CustomItem class that is located in a MIDP 2.0 compatible SDK's JavaDocs and java.beans documentation in http://java.sun.com/j2se/1.4.2/docs/api/java/beans/package-summary.html if you want to develop your own bean component. Your BeanInfo class must implement the BeanInfo interface and it must include an implementation for the following methods in order to work with UI Designer: getPropertyDescriptors(), getIcon(int iconKind).

JavaBeans are packaged and distributed within JAR files. When you have successfully compiled and preverified your bean classes, you should create a single JAR file for the project containing all of the classes and resources needed by the beans in the project.

To add a custom component to your UI Designer component palette, right-click the component toolbar. Select **Customize...** from the pop-up menu. Custom components and component pages can be added in the *Edit Component Palette* dialog. You can also remove and organise components in the same dialog.

To create a new component page, select **Add Component Page** and enter a name for the component page. To add a component to the page, select **Add Component**. Browse for a suitable JAR file containing the custom component and enter additional information if needed.

Removing a custom component from a form does not remove "import examples.customitems.*component_name*" from the source code.

When you add a custom item to your MIDlet project, please make sure that the custom item's preverified .class file exists in the MIDlet's application package.

### 12.1.7 Custom Component Example

You may try out and use an example custom component that illustrates traffic lights. Select trafficlights.jar in the *Add component* dialog located in the folder NDS_INSTALL_DIR\examples\TrafficLight.

| | |
|---|---|
| name | |
| alignment | Select the position of the traffic lights: horizontal or vertical |
| lights | Select active lights with the custom bean editor |
| shape | Select the shape of the traffic lights: round or square |
| label | Enter a label for the lights |
| layout | Select the default layout or modify it in the dialog that opens |
| preferredSize | Enter size (width, height) |

You must add the preverified TrafficLight.class located in NDS_INSTALL_DIR\examples\ TrafficLight.jar to your project and make sure that it is placed in the MIDlet's application package.

In Sun Java Studio you can mount the TrafficLight.jar and copy the TrafficLight.class to output path \examples\customitems.

### 12.1.8 Trying It Out on an Emulator

In order to try out the form class you have created on an emulator, you must create a MIDlet that displays the Form. You will find a simple project with a MIDlet class and an empty form in NDS_INSTALL_DIR\examples\FormMIDlet.

In the standalone Nokia Developer's Suite for J2ME ™ you can add items to MyForm.java. The MyForm class is compiled automatically when you save the file. You can then recreate the FormMIDlet's application package and test the design on an emulator.

In Sun Java Studio, NetBeans and in Borland JBuilder you can create the application package using the tools provided by the IDE or you can select the *New Application Package...* option from the Nokia Developer's Suite for J2MD™ menu. Make sure that the needed classes and resources are included in the application package.

## 12.2 Tiled Layer Design

### 12.2.1 UI Designer Layout in the Layer Manager Mode

The left pane is meant for **layer design**.

The upper right pane displays the **layers** that are added to the layer manager.

The lower right pane displays the **tiles** that can be used in a layer.

You can also select *Viewport* mode to see how the items fits on the screen of the selected phone series. If you have selected *Custom*, you may customise the width and height of the viewport. The mode is only for viewing. Layers are not editable when a viewport is displayed.

A **Grid** can be shown over the layer design area to help you visualise the size of a single tile as well as the width and height of the selected layer.

When using Nokia Developer's Suite for J2ME ™ as a standalone product, you will also have toolbar buttons for creating a new class, opening an existing class, saving, as well as undoing and redoing the last action you performed. The toolbar also offers you tools for modifying layers.

### 12.2.2 Creating a Layer Manager Class

In the standalone Nokia Developer's Suite for J2ME, you can create a new LayerManager class by clicking the **New** button. A form creation dialog will appear in the window. Browse for a suitable base dir in the file system, fill in a package and a name for the class, and select class type (javax.microedition.lcdui.game.LayerManager). Click **Create** to continue.

In Borland JBuilder, Sun Java Studio, and NetBeans, you can create a LayerManager class with the options offered by the respective IDE.

### 12.2.3 Opening an Existing Layer Manager Class

To open an existing LayerManager class with the standalone Nokia Developer's Suite for J2ME™, click **Open** on the UI Designer's toolbar. Browse for the class in the file system and click **Open** to continue.

If tiled layers and palette are not displayed when you open a LayerManager class, please ensure that the images are located in the correct path.

**Note : Please note that images should be located in the src\-folder or in its sub folder.**

In Borland JBuilder: Open a LayerManager class and access the UI Designer view through the *Nokia UI Designer* tab in the JBuilder application window.

In Sun Java Studio or NetBeans: Select a Form class, build the class and then select the *Nokia UI Designer* from the drop-down list. An init method is created at this point if it is does not exist.

### 12.2.4 Creating a New Layer

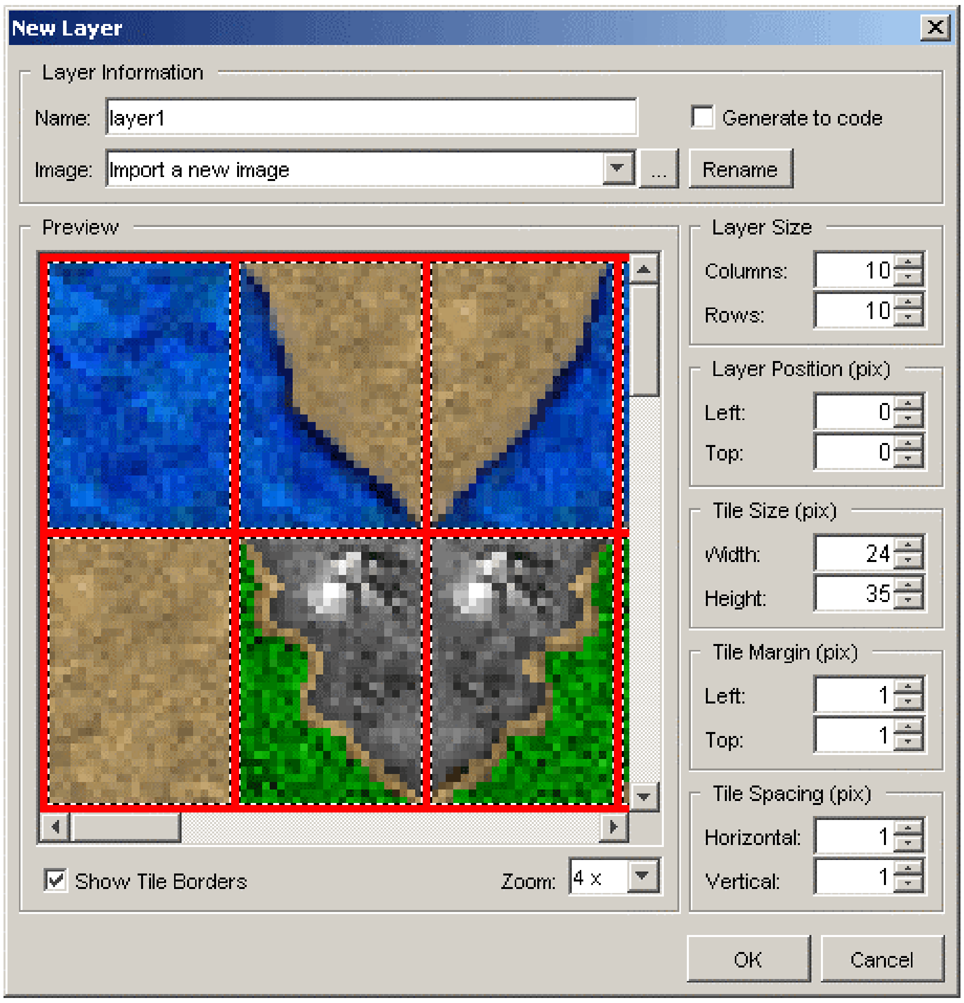
Once you have opened a layer manager class, you can create layers. Create a new layer by selecting **Add** in the layer pane.

Enter a name for the layer and use *Layer Size* options to specify how many columns and rows the layer will have (Figure 34). You may also want to modify the *Layer Position*.

Then select a PNG image that contains the tiles for the layer. You can either Import a new image or use a previously imported image as the basis of your LayerManager class. Existing image resources are listed in the *Image* drop-down box. By default, all new tile palette resources are named after the layer you are creating. Click **Rename**, if you want to enter a different name for the tile palette image.

When you are importing a new image you can use example_tile_palette.png that is located in the folder NDS_INSTALL_DIR\examples\Spaceworld\src\images.

Specify *Tile Size* to match the tile size of the selected image. *Zoom* and *Show Tile Borders* are useful when you are changing the tile size. Use the *Layer Margin* and *Tile Spacing* options to remove any possible margins from the original PNG tile image.


New Layer

Layer Information

Name: layer1

☐ Generate to code

Image: Import a new image

Preview

☑ Show Tile Borders

Zoom: 4 x

Layer Size

Columns: 10

Rows: 10

Layer Position (pix)

Left: 0

Top: 0

Tile Size (pix)

Width: 24

Height: 35

Tile Margin (pix)

Left: 1

Top: 1

Tile Spacing (pix)

Horizontal: 1

Vertical: 1

Figure 34: New Layer Manager

By default, layer data is generated in a resource file with the extension ".tlr". Select the *Generate to code* check box if you do not want to use binary format. If you use binary format, make sure that the .tlr files have been added as resources when you create the application package.

### 12.2.5 Modifying a Layer

To edit layer properties, click **Edit** in the layer pane. Edit information in the dialog that opens. You can remove a layer by selecting it and clicking the **Remove** button.

To modify layer layout, select a layer and modify it with following tools:

*Add* single tiles to the layer.

*Fill* an area with the selected tile.

*Select* an area. You can move the selected area by placing the mouse pointer on the border of the area. You can also right-click the selected area and select *Copy selection as a new layer*.

Use the right mouse button to remove tiles from a layer.

To change the order of the layers, drag and drop layers on the layer list. The uppermost listed layer is displayed on top of the layer design. You can show and hide unselected layers by clicking the button on the left side of the layer name.

If you want to add new tiles to a layer's tile palette, you must modify the tile palette image of the respective layer. Tile palette images are located in the *src* folder of your project. Simply create an extra row for the new tiles in the resource image with an image editing application. The changes to the image resource will take effect the next time you open your LayerManager class.

### 12.2.6 Trying It Out on an Emulator

In order to try out the LayerManager class you have created on an emulator, you must create a MIDlet that includes a game engine that uses the LayerManager. You will find a simple project with a MIDlet class, game engine and LayerManager in NDS_INSTALL_DIR\examples\Spaceworld.

You can edit the Spaceworld MIDlet's layers by opening the spaceworld.java file with UI Designer. The Spaceworld class is compiled automatically when you save the file. You can then recreate the Spaceworld's application package and test the application on an emulator.

## 12.3 Screen Flow Design

### 12.3.1 Screen Flow Designer Layout

Flow designer layout is divided into a flow diagram *resource tree* on the left and a *flow diagram designer* on the right. The tree represents the components and resource files that have been placed on the diagram. *Component palette* shows the available components you can place in a diagram.

### 12.3.2 Opening a Screen Flow Diagram

You can start creating a flow diagram by selecting *New* . Enter class information and select *Screen Flow Designer XML File* as Class Type. Click **Next** to continue. Define the required Flow Designer information. *Command handler package* and *Command handler name* are used to define a command handler. A command handler is created for each diagram. If you do not define the command handler package, you cannot use displayables with a package structure. Complete the flow diagram creation process by clicking **Create**. The created command handler appears in the tree structure.

To open an existing flow design, select *Open* and then select a flow file (.flw) from the file system.

In Sun Java Studio or NetBeans: To create a new screen flow diagram, select *File* | *New* . Choose *MIDP* | *Nokia UI Designer* | *Empty Flow Diagram* as the template in the *New Wizard*. Continue by clicking **Next**. Enter a name for the flow diagram and select the location. To complete the flow creation process, click **Finish**.

In Borland JBuilder: To create a new screen flow diagram, select *File* | *New File*. In the *Create New File* dialog, select *flw* as the file type and enter other information. Select the UI Designer tab from the current editor pane. You can switch between the editor, UI Designer and History panes by clicking the tabs during development.

In Eclipse: see Using Nokia Developer's Suite from Eclipse - UI Designer

### 12.3.3 Modifying a Screen Flow Diagram

Once you have opened a new or an existing flow diagram, you can add components to the diagram. The following components are available for flow diagrams:

Selects and drags items.

Adds a MIDlet to the diagram.

Adds a form to the diagram.

Adds a list to the diagram.

Adds a canvas to the diagram.

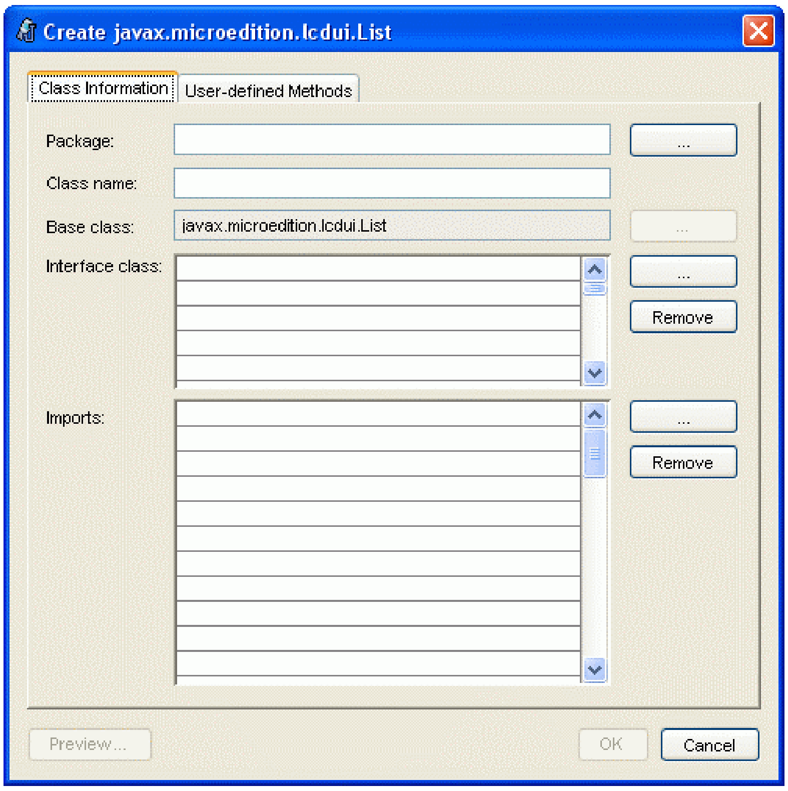
Adds an alert to the diagram.

Adds a textbox to the diagram.

Aligns selected items vertically.

Aligns selected items horizontally.

Select a component icon from the component bar and click the flow diagram creation pane. If you select a MIDlet, a form, a list, a canvas, an alert or a textbox, a dialog (see Figure 35) opens after you have clicked the diagram pane. Define the required component information and click **OK**. Accept the class source file generation by clicking **OK** in the pop-up dialog. The component appears in the diagram pane and the tree structure.



Base class: javax.microedition.lcdui.List

Figure 35: A dialog for creating a component

Only one MIDlet can be included in a diagram. You can create several MIDlets in the tree structure but you can only place one in the flow diagram at a time. You can drag and drop components from the tree structure to the diagram pane. Once you have added components to the diagram pane, you can arrange them vertically and horizontally. Select components that you want to arrange and click the *Align vertically* or the *Align horizontally* button in the component bar.

You can connect components with commands in the diagram pane. A MIDlet class starts the flow progression and it can be only connected to one other component. To create a connection, hold the mouse button down (do not click) on an unselected component and release the mouse button on another component. While dragging, an arrow follows the movements of the mouse. When you release the mouse button on a component, a command arrow is created between the components. You can also change the arrow shape. If you select the arrow and hold *Shift* down on the keyboard, you can create focus points by clicking the arrow. When the arrow is selected, the points are displayed and you can drag the arrow to the desired shape.

To rename a component or an arrow, right-click the item and select *Rename* from the pop-up menu. Enter the new name in the dialog. If you want to modify a component, right-click the component and select *Edit*. If you selected a form, the MIDP user interface layout designer (Form Designer) opens. You can modify the form as usual using UI Designer tools. To return to the Flow Designer, click the **Back** button in the upper right corner of the layout designer pane. To modify properties of other components, an *Edit Component* dialog opens and allows you to modify component information. The modifications are saved in the source code after the dialog is closed.

To remove a component from the diagram pane, right-click the component and select *Remove* from the pop-up menu. The component is removed from the diagram pane but it remains in the tree structure. If you want to delete a component file permanently from the diagram and the tree structure, right-click the item in the tree structure and select Delete from the pop-up menu.

Save the created flow diagram by clicking the *Save* icon. The save action compiles the screen flow diagram when you are using a standalone version of Nokia Developer's Suite 3.0 for J2ME™.

### 12.3.4 Trying It Out on an Emulator

You can run the compiled flow diagram class with the NDS *Start Emulators* functionality. Select the MIDlet class in the *Application* field and continue as usual when using the *Start Emulators* functionality.

If you are using Nokia Developer's Suite 3.0 for J2ME™ with an IDE integration, compile the screen flow classes and run the screen flow designer class as usual with the tools provided by the IDE. If you have used resources such as images in the application developed with the Screen Flow designer, you need to create an application package (see Chapter 7) before running the application in an emulator.

# 13 DRM Editor

Digital Rights Management (DRM) is a system of media content types and cell phone behaviour that allows content providers to control how a phone user uses and re-distributes the content, such as MIDlets. This control is achieved by pairing the content with rights that control how often and for how long a phone user can use the content on the cell phone. For example, a content provider can permit a phone user to receive and execute a MIDlet once as a preview, then allow the phone user to purchase the rights to play the game indefinitely.

DRM provides three message types that protect a MIDlet with varying levels of security:

| DRM Message Type | Description |
|---|---|
| Forward Lock | A single message that allows a phone user to use the MIDlet without limitation in a single device (typically, a cell phone), but prevents the user from forwarding the MIDlet to another device. |
| Combined Delivery | A single message that allows a phone user to use the MIDlet with some limitations in a single device, but prevents the user from forwarding the MIDlet to another device. |
| Separate Delivery | A pair of messages (one with a MIDlet, one with the rights to the MIDlet) that allows a phone user to use the MIDlet with some limitations in a single device if the user has received the rights to use the MIDlet. The phone user can forward the MIDlet, but not the rights, to another user. Information on obtaining rights to use the MIDlet is embedded in the MIDlet file so the new receiver of the MIDlet can obtain the rights to use the MIDlet. |

The DRM Editor lets you:

- Select and prepare a JAR file containing a MIDlet to be wrapped in a digital rights message. The DRM message works with a copy of the original MIDlet so that the original MIDlet file is never altered.

- Create and prepare the rights that allow the phone user to use the MIDlet.
- Edit the headers sent with the messages that can affect how a device handles the MIDlet.
- Modify the JAD file so that it points to the DRM message in which the MIDlet is wrapped.

How the MIDlet provider chooses to deliver the MIDlet and its rights depends on several factors, including whether the MIDlet has more than one set of rights available and how the MIDlet will be distributed. No MIDlet within a DRM message can be used without its associated rights.

## 13.1 DRM Message Creation

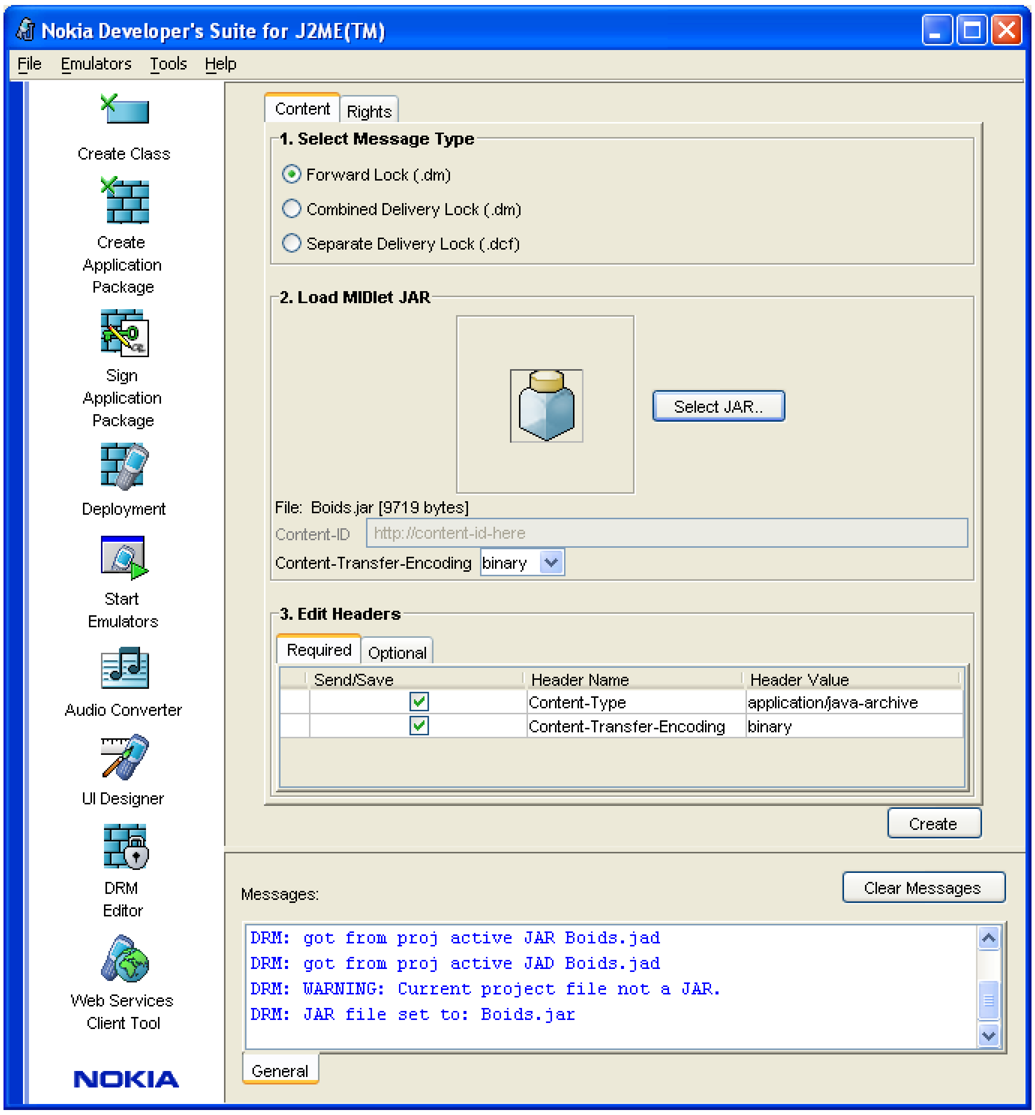
To access the DRM Editor in NDS for J2ME, select *Tools* | *DRM Editor* or click the **DRM Editor** icon on the left panel. If you are working within an IDE, select *Tools* | *Nokia Developer's Suite for J2ME* |*DRM Editor*.

The DRM window is divided into two tabs: **Content**, where you can select the JAR file you are going to wrap in the DRM message, and **Rights**, where you can select the rights you want to associate with the media. When you select **Forward Lock**, the Rights tab lists no rights. When you select **Combined Delivery** or **Separate Delivery**, the Rights tab lists the rights you can set. You can create a DRM message in four basic steps. The complexity of each step depends on what type of DRM message you are creating. Some fields that apply to one type of message do not apply to others. The basic steps are:

1. Select the type of message (Forward Lock, Combined Delivery, Separate Delivery).
2. Select the JAR file. (Browse to the location of the file that contains the MIDlet to which you are going to assign rights.)
3. Confirm the required headers and edit the optional headers, if necessary.
4. Specify the rights. (Not applicable to Forward Lock messages).

### 13.1.1 Creating a Forward Lock Message

A Forward Lock message contains a JAR file that can be used without limitations on the device that receives it but cannot be forwarded to any other device. When you prepare a Forward Lock message, the message is saved in a file that has a *.dm* extension. A *.dm* file uses a standard multipart format with the MIDlet as its only part. The MIDlet of a Forward Lock message is always unencrypted.



1. Select Message Type

(● Forward Lock (.dm)) ○ Combined Delivery Lock (.dm) ○ Separate Delivery Lock (.dcf)

2. Load MIDlet JAR

File: Boids.jar [9719 bytes]

Content-ID: http://content-id-here

Content-Transfer-Encoding: binary

3. Edit Headers

| Send/Save | Header Name | Header Value |
|---|---|---|
| ✓ | Content-Type | application/java-archive |
| ✓ | Content-Transfer-Encoding | binary |

Messages:

```
DRM: got from proj active JAR Boids.jad
DRM: got from proj active JAD Boids.jad
DRM: WARNING: Current project file not a JAR.
DRM: JAR file set to: Boids.jar
```

Figure 36: Creating a Forward Lock Message

To prepare a Forward Lock message:

1. In the DRM Editor under Select Message Type, select Forward Lock (Figure 36). The Rights tab will not list rights because a Forward Lock message does not contain a rights object. A Forward Lock message has implied unlimited rights associated with its MIDlet.
2. Under Load MIDlet JAR, click Select JAR to browse to the JAR file you want to wrap in the message and click Open.
3. Under Content-Transfer-Encoding, select one of the following:

- **Binary** – Leaves the MIDlet unencoded. (Recommended for most MIDlets.)
- **Base64** – Encodes the MIDlet by converting it from binary to 7-bit (ASCII) using a Base64 encoding scheme. (Recommended for use with mail programs that do not handle binary message content.)

4. Under Edit Headers, check that the:

- **Content-Type** value reflects the content type of the file. If this entry is incorrect, you can edit the header. The entry may be incorrect if the original JAR file did not have the correct suffix (*.jar*).
- **Content-Transfer-Encoding** value matches the value you selected for **Content-Transfer-Encoding**.

5. Click Create.
6. Provide a name for the DRM message and click Save. It is recommended that you name the DRM message *MyMIDlet_drm.dm* although the system suggests a file name without the *_drm* appendix.
7. When asked "Do you want to associate a JAD file with this DRM file," click:

- **Yes** to select the JAD file you want. A copy of this file is edited to point to the DRM message as the source of the JAR file and to update the size of the file. The name of the JAD file is also modified so that the new JAD file does not overwrite the original JAD file. The original name of the JAD file is appended with an *_drm* extension. For example, *MyMIDlet.jad* is renamed *MyMIDlet_drm.jad*.
- **No** to not associate any JAD file with the DRM message.

Information about the creation of the message is displayed under **Messages**. To clear this area, click **Clear Messages**.

**Note : Make sure that the created DRM message and the associated JAD file have the same name, for example MyMIDlet_drm.dm and MyMIDlet_drm.jad.**

**This must be ensured because real devices check that the imported JAD and JAR files have the same name. Likewise, it must be checked that the DRM protected JAD and DM files are named similarly.**

## 13.2 Combined Delivery Message Creation

A Combined Delivery message delivers the JAR file and the rights to use the MIDlet in the JAR file to the receiving device in a single file. For example, a phone user who receives a game in a Combined Delivery message can play the game in accordance with the rights specified in the file – say, five times over a period of one month. The phone user cannot forward the message to another phone user because a Combined Delivery message contains rights and phone users are not allowed to forward rights.

When you prepare a Combined Delivery message, you select the JAR file and set the rights for the MIDlet. The file is saved with a *.dm* extension. The MIDlet is always unencrypted. Generally, Combined Delivery messages are used when you have a MIDlet that always ships with a specific set of rights, and when you do not require the MIDlet to be encrypted for security.

To create a Combined Delivery message:

1. Under Select Message Type, select Combined Delivery. The Rights tab now contains the rights you can later set.
2. Under Load MIDlet JAR, click Select JAR to browse to the JAR file you want to wrap in the message and click Open.

3. Under Content-ID, enter an identifier for the MIDlet. This identifier must be unique because it links the object to the rights.
4. Under Content-Transfer-Encoding, select one of the following:

- **Binary** – Leaves the MIDlet unencoded. (Recommended for most MIDlets.)
- **Base64** – Encodes the MIDlet by converting it from binary to 7-bit (ASCII) using a Base64 encoding scheme. (Recommended for use with mail programs that do not have binary conversion schemes.)

5. Under **Edit Headers**, check that the headers listed under the **Required** tab meet the following criteria:

- The **Content-Type** value reflects the content type of the file. If this entry is incorrect, you can edit the header. The entry may be incorrect if the MIDlet file does not have a *.jar* suffix.
- The **Content-ID** value matches the value you entered under **Content-ID** .
- The **Content-Transfer-Encoding** value matches the value you selected for **Content-Transfer-Encoding**

6. Click the **Rights** tab, and enter the rights you want to associate with the MIDlet. (See *Working with Digital Rights*.)
7. Click **Create**.
8. Provide a name for the DRM message and click **Save**.
9. When asked “**Do you want to use an associated JAD file with this DRM file**,” click:

- **Yes** to select the JAD file you want. A copy of this file is edited to point to the DRM message as the source of the JAR file and to update the size of the file. The name of the JAD file is also modified so that the new JAD file does not overwrite the original JAD file. The original name of the JAD file is appended with an *_drm* extension. For example, *MyMIDlet.jad* is renamed *MyMIDlet_drm.jad*.
- **No** to not associate any JAD file with the DRM message.

Information about the creation of the message is displayed under **Messages**. To clear this area, click **Clear Messages**.

## 13.3 Separate Delivery Message Creation

A Separate Delivery message is composed of two discrete files that work together to form a DRM message. One file contains the encrypted MIDlet JAR file. The other file contains a key to decrypt the MIDlet and the rights to use it. The phone user can only use the MIDlet on the device for which he or she has acquired the rights to use the MIDlet.

The separation of the MIDlet and the rights permits the phone user to forward the MIDlet to another phone user who can then acquire (purchase) the rights from the MIDlet provider. The initial phone user cannot forward the rights. Each subsequent receiver of the MIDlet must acquire new rights from the MIDlet provider to use the MIDlet.

The Separate Delivery message has only one file type for the MIDlet (*.dcf* ), but the rights in a Separate Delivery message are saved in two formats:

- Unencoded (*.dr*) – Unencoded *.dr* preserves the rights in an XML format.
- Encoded (*.drc*) – This format encodes the rights with WBWML. Use this format when you know you will be dealing with a gateway that uses a binary format.

To create a Separate Delivery message:

1. Under **Select Message Type**, select **Separate Delivery.** The **Rights** tab now contains the rights you can later set.
2. Under **Load MIDlet JAR**, click **Select JAR** to browse to the JAR file you want to wrap in the message.
3. Under **Content-ID**, enter an identifier for the MIDlet. This identifier must be unique because it links the object to the rights.
4. Under **Edit Headers**, edit the headers listed under the **Optional** tab. Only Separate Delivery messages have optional headers that you can edit.
5. Click the **Rights** tab and enter the rights you want to associate with the MIDlet. (See *Working with Digital Rights*.)
6. Click **Create**.
7. Provide the DRM Message with a name and click **Save**. The *.dcf*, *.dr.* and *.drc* files are saved.
8. When asked “Do you want to associate a JAD file with this DRM file,” click:

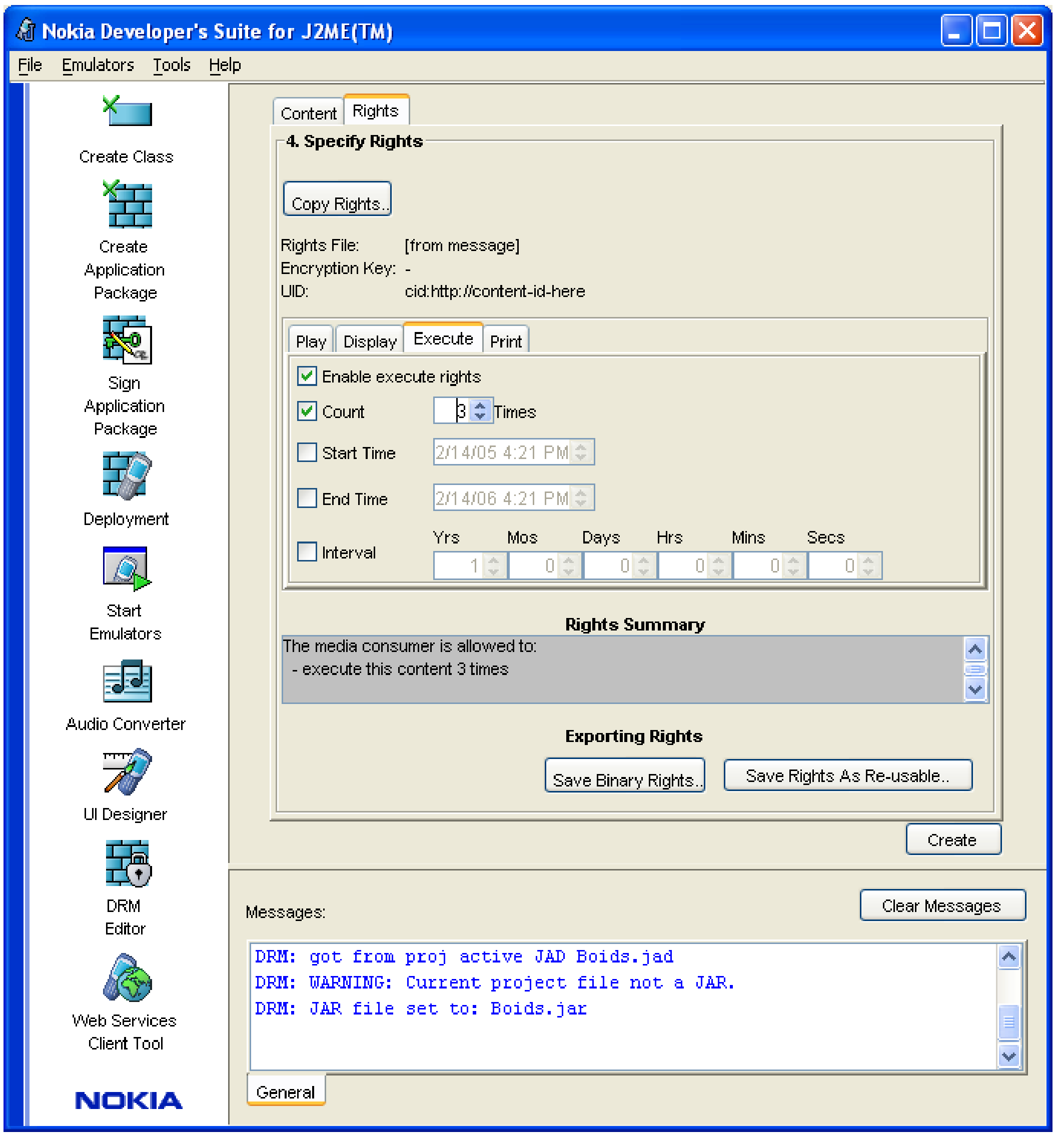
- **Yes** to select the JAD file you want. A copy of this file is edited to point to the DRM message as the source of the JAR file and to update the size of the file. The name of the JAD file is also modified so that the new JAD file does not overwrite the original JAD file. The original name of the JAD file is appended with an *_drm* extension. For example, *MyMIDlet.jad* is renamed *MyMIDlet_drm.jad*.
- **No** to not associate any JAD file with the DRM message.

Information about the creation of the message is displayed under **Messages**. To clear this area, click **Clear Messages**.

## 13.4 Working With Digital Rights

You specify digital rights when you create a Combined Delivery or Separate Delivery message. A Forward Lock message has implied unlimited rights associated with its MIDlet, so you do not need to specify rights.

To create digital rights, click the **Rights** tab (Figure 37).



Rights File: [from message]
Encryption Key: -
UID: cid:http://content-id-here

☑ Enable execute rights
☑ Count 3 Times
☐ Start Time 2/14/05 4:21 PM
☐ End Time 2/14/06 4:21 PM
☐ Interval Yrs 1 Mos 0 Days 0 Hrs 0 Mins 0 Secs 0

Rights Summary
The media consumer is allowed to:
- execute this content 3 times

Exporting Rights

Messages:

```
DRM: got from proj active JAD Boids.jad
DRM: WARNING: Current project file not a JAR.
DRM: JAR file set to: Boids.jar
```

Figure 37: Defining Rights for a Combined Delivery Lock Message

Digital rights have the following parts:

Permissions – General rights that fall into these categories: Play, Execute, Print, and Display.

Constraints – Limitations applied to a permission. For example, you can constrain the Execute permission to a single execution of the MIDlet for ten minutes. Unless you set constraints, a permission is unlimited.

### 13.4.1 How MIDlets and Rights Are Matched

A MIDlet and its rights are linked internally with these identifiers:

| Identifier | Definition | Used in |
|---|---|---|

| | | |
|---|---|---|
| Content-ID | A string that uniquely identifies the DRM MIDlet. | Combined Delivery<br><br>Separate Delivery |
| UID | A matching ID string that matches the MIDlet and appears in the rights object. | Combined Delivery<br><br>Separate Delivery |
| Content-ID header | A required header whose value is taken from the Content-ID field. | Combined Delivery |
| Encryption key | A unique key that unlocks its associated MIDlet for use. This key is automatically generated by the DRM Message Editor and is linked to the UID of a file. | Separate Delivery |

When a device receives several DRM messages, each with MIDlets and rights, the rights must be correctly linked with the specific MIDlet for which it is meant. Preserving this association is especially important in Separate Delivery messages where the MIDlet and its rights are sent separately.

### 13.4.2 Setting Digital Rights

You can set these permissions only when you create a Combined Delivery or a Separate Delivery message:

- **Play** – Controls permissions for audio and video files (for example, *.mp3* ).
- **Execute** – Controls permissions for executables (for example, *.jar* or *.exe* ).
- **Print** – Controls permissions for printing (for example, *.jpeg*, or *.gif*.), if the cell phone is designed

  to perform wireless printing.
- **Display** – Controls permissions for viewing images (for example, *.jpeg* or *.gif* ).

Each permission is set in an identical way. The rights you set are typically dictated by the MIDlet and the way you want the phone user to use it.

SDKs and phones only apply the Execute permission to MIDlets. All other permissions are ignored for MIDlets.

**Note : Enabling a Permission**

If you do not enable a permission, the phone user will have no rights to that permission. To enable permissions, click the tab of the permission you want to enable (**Play**, **Execute**, **Print**, or **Display**).

To set unlimited rights within that permission, check **Enable <permission> rights**.

**Note : Applying Constraints to a Permission**

Constraints allow the MIDlet to be executed with permission, but only under the terms you specify.

To apply constraints to an enabled permission:

1. Make sure **Enable <permission> rights** is checked.
2. Check the constraints you want to apply and enter a value for each one:

- **Count** – Enter the number of times the MIDlet can be used.
- **Start Time** – Enter the date and time after which the MIDlet can be used.
- **End Time** – Enter the date and time after which the MIDlet can no longer be used.
- **Interval** – E nter the maximum period of time during which a MIDlet can be used, beginning from the first use.

A summary of the permissions and constraints appears under **Rights Summary** as you enter the rights.

**Note : Exporting Digital Rights**

When you have finished setting up permissions and their constraints, you can export the rights without the MIDlet in the following ways:

| Export Option | What the option does |
|---|---|
| **Save Binary Rights** | Creates a WBXML-encoded digital rights file that is associated with the MIDlet. Use this option when you work with a push server that cannot encode Rights Expression Language files (*.dr*). The suffix of this file type is *.drc*. |
| **Save as Re-usable Rights** | Saves the digital rights without any UID or Encryption key. This option allows you to re-use the rights with different MIDlets. |

### 13.4.3 Creating Re-usable Rights

The difference between a rights file that is associated with a MIDlet and a re-usable rights file is that a re-usable rights file does not have a value for the UID, does not have a value for the Encryption key, and can be applied to more than one MIDlet after the UID and Encryption key are generated.

To create re-usable rights in the DRM window:

1. Select **Combined Delivery** or **Separate Delivery** to access the Rights tab.
2. Enter the permissions (**Print**, **Execute**, **Play**, **Display**) with the constraints (**Count**, **Start Time**, **End Time**, **Interval**) you want.

3. Click **Save as Re-usable Rights** and navigate to the location where you want to save the re-usable rights file.

The digital rights file that is saved contains all the rights you entered, but omits the value for the UID and the Encryption key.

### 13.4.4 Copying Digital Rights

In the DRM Message Editor, you can import the rights from any *.dr* file, including associated rights files and re-usable rights files. When you import digital rights, permissions and constraints are copied without the UID or Encryption key and the original rights are not changed.

To copy digital rights, click **Copy Rights** on the **Rights** tab.

# 14 J2ME™ Web Services (JSR-172) Client Tool

J2ME™ Web Services Client Tool is a development tool that generates Java stub classes from WSDL files. These classes provide a simple interface for accessing web services and make it easier for developers to create mobile web services clients.

Web Services Description Language (WSDL) is an XML format that defines Web Services and how to access them. WSDL definition define communication endpoints (server locations) that process messages, ports that describe operations and messages, messages that are input and output values of ports. Also operations and messages that are attached using bindings to a network accessible format and type definitions that are used in the input and output messages (remote interface) are defined by the WSDL definition.

Web Services Client Tool supports Web Service development using an SDK that includes Web Services API (JSR-172) [8] (please follow SDK updates on the Forum Nokia Web site, http://www.forum.nokia.com/tools). The tool supports WSDL, which complies with the WS-I Basic Profile 1.0 specification [9], but with certain restrictions.

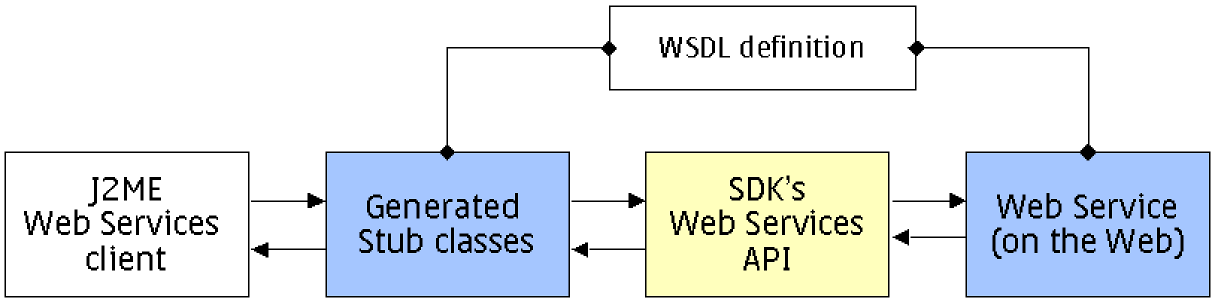
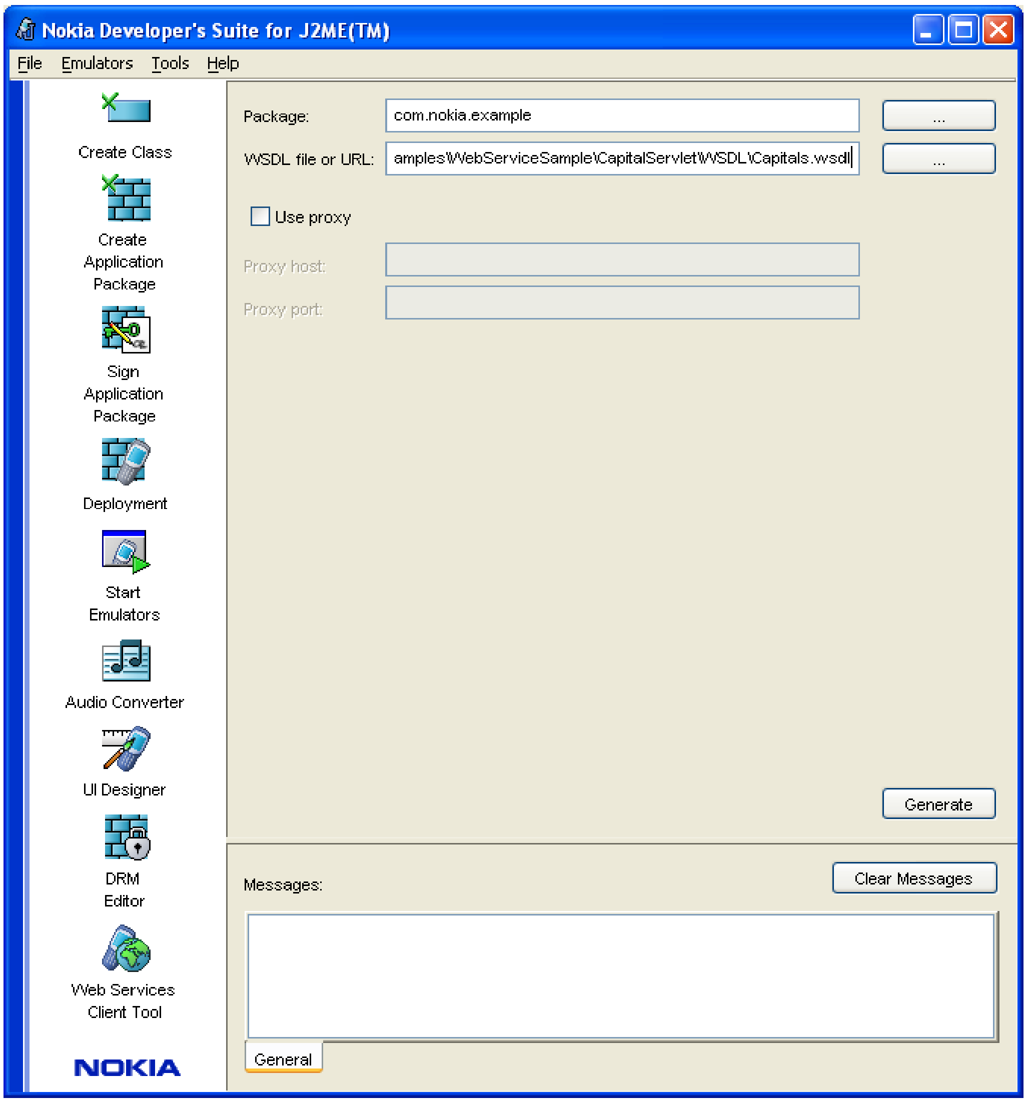


Figure 38: J2ME Web Services Development

**Note : Web Services Client Tool only supports the xsd namespace in WSDL documents.**

Standalone Nokia Developer's Suite 3.0 for J2ME™: Choose *Tools* | *Web Services Client Tool* or click the **Web Services Client Tool** button in the Nokia Developer's Suite 3.0 for J2ME™ main window.



Package: com.nokia.example

WSDL file or URL: amples\WebServiceSample\CapitalServlet\WSDL\Capitals.wsdl

Use proxy

Proxy host:

Proxy port:

Messages:

Figure 39: Web Services Client Tool

- **Package –** Define or browse package for client stub classes. With the **Browse** button you get list of directories under the working directory defined in *File* | *Preferences*.
- **WSDL file or URL -** Define or browse the WSDL input file for which you wish to create a client stubs. You can also specify a file through an HTTP address.
- **Use proxy -** Select this option if you are using a HTTP proxy and you are using a remote WSDL input file
- **Proxy host -** Define HTTP proxy host name
- **Proxy port -** Define HTTP proxy port

The **Generate** button generates and saves the generated stub classes. The default location for generating the stub classes is under the working directory defined in *File* | *Preferences*.

## 14.1 Capital Service example Web Service

Capital Service is a simple servlet program that implements a Web Services. Using the provided WSDL file you can create a client that queries this Web Service for a nation's capital. The example works with a locally installed Tomcat Servlet container (http://127.0.0.1 port 8080).

Capital Web Service is available on the Forum Nokia server: http://194.137.80.38/CapitalService/ and the Capitals.wsdl: http://194.137.80.38/CapitalService/Capitals?wsdl. You can use the Capital Web Service from Forum Nokia server or you can install it to your local server.

### 14.1.1 Capital Service servlet installation to local server:

1. Unpack or install Tomcat 5.0 or newer to any directory.
2. Copy CapitalService.war from <NDS_INSTALL_DIR>/examples/WebServiceSample/CapitalServlet/release/ to <TOMCAT_INSTALL_DIR>/webapps
3. Start Tomcat with <TOMCAT_INSTALL_DIR>/bin/startup.bat

After Tomcat has started you can see that he installation was successful by directing your browser to http://localhost:8080/CapitalService/

The response should be: HTTP Status 405 - HTTP method GET is not supported by this URL

This means that the CapitalService has been installed byt GET method is not supported. This is not a problem, Web Services use POST method.

### 14.1.2 Running Capital Service Client:

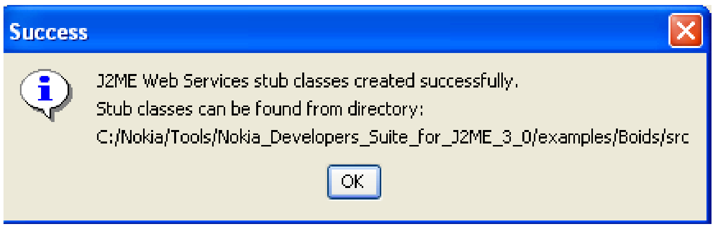
1. Make sure you have Capital Servlet running on http://localhost:8080/CapitalService/
2. Run the Capital MIDlet from <NDS_INSTALL_DIR>/examples/WebServiceSample/CapitalClient/bin using SDK that includes Web Services API (JSR-172).
3. When the Capital MIDlet starts, enter a nation and press Get capital command button. When requested for network access, anwer Yes.

### 14.1.3 Creating own Capital Service Client:

Using Capitals.wsdl file, create interface for a client that calls the Web Service. Use the created interface to query a nation's capital from the Web Service. The parameter given to method String getCapital(String nation) can for example be "Finland", "United Kingdom", "France", etc.

1. Start Nokia Developer's Suite 3.0 for J2ME™ for J2ME ™
2. Choose *File* | *Preferences...* and set the working directory to <NDS_INSTALL_DIR>/examples/WebServiceSample/CapitalClient
3. Choose Tools | Web Services Client Tool.
4. Define package e.g. com.nokia.example for stub classes.

5. Browse to the Capitals.wsdl file in NDS_INSTALL_DIR/examples/WebServiceSample/CapitalServlet/WSDL or use the following URL http://194.137.80.38/CapitalService/Capitals?wsdl
6. If you use a remote WSDL input file and you use proxy, choose to Use proxy and define proxy host and proxy port.
7. Click **Generate**.


Success

J2ME Web Services stub classes created successfully.
Stub classes can be found from directory:
C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME_3_0/examples/Boids/src

Figure 40: Result of the successful generation

Two stub classes: capitalBinding_Stub.java and capitalPortType.java are generated to the <NDS_INSTALL_DIR>/examples/WebServiceSample/CapitalClient/src/com/nokia/example.

8. Build stub classes using SDK that includes Web Services API (JSR-172).
9. Run the CapitalClient MIDlet using SDK that includes Web Services API.
10. When the Capital MIDlet starts, enter a nation and press Get capital command button. When requested for network access, answer Yes.

## 15 Nokia Developer's Suite for J2ME™ Ant Tasks

Nokia Developer's Suite for J2ME™ provides you with the following Ant tasks:

*Nokia Developer's Suite for J2ME™ Ant Tasks*

| Task | Description |
|---|---|
| NdsJ2meJavac | Compiles MIDlet .java files using the boot class path of an emulator device. |
| NdsJ2mePackage | Packages a MIDlet application in a JAD/JAR file pair. |
| NdsJ2mePpJar | Packages a Personal Profile application in a JAR file. |
| NdsJ2meRun | Executes a MIDlet .class, a JAD/JAR package or a Personal Profile JAR in an emulator device. |
| NdsJ2meConvertAudio | Converts MIDI and XML files to other audio formats (OTA, bytes or XML). |
| NdsJ2meSign | Signs a MIDlet. |

| | |
|---|---|
| NdsJ2meDeployDevice | Deploys applications to actual devices. |
| NdsJ2meDeployFtp | Deploys applications to FTP servers. |

## 15.1 Getting started

The tasks use features from Nokia Developer's Suite for J2ME™. Therefore, before using the Ant tasks, you must set the home directory of Nokia Developer's Suite 3.0 for J2ME™ to the build.xml file.

When the tasks are executed, they use a device from an emulator SDK installed in Nokia Developer's 3.0 Suite for J2ME™. In the build script, you must define a property that sets the project emulator.

*Mandatory properties*

| Property | Description |
|---|---|
| ndsj2me.home | Home directory of the Nokia Developer's Suite for J2ME™ |
| ndsj2me.emulator | Project emulator/SDK. A device from the project emulator will be used by the tasks. |

The project emulator may contain more than one device. In these cases you can also set the project device. See below for more information.

### 15.1.1 Setting the Nokia Developer's Suite 3.0 for J2ME™ home and defining the tasks

Before you can use the tasks they must be made available to Ant. This consists of setting the 'ndsj2me.home' property, adding Nokia Developer's Suite 3.0 for J2ME™ JARs to a path element and then using this path to define the tasks. Add the following lines to your Ant build script and modify the 'ndsj2me.home' property if you have installed Nokia Developer's Suite for J2ME™ somewhere else. The tasks are defined with a resource named 'ndsj2me.properties' using classpathref that includes all Java™ classes distributed with Nokia Developer's Suite 3.0 for J2ME™.

```
<!-- Declare Nokia Developer's Suite for J2ME Ant tasks -->
<property name="ndsj2me.home"
value="C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME_3_0" />
<path id="ndsj2me.classpath">
        <fileset file="${ndsj2me.home}/bin/NDS_MIDPToolSet.jar" />
        <fileset dir="${ndsj2me.home}/bin/modules" includes="*.jar" />
        <fileset dir="${ndsj2me.home}/bin/lib" includes="*.jar" />
</path>
<taskdef resource="ndsj2me.properties" classpathref="ndsj2me.classpath" />
```

### 15.1.2 Setting the project emulator

Nokia Developer's Suite for J2ME™ Ant tasks use emulator devices installed in Nokia Developer's Suite for J2ME™ in one of several ways. For example, when compiling Java classes with NdsJ2meJavac the boot class path is set to the device’s boot class path. Therefore, before you can start using Nokia Developer’s Suite for J2ME™ Ant tasks you must define the project emulator and device to be used. It is mandatory to manually specify the emulator/SDK, but the device is chosen automatically if you do not set it explicitly. Emulators may contain more than one device, therefore you may want to set the project device manually.

**Note : You can find the emulators and devices available in the Configure Emulators tool of Nokia Developer’s Suite 3.0 for J2METM.**

Define the project emulator and device with the properties ‘ndsj2me.emulator’ and ‘ndsj2me.device’ in the following way:

```
<!-- Set emulator to use with Nokia Developer’s Suite for J2ME Ant tasks -->
<property name="ndsj2me.emulator" value="Nokia Prototype SDK 2.0 for J2ME™"
/>
<!-- The line below is not necessary, but it explicitly chooses the device
to use. -->
<property name="ndsj2me.device" value="Prototype_2_0_S60_MIDP_Emulator" />
```

Do not worry if you write the emulator or device name incorrectly at first, the build will fail and print out all the emulators and/or devices installed in Nokia Developer’s Suite for J2ME™. All tasks that use devices also provide you with the option of overriding project emulator and device settings.

## 15.2 Tutorials and example

### 15.2.1 Using the tasks from the command line:

- Install Ant 1.6.1.
- Make sure that the ANT_HOME environment variable is correctly set to the directory where you installed Ant
- Create a build.xml file in your project root directory
- In build.xml, set the home and define the tasks
- In build.xml, define the project emulator
- In build.xml, create targets that use Nokia Developer’s Suite for J2ME Ant tasks
- Start the Ant build from the root directory of your project by typing the command “ant”

### 15.2.2 Using the tasks in Eclipse:

- Install Eclipse 3.0.0.

- Create a build.xml file in your project's root directory
- In build.xml, set the home and define the tasks
- In build.xml, define the project emulator
- In build.xml, create targets that use Nokia Developer's Suite for J2ME Ant tasks
- Select *Window* | *Show View* | *Ant*
- In the *Ant* view, select *Add Buildfiles* and choose the project's build.xml
- In the Ant view, double-click the target you wish to build to start the build

**Note : Make sure that Eclipse is using J2SE runtime environment for Ant. This setting is found in Run | External Tools | External Tools... | Ant Build | [project build.xml] | JRE. Select Runtime JRE as a Separate JRE that is a Java 2, Standard Edition runtime. Also make sure the working directory on the same tab is set correctly.**

### 15.2.3 Example

An example build file (build.xml) can be found and used from the <NDS_INSTALL_DIR>/examples/Boids directory.

## 15.3 NdsJ2meJavac

Extends the default Javac task to compile a MIDlet using the boot class path of a device. The task also sets the "target" attribute of Javac to 1.1. For more information on the attributes and nested elements they accept, see the documentation for Javac.

*Mandatory task attributes*

| **Attribute** | **Type** | **Description** |
|---|---|---|
| srcdir | Path | Location of the Java files. For more information, see the documentation for Javac. |

*Optional task attributes*

| **Attribute** | **Type** | **Description** |
|---|---|---|
| emulator | String | Name of the emulator to be used for the boot class path. |
| device | String | Name of the device to be used for the boot class path. |

### 15.3.1 Examples

```
<ndsj2mejavac srcdir="${source.dir}" debug="true" destdir="${build.dir}"
/>
```

Compiles all .java files under the ${source.dir} directory, and stores the .class files in the ${build.dir} directory. The boot class path used is from the project's default device, and compiling with debug information is on.

```
<ndsj2mejavac srcdir="${source.dir}" debug="true" destdir="${build.dir}"
emulator="Nokia Prototype SDK 2.0" />
```

Compiles all .java files under the ${source.dir} directory, and stores the .class files in the ${build.dir} directory. The boot class path used is from the default device of Nokia Prototype SDK 2.0, and compiling with debug information is on.

## 15.4 NdsJ2mePackage

Packages a MIDlet application in a JAD/JAR file pair. If optional attributes and nested elements are not set, the task will add all MIDlets, classes and resources it finds in the class and resource paths to the package.

*Mandatory task attributes*

| **Attribute** | **Type** | **Description** |
|---|---|---|
| classpath | Path | A directory that contains the .class files to be packaged. You can only use one directory for the classpath. |
| jad | File | The application descriptor (JAD) file that will be created for the application package. |
| jar | File | The archive (JAR) file that will be created for the application package. |

*Optional task attributes*

| **Attribute** | **Type** | **Description** |
|---|---|---|
| name | String | Name of the MIDlet suite. |
| version | String | MIDlet suite version. |
| vendor | String | MIDlet suite vendor. |
| configuration | String | MicroEdition-Configuration for the MIDlet suite. If not set, configuration will be set to match the capabilities of the emulator |

| | | |
|---|---|---|
| | | device that is used in packaging the MIDlet suite. |
| profile | String | MicroEdition-Profile for the MIDlet suite. If not set, profile will be set to match the capabilities of the emulator device that is used in packaging the MIDlet suite. |
| resourcedir | String | Optional directory where resources (e.g. pictures and audio files) for the MIDlet are located. |
| preverifydir | String | Directory where preverified classes are output. Usually you do not want to set this. |
| emulator | String | Name of the emulator to be used for preverification and packaging. |
| device | String | Name of the device to be used for preverification and packaging. |

### 15.4.1 Nested Elements of the NdsJ2mePackage

NdsJ2mePackage supports several nested elements used for describing the package you are creating.

#### *15.4.1.1 midlet*

Provides the task with information about each MIDlet you wish to have included in your application package. MIDlets are numbered according to the order in which they appear in the build file.

| Attribute | Type | Description |
|---|---|---|
| name | String | Name of the MIDlet. |
| icon | String | Icon for the MIDlet. |
| classname | String | MIDlet class name. |

#### *15.4.1.2 class*

Provides the task with information about each class you wish to have included in your application package. Use the fully qualified class name when adding classes manually.

| Attribute | Type | Description |
|---|---|---|
| classname | String | Fully qualified class name. E.g. "com.nokia.example.ant.HelloNdsAnt". |

#### *15.4.1.3 resource*

Provides the task with information about each resource you wish to have included in your application package.

| Attribute | Type | Description |
|---|---|---|
| name | String | File name of the resource. |

### *15.4.1.4 push*

Provides the task with information about push connections in your application package. Push fields are numbered according to the order in which they appear in the build file.

| Attribute | Type | Description |
|---|---|---|
| url | String | Connection URL. |
| midletclass | String | MIDlet class name. |
| sender | String | Allowed push sender. |

### *15.4.1.5 Permission*

Provides the task with information about permissions your application needs.

| Attribute | Type | Description |
|---|---|---|
| value | String | The requested permission. |

### *15.4.1.6 permissionopt*

Provides the task with information about optional permissions your application may want to use.

| Attribute | Type | Description |
|---|---|---|
| value | String | The requested permission. |

### *15.4.1.7 userattribute*

Provides the task with information about user attributes that will be added to the application descriptor.

| Attribute | Type | Description |
|---|---|---|
| key | String | Key of the user attribute. |
| value | String | Value of the user attribute. |

### 15.4.2 Examples

```
<target name="package" depends="compile">
  <ndsj2mepackage
    classpath="${build.dir}"
    jad="${hellondsant.jad}" jar="${hellondsant.jar}"/>
</target>
```

Creates an application package from the MIDlet classes in the class path and resources in the resource directory using default values for JAD content. The configuration and profile of the package are set to match the capabilities of project’s default emulator device.

```
<target name="package" depends="compile">
  <ndsj2mepackage
    classpath="${build.dir}"
    name="Example MIDlets"
    version="1.0.0"
    vendor="Forum Nokia"
    configuration="CLDC-1.0" profile="MIDP-1.0"
    jad="${hellondsant.jad}" jar="${hellondsant.jar}">

            <midlet name="Hello world"
            classname="com.nokia.example.ant.HelloNdsAnt"/>
            <midlet name="Calculator"
            classname="com.nokia.example.ant.CalculatorMidlet"/>

    <class classname="com.nokia.example.ant.ExtraClass" />

    <resource name="resource.txt" />

    <push url="datagram://:444"
            midletclass="com.nokia.example.ant.HelloNdsAnt"
            sender="12.23.101.2"/>
    <push url="datagram://:446"
                     midletclass="com.nokia.example.ant.CalculatorMidlet"
            sender="12.23.*.*"/>

    <permission value="javax.microedition.io.Connector.http"/>
    <permission value="javax.microedition.io.Connector.https"/>

    <permissionopt value="javax.microedition.io.PushRegistry"/>

    <userattribute key="mykey" value="myvalue"/>
    <userattribute key="mykey2" value="myvalue2"/>

  </ndsj2mepackage>
    </target>
```

Creates an application package with manually defined MIDlet, push, permission and user attribute information.

## 15.5 NdsJ2mePpJar

Extends the default JAR task to package a Personal Profile application in a JAR file. For more information about the attributes and nested elements they accept, see JAR task documentation.

| *Mandatory task attributes* | | |
|---|---|---|
| **Attribute** | **Type** | **Description** |
| mainclass | String | Main class of the application. |
| destfile | File | The JAR file to be created. |

| *Optional task attributes* | | |
|---|---|---|
| **Attribute** | **Type** | **Description** |
| name | String | Name of the Personal Profile application. |
| version | String | Application version. |
| vendor | String | Application vendor. |
| icon | String | Optional icon for the application. |
| ibmj9 | String | Optional manifest line for IBM J9 virtual machine support. |

#### *15.5.1.1 Nested Elements of NdsJ2mePpJar*

NdsJ2mePpJar supports several nested elements for describing the package you are creating. This document only describes the nested elements that are not a part of default JAR task implementation. Please see JAR task documentation for a list of other nested elements available.

#### *15.5.1.2 userattribute*

Provides the task with information about user attributes that will be added to the application manifest.

| Attribute | Type | Description |
|---|---|---|
| key | String | Key of the user attribute. |
| value | String | Value of the user attribute. |

### 15.5.2 Examples

```
<target name="ppjar" depends="compile">
  <ndsj2meppjar
    mainclass="com.nokia.example.MyApplication"
    destfile="${dist}/application.jar" basedir="${build}" />
</target>
```

Creates a Personal Profile package for the application whose main class is com.nokia.example.MyApplication.

```
<target name="ppjar" depends="compile">
  <ndsj2meppjar destfile="${dist}/application.jar"
    mainclass="com.nokia.example.MyApplication"
    name="MyApplication" version="1.0.0" vendor="Forum Nokia">

    <fileset dir="${build}"
      excludes="${build}/Test.class" />
    <fileset dir="${src}/resources"/>
    <userattribute key="key1" value="value"/>
  </ndsj2meppjar>
</target>
```

Creates a Personal Profile package from build and resource directories with a manually defined name, version, vendor and user attribute information.

## 15.6 NdsJ2meRun

Runs an application in an emulator device. The application can be a MIDlet class, a JAD/JAR file pair or a Personal Profile application JAR file.

*Mandatory task attributes*

| Attribute | Type | Description |
|---|---|---|
| application | File | Application file. Can be either a .class file that extends `javax.microedition.midlet.MIDlet`, a JAD file or a JAR file that contains a Personal Profile application. |

*Optional task attributes*

| Attribute | Type | Description |
|---|---|---|
| classpath | Path | Class path for .class files. Not necessary when running packaged JAD/JAR files or a Personal Profile application JAR. |
| wait | boolean | Set this attribute to true if you want the task to wait until the emulator process is finished. It is false by default, i.e. the task will finish asynchronously immediately after it has started the emulator process. |
| debug | String | Debug address that enables remote debugging. The debugger |

| | | |
|---|---|---|
| | | address should be specified in the form <port> or <host>:<port>. |
| heapsize | String | Heap size for the emulator. |
| emulator | String | Name of the emulator to be used for emulation. |
| device | String | Name of the device to be used for emulation. |

### 15.6.1 Examples

```
<ndsj2merun
application="${build.dir}/com/nokia/example/ant/HelloNdsAnt.class" />
```

Runs the MIDlet class on the project's default device.

```
<ndsj2merun                  application="${release.dir}/HelloNdsAnt.jad"
device="Prototype_2_0_S90_MIDP_Emulator" />
```

Runs the specified JAD/JAR application package on the Series 90 device of Nokia Prototype SDK 2.0.

```
<ndsj2merun application="${ppapp.jar}" emulator="S80_DP2.0_PP_SDK" />
```

Runs the Personal Profile JAR on the S80 Personal Profile emulator.

## 15.7 NdsJ2meConvertAudio

This task converts MIDI and XML files to other audio formats (OTA ringing tone, bytes or XML). When this task is run from inside the Eclipse, you need to terminate the build process manually. When you use Ant from the command line, the process finishes properly.

*Mandatory task attributes*

| **Attribute** | **Type** | **Description** |
|---|---|---|
| src | File | A source MIDI file, an audio XML file or a directory containing multiple audio files. If src is a directory, the task will iterate over all audio files it finds in the directory, convert them and save them in the dest directory. Known source file types are .mid, .midi and .smf. |
| dest | File | Destination where conversion content is written. If src is a file, dest must also be a file. If src is a directory, dest must also be a directory. |

*Optional task attributes*

| Attribute | Type | Description |
|---|---|---|
| channel | int | MIDI channel to be converted when converting a MIDI file. By default 1. |
| desttype | String | The contents of the destination file are determined by this attribute. It must be "ota," "bytes" or "xml." By default, audio is converted to the "ota" type. OTA type files will default to the extension .txt, bytes to .java and xml to .xml. |

### 15.7.1 Examples

```
<ndsj2meconvertaudio            src="${audio.dir}"            channel="1"
dest="${resource.dir}" desttype="ota" />
```

Converts all audio files in 'audio.dir' to "ota" type audio files in 'resource.dir.' Any MIDI files found will be converted using channel 1.

## 15.8 NdsJ2meSign

Signs an application package with a key pair from a keystore.

*Mandatory task attributes*

| Attribute | Type | Description |
|---|---|---|
| jad | File | Application descriptor file for the package you want to sign. |
| alias | String | Key pair to be used for the signing operation. |
| keypassword | String | Password for accessing the alias in the keystore. |

*Optional task attributes*

| Attribute | Type | Description |
|---|---|---|
| keystore | File | Keystore from which the key pair alias is opened. If not set, Nokia Developer's Suite for J2ME™'s default keystore will be used. |
| keystorepassword | String | Password for opening the keystore. Only needed when not using the default keystore of Nokia Developer's Suite for J2ME™ |

### 15.8.1 Examples

```
<ndsj2mesign          jad="${hellondsant.jad}"          alias="default"
keypassword="${pword}" />
```

Signs the application package described with ${hellondsant.jad} using a key pair with alias 'default' read from the default keystore of Nokia Developer's Suite for J2ME™.

## 15.9 NdsJ2meDeployDevice

Deploys applications to actual devices. By default, it deploys the given application package to all devices it finds.

*Mandatory task attributes*

| Attribute | Type | Description |
|---|---|---|
| application | File | Application file for the package you want to deploy. Supports SIS and JAD files. For an application descriptor (JAD) file, the corresponding archive file (JAR) is located automatically and deployed. |

*Optional task attributes*

| Attribute | Type | Description |
|---|---|---|
| destdevice | String | Device name or part of device name for manually selecting a destination device. When this attribute is set, the application will be deployed to the selected device if it is found. |
| failonerror | boolean | If deployment to any of the devices fails, this attribute determines whether the build also fails. By default, the value is 'true,' but if you do not want the build to fail when deployment has failed, set this attribute to 'false.' |

### 15.9.1 Examples

```
<ndsj2medeploydevice application="${hellondsant.jad}" />
```

Deploys the application package described with ${hellondsant.jad} to all devices Nokia Developer's Suite for J2ME™ finds.

## 15.10 NdsJ2meDeployFtp

Deploys application packages to an FTP server.

*Mandatory task attributes*

| Attribute | Type | Description |
|---|---|---|
| file | File | The application file you want to deploy. For an application descriptor (JAD) file, the corresponding archive file (JAR) is located automatically and deployed. This attribute is not mandatory if nested FileSet elements are used for declaring the files to be uploaded. |
| host | String | Host name for the FTP server to deploy to. |
| username | String | Username for the FTP server. If set to “anonymous,” attribute ‘password’ does not need to be set. |
| password | String | User password for the FTP server. If username was set to “anonymous,” this attribute need not be set, as it will also use “anonymous” for password. |

*Optional task attributes*

| Attribute | Type | Description |
|---|---|---|
| ftpdir | String | The FTP server directory where the files will be uploaded to. |
| port | int | Port number of the FTP server, 21 by default. |
| failonerror | boolean | If deployment fails, this attribute determines whether the build also fails. By default, the value is ‘true,’ but if you do not want the build to fail when deployment has failed, set this attribute to ‘false.’ |

### 15.10.1 Nested Elements of NdsJ2meDeployFtp

NdsJ2meFtp supports nested FileSet elements for declaring the files to be uploaded. Use only one of the two possibilities: an attribute ‘file’ or nested FileSets.

#### *15.10.1.1 fileset*

Provides the task with information about files that should be uploaded to the FTP server. For more information, see Ant core tasks documentation.

### 15.10.2 Examples

```
<ndsj2medeployftp        file="${hellondsant.jad}"        host="127.0.0.1"
username="anonymous" failonerror="false" />
```

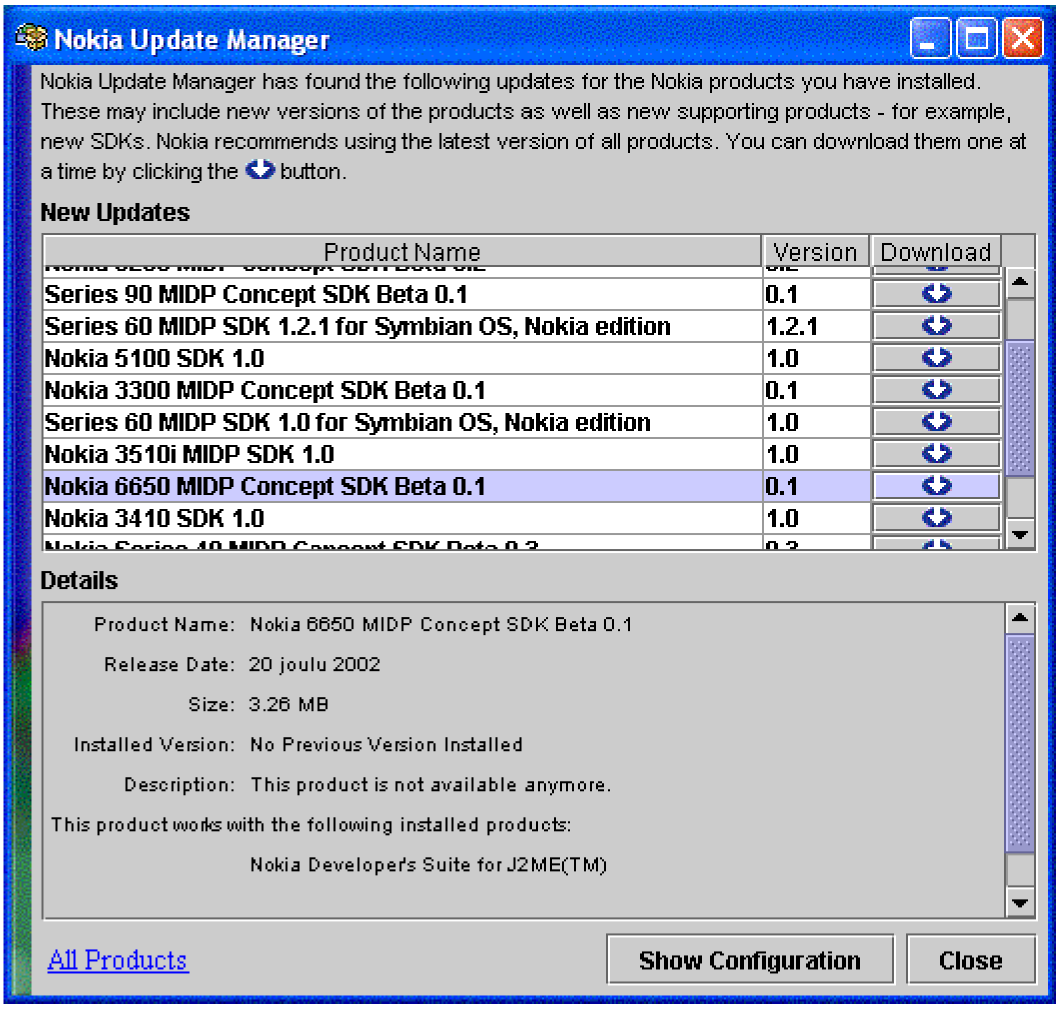
Deploys the application package described with ${hellondsant.jad} to the local FTP server using username “anonymous” and password “anonymous.” Even if the deployment fails the build does not.

# 16 Check for Updates

Choosing *Tools* | *Check for Updates* lauches the Nokia Update Manager (Figure 41) which:

- Presents updates for currently-installed Forum Nokia products and related products; users can view details about those products and download them
- Provides a link to other SDKs and Tools available at Forum Nokia

Nokia Update Manager requires network connectivity and a valid proxy information if your computer is behind a firewall. Nokia Update Manager is also scheduled to run automatically in periodic intervals. Users can change scheduler and proxy configuration.


Nokia Update Manager

Nokia Update Manager has found the following updates for the Nokia products you have installed. These may include new versions of the products as well as new supporting products - for example, new SDKs. Nokia recommends using the latest version of all products. You can download them one at a time by clicking the button.

New Updates

| Product Name | Version | Download |
|---|---|---|
| Series 90 MIDP Concept SDK Beta 0.1 | 0.1 | ⇩ |
| Series 60 MIDP SDK 1.2.1 for Symbian OS, Nokia edition | 1.2.1 | ⇩ |
| Nokia 5100 SDK 1.0 | 1.0 | ⇩ |
| Nokia 3300 MIDP Concept SDK Beta 0.1 | 0.1 | ⇩ |
| Series 60 MIDP SDK 1.0 for Symbian OS, Nokia edition | 1.0 | ⇩ |
| Nokia 3510i MIDP SDK 1.0 | 1.0 | ⇩ |
| Nokia 6650 MIDP Concept SDK Beta 0.1 | 0.1 | ⇩ |
| Nokia 3410 SDK 1.0 | 1.0 | ⇩ |

Details

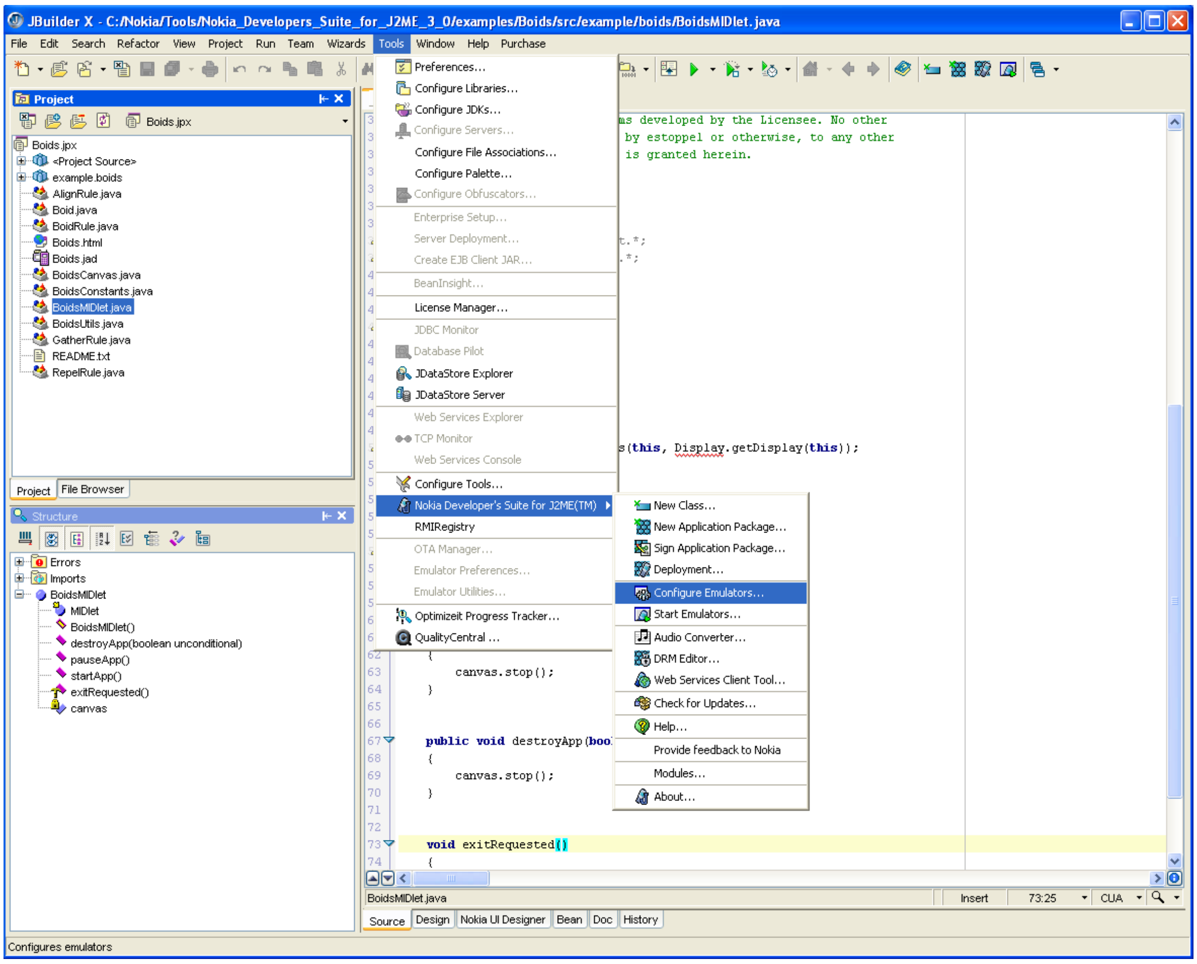
Product Name: Nokia 6650 MIDP Concept SDK Beta 0.1
Release Date: 20 joulu 2002
Size: 3.26 MB
Installed Version: No Previous Version Installed
Description: This product is not available anymore.
This product works with the following installed products:
Nokia Developer's Suite for J2ME(TM)

Figure 41: Nokia Update Manager

## 17 Using the tools with Borland JBuilder

See the Installation and Configuration Guide [7] for instructions on how to integrate Nokia Developer's Suite 3.0 for J2ME™ with Borland JBuilder. If you are using JBuilder 7 or older, it must be configured to use JDK 1.4.1 or later for Nokia Developer's Suite 3.0 for J2ME™ to work. See section 14.4 for more information. If you want to do JBuilder integration after Nokia Developer's Suite 2.2 for J2ME™ installation, copy the content of the folder NDS_INST_DIR\jb to the JBuilder's \lib\ext -folder.

Nokia Developer's Suite 3.0 for J2ME™ tools can be started from JBuilder's menu in *Tools* | *Nokia Developer's Suite for J2ME* (Figure 42).



```
ms developed by the Licensee. No other
 by estoppel or otherwise, to any other
 is granted herein.

t.*;
.*;

s(this, Display.getDisplay(this));

    {
        canvas.stop();
    }


    public void destroyApp(boo
    {
        canvas.stop();
    }


    void exitRequested()
    {
```

Figure 42: IDE of JBuilder

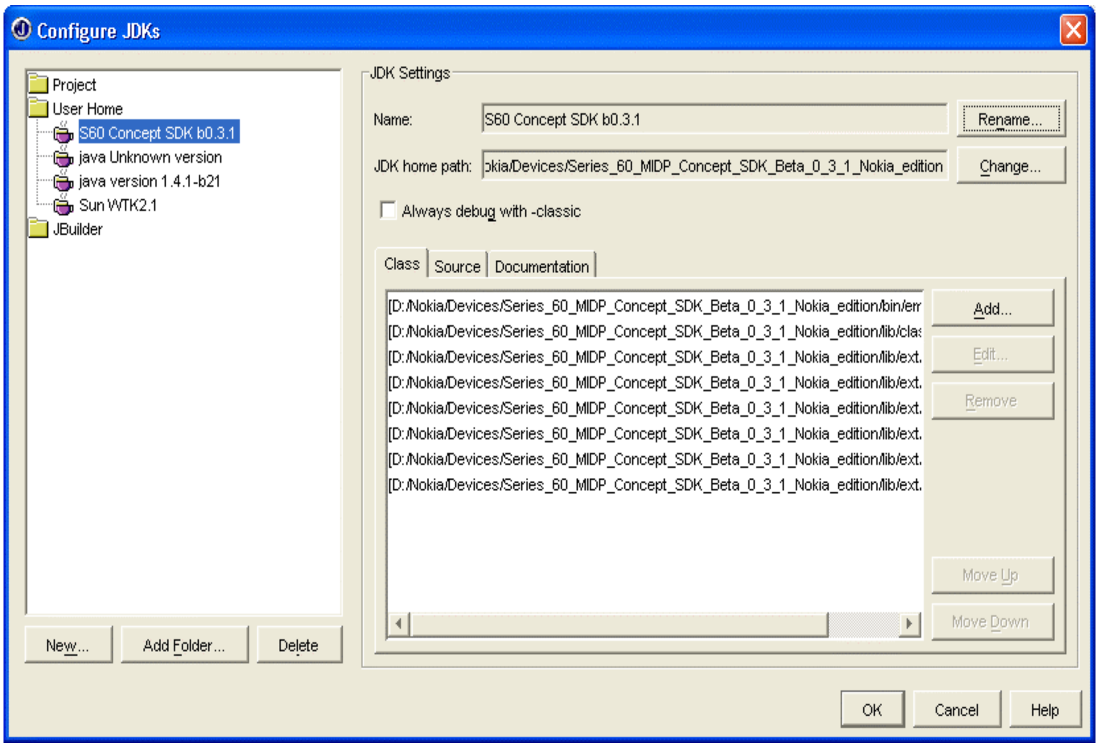
To compile and deploy MIDP and PP applications successfully in JBuilder, you need to configure JBuilder to use Java 2 Platform, Micro Edition, in particular, CLDC/ CDC and MIDP/PP API classes. The preferred way is to configure a project to use a specific Nokia MIDP (for example, Nokia Prototype 2.0) or Personal Profile SDK as its JDK.

## 17.1 Configuring JDKs

In JBuilder, you are able to add several JDKs to JBuilder's configuration. Each project in JBuilder uses one of the configured JDKs. For MIDlet and PP application development, if you have not already done so, add a suitable JDK to JBuilder's configuration by selecting:

*Tools | Configure JDKs...*

From the dialog that opens (Figure 43), press **New...**. A new dialog opens, in which you can browse for an existing JDK home path. Browse and select the root directory of an emulator. For example, if you wish to use a particular Nokia MIDP SDK/Emulator, select the root directory of that Nokia emulator. Having selected the root directory, edit the JDK name to a more descriptive one if you wish. Accept the changes in both dialogs.


Configure JDKs

Project
User Home
S60 Concept SDK b0.3.1
java Unknown version
java version 1.4.1-b21
Sun WTK2.1
JBuilder

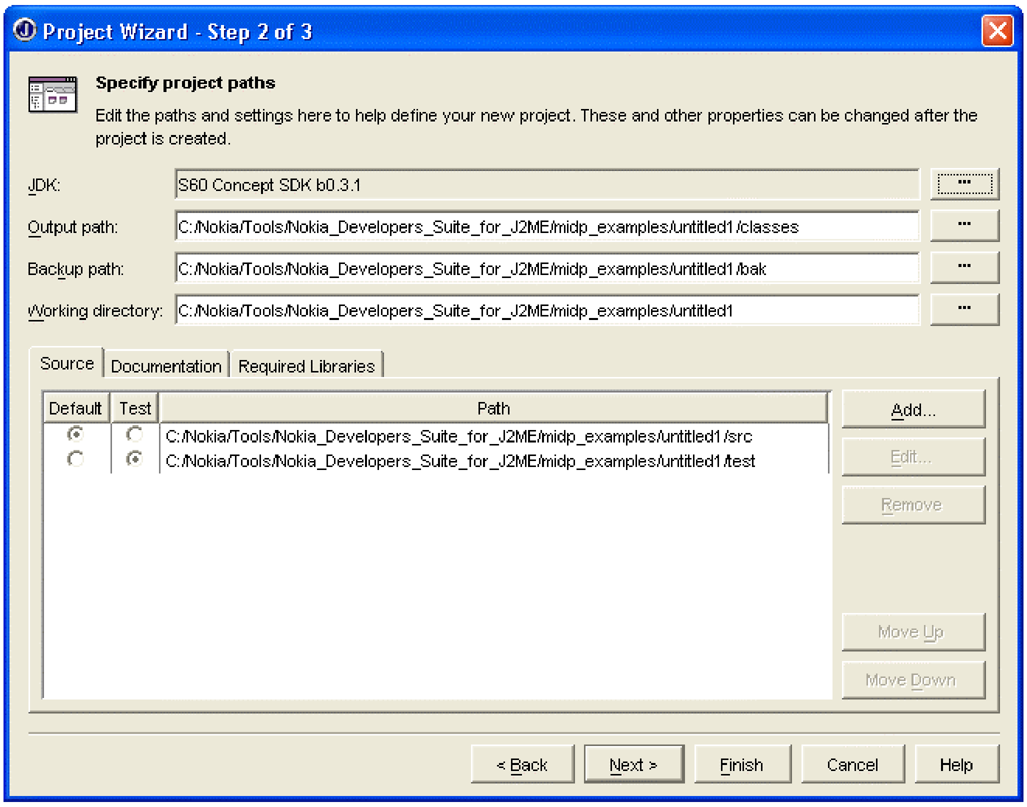
JDK Settings
Name: S60 Concept SDK b0.3.1
JDK home path: okia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition
Always debug with -classic

Class | Source | Documentation

[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/bin/em
[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/lib/clas
[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/lib/ext.
[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/lib/ext.
[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/lib/ext.
[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/lib/ext.
[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/lib/ext.
[D:/Nokia/Devices/Series_60_MIDP_Concept_SDK_Beta_0_3_1_Nokia_edition/lib/ext.

Figure 43: Configure JDKs dialog

The selected SDK is now in JBuilder's configuration. To use it in a project, you must configure the project's JDK property. When creating a new project, change the JDK setting on the second page of the Project Wizard (Figure 44). Change the setting to the J2ME ™ JDK.



**Specify project paths**

Edit the paths and settings here to help define your new project. These and other properties can be changed after the project is created.

| | |
|---|---|
| JDK: | S60 Concept SDK b0.3.1 |
| Output path: | C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_examples/untitled1/classes |
| Backup path: | C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_examples/untitled1/bak |
| Working directory: | C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_examples/untitled1 |

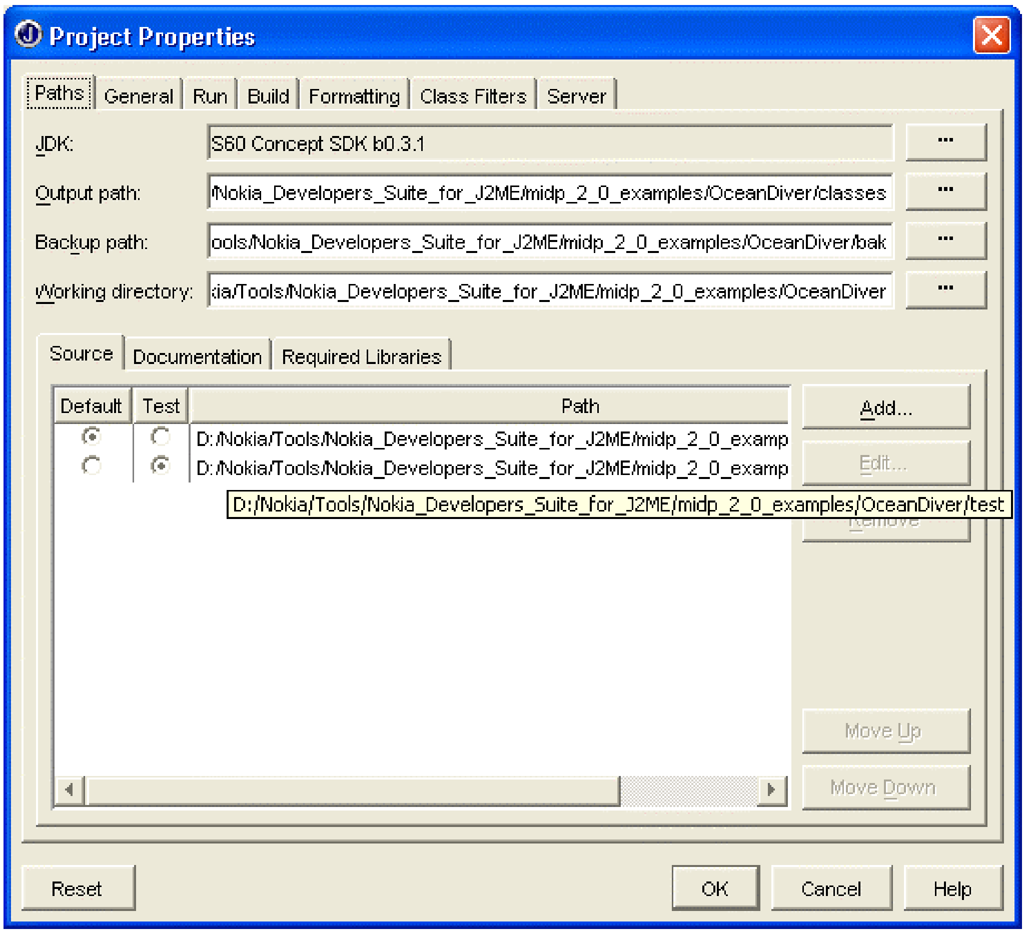
| Default | Test | Path |
|---|---|---|
| ● | ○ | C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_examples/untitled1/src |
| ○ | ● | C:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_examples/untitled1/test |

Figure 44: Project Wizard page 2

To change the property at a later time or for an existing project, select:

*Project | Project Properties...*

From the dialog (Figure 45), change the JDK property to the J2ME™ JDK.


Project Properties

Paths | General | Run | Build | Formatting | Class Filters | Server

JDK: S60 Concept SDK b0.3.1

Output path: /Nokia_Developers_Suite_for_J2ME/midp_2_0_examples/OceanDiver/classes

Backup path: ools/Nokia_Developers_Suite_for_J2ME/midp_2_0_examples/OceanDiver/bak

Working directory: kia/Tools/Nokia_Developers_Suite_for_J2ME/midp_2_0_examples/OceanDiver

Source | Documentation | Required Libraries

| Default | Test | Path |
|---|---|---|
| ● | ○ | D:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_2_0_examp |
| ○ | ● | D:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_2_0_examp |

D:/Nokia/Tools/Nokia_Developers_Suite_for_J2ME/midp_2_0_examples/OceanDiver/test

Figure 45: Project properties dialog

### 17.2 JBuilder® 7.0 and earlier versions

Though not supported, it is possible to use NDS with older versions (7.0 and older) of JBuilder that by default use JDK 1.3.1 or earlier. To use Nokia Developer's Suite 3.0 for J2ME™, you need to configure JBuilder to use a more recent version of the J2SDK™. In JBuilder's bin directory, you will find a file called jdk.config. Copy this file to another file named jdk141.config. Then edit the lines in jdk141.config containing javapath and addpath directives to point to your installation of J2SDK™ 1.4.1.

For example, in JBuilder® 7.0:

Javapath C:/j2sdk1.4.1/jre/bin/client/jvm.dll

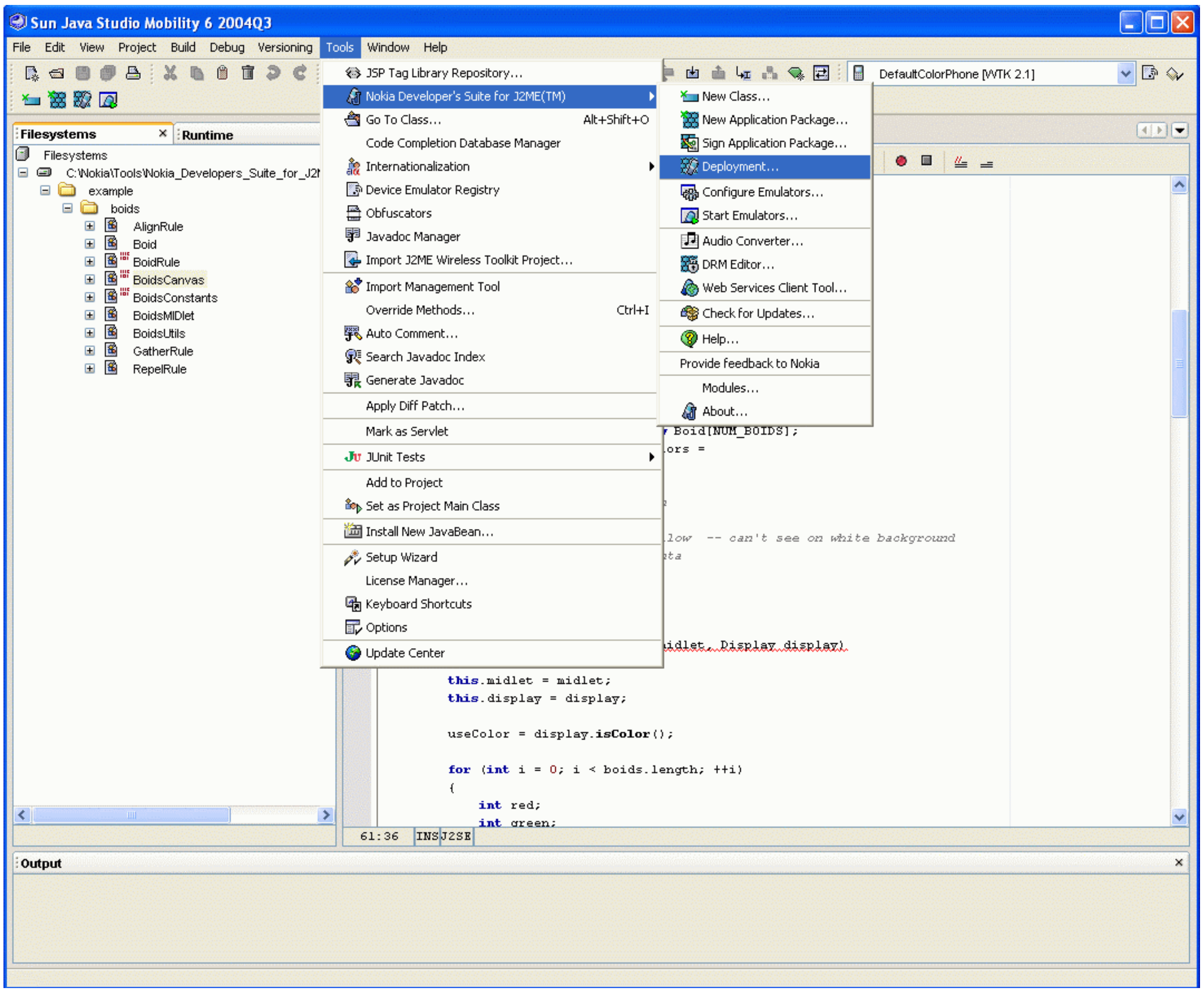
addpath C:/j2sdk1.4.1/lib/tools.jar

# 18 Using the tools with Sun™ Java Studio Mobility 6 or NetBeans 4.0

See the Installation and Configuration Guide [7] for instructions on how to integrate Nokia Developer's Suite 3.0 for J2ME™ with Sun™ Java Studio or NetBeans.

When Nokia Developer's Suite 3.0 for J2ME™ has been integrated with Sun Java Studio, you can start the tools from the Sun Java Studio's main menu in *Tools | Nokia Developer's Suite for J2ME™* (Figure 46). To start creating applications, create a new project and mount a file system onto it.

**Note : In Sun™ Java Studio, the source and class directories are the same by default, that is, the Java source files are located in the same directory as the compiled classes.**



```
Boid[NUM_BOIDS];
ors =

low  -- can't see on white background
ta

idlet, Display display)

    this.midlet = midlet;
    this.display = display;

    useColor = display.isColor();

    for (int i = 0; i < boids.length; ++i)
    {
        int red;
        int green;
```

Figure 46: IDE of Sun Java Studio

A Sun Java Studio project may have several file systems mounted. Therefore, before starting any of the Nokia Developer's Suite 3.0 for J2ME™ tools, you have to select the file system. To select a file system from Filesystems, click a node in the desired file system. The file system where the node is located will be used as the user class path and source path by Nokia Developer's Suite 3.0 for J2ME™. If there is a "res" directory at the same level as the class path or directly under the class path, it will be used to look for resources. If you are editing a file while starting any of the Nokia Developer's Suite 3.0 for J2ME™ tools, the file system for that file will be used when deciding the user class path, source path and resource path.

When creating an Application Package, the tool will suggest a directory in which you can create the JAR and JAD file.

## 18.1 MIDlet Creation Tutorial

This section provides step-by-step instructions on how to create a project in Sun Java Studio and NetBeans, how to make a MIDlet and, finally, how to make an Application Package and test it in the emulator. A Personal Profile application can be created, packed and tested in the similar way.

Creating a project in Sun Java Studio requires more actions than in NetBeans. Otherwise the functions of the two IDEs are very similar.

### 18.1.1 Step 1: Create a project

**In Sun Java Studio:**

Select *Project* | *Project Manager* | *New...*

Enter a name for your project. For example, enter MyProject and confirm the name with **OK**. The project is opened in the IDE and the Filesystems pane displays Filesystems (Figure 47).

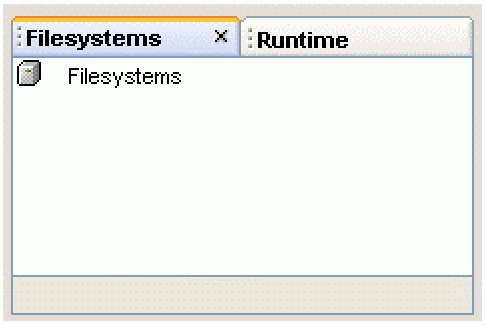


Figure 47: The Filesystems window

**Note : If the Filesystems window is not visible, select Window | *Filesystems.***

In your local file system, create a directory that will be used as the root-directory of the project. For the sake of this example, we will use Windows Explorer to create the D:\myprojectfolder\src directory that will be used as the project's source directory. You must not have embedded spaces in the project path.

Next, you need to mount your project source directory in Sun Java Studio. Make sure that the Filesystems tab is active, right-click the Filesystem node and select Mount | Local Directory....

Browse to your project source directory, select it and accept it with **Finish**.

**Note : You should not select a directory that contains classes other than J2ME classes. The root of a drive, for example, C:\ or D:\, should also not be used.**

**In NetBeans:**

Select *File* | *New Project*. A New Project wizard opens. Under Categories, select Mobile. Under Projects, select Mobile Application. To continue, click **Next**. Enter a Project Name and define Project Location, for example, D:\myprojectfolder\. Complete the project creation by clicking **Finish**.

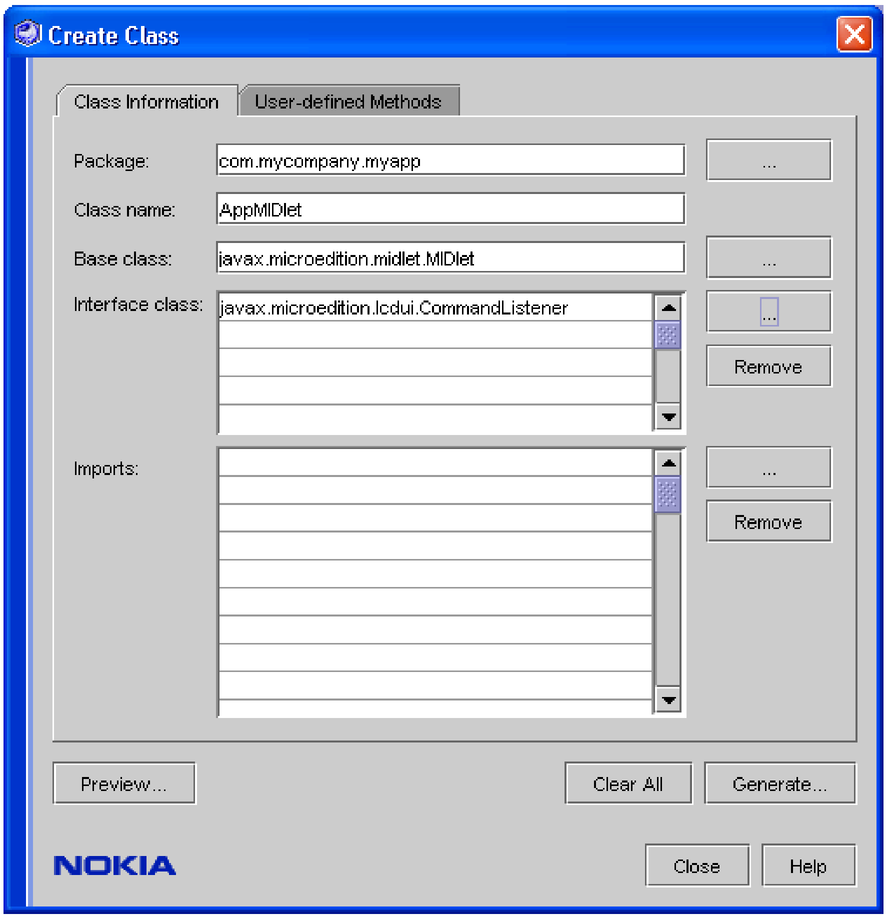
Your newly created project opens in the Projects and Files pane.

### 18.1.2 Step 2: Create a MIDlet class

On the *Filesystems* tab, select D:\myprojectfolder\src node to enable Nokia Developer's Suite 3.0 for J2ME™ tools if they are disabled. Then select:

*Tools* | *Nokia Developer's Suite for J2ME* | *New Class...*

Create your MIDlet using the *Create Class* tool (Figure 48).



Package: com.mycompany.myapp
Class name: AppMIDlet
Base class: javax.microedition.midlet.MIDlet
Interface class: javax.microedition.lcdui.CommandListener
Imports:

Figure 48: Create class tool

The com.mycompany.myapp.AppMIDlet class you created appears on the *Filesystems* tab under the node that *Create Class* tool created in the selected Filesystems (Figure 49).

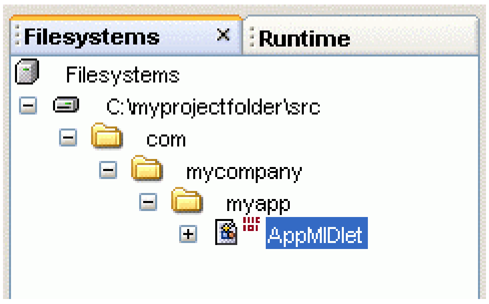


Figure 49: Filesystems tab showing the created class

Double-click the AppMIDlet node on the *Filesystems/Files* tab. The *Source Editor* view is opened showing the created source code. You can now edit your code.

### 18.1.3 Step 3: Compile your class

Classes are compiled according to the default emulator libraries. The default emulator configuration is displayed in the default emulator text field in the right upper corner of the Sun Java Studio / NetBeans window. You can select an emulator from the drop-down list.

If you want to add a new emulator to the **Sun Java Studio**, select *Device Emulator Registry* icon next to the text field. The *Explorer [Installed Emulators]* pane opens (Figure 50). Select **Add...**. Add an emulator using the *Add Emulator* wizard in the following way:

1. Browse to a suitable emulator in the wizard. Click **Next** to continue.
2. Configure the emulator. Click **Next** to continue.
3. Choose the emulator devices that you want to add. Complete the process by clicking **Finish**.

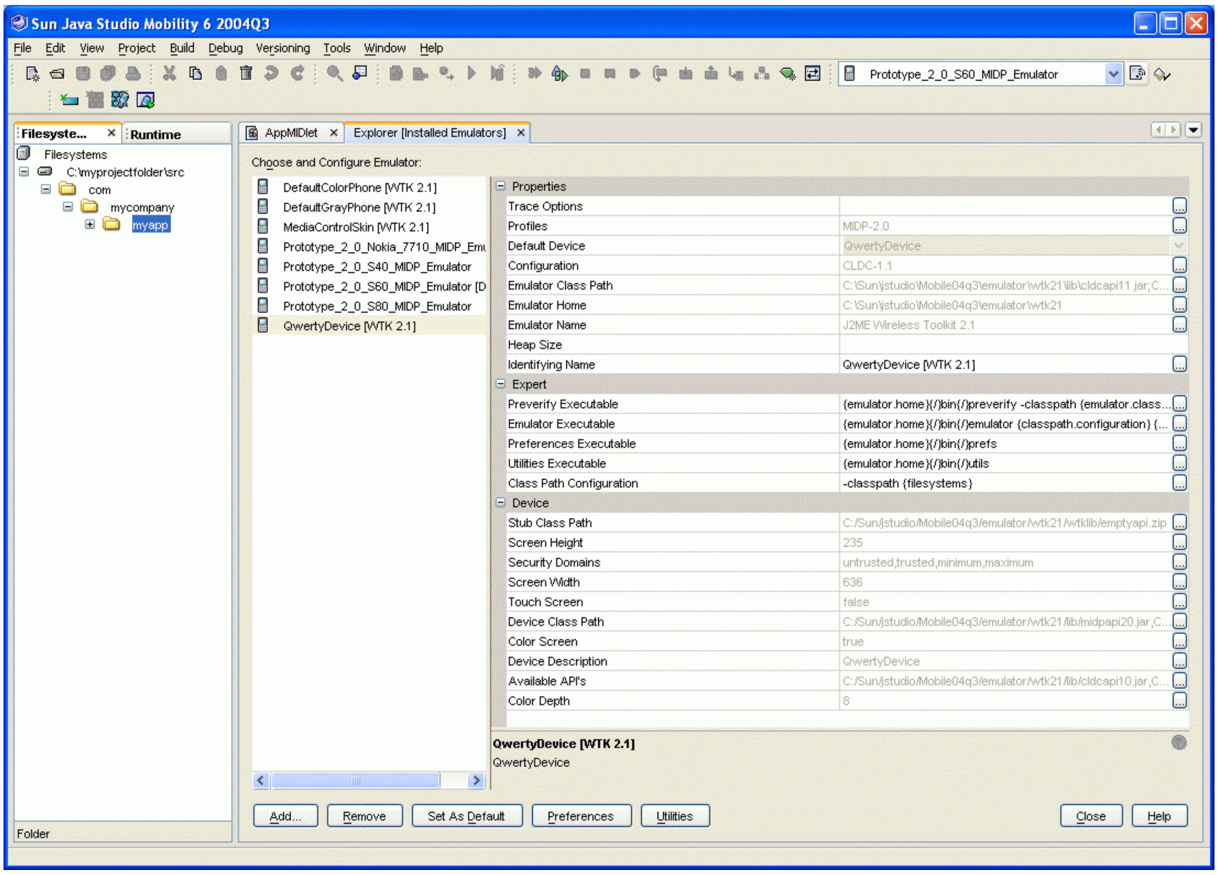
The emulator you added should appear in the *Choose and Configure Emulator* list and in the drop-down list.

In **NetBeans**, you can add a new Emulator Platform in the following way:

1. Select *Tools* | *Java Platform Manager*.
2. In the Java Platform Manager dialog, click the **Add Platform...** button.
3. Browse to the directory of the emulator you want to add. For example, C:\Nokia\Devices\Nokia_Prototype_SDK_2_0. Click **Next**.
4. The IDE configures the platform. Click **Finish** to complete the configuration.
5. Exit the Java Platform dialog.
6. Choose *File* | “Mobile Application” Properties.
7. Select the Platform node. You can change the emulator or add a new configuration. To add a new configuration, select Add Configuration from the drop-down list in the Project Configuration field and then enter the requested information. If you only want to change the emulator, select the new emulator from the Emulator Platform drop-down menu. Click **OK**.

The new configuration appears in the default emulator text field drop-down list in the Sun Java Studio / Net Beans window.

In both IDEs, you can then compile the class by right-clicking the application node in the *Filesystems/Files* tab and selecting *Compile/Compile File*. If the compilation is successful, you will receive a "Finished" message in the Output Window.


Sun Java Studio Mobility 6 2004Q3
File Edit View Project Build Debug Versioning Tools Window Help
Prototype_2_0_S60_MIDP_Emulator
Filesyste... Runtime
Filesystems
C:\myprojectfolder\src
com
mycompany
myapp
Folder
AppMIDlet
Explorer [Installed Emulators]
Add... Remove Set As Default Preferences Utilities Close Help

Choose and Configure Emulator:

DefaultColorPhone [WTK 2.1]
DefaultGrayPhone [WTK 2.1]
MediaControlSkin [WTK 2.1]
Prototype_2_0_Nokia_7710_MIDP_Emu
Prototype_2_0_S40_MIDP_Emulator
Prototype_2_0_S60_MIDP_Emulator [D
Prototype_2_0_S80_MIDP_Emulator
QwertyDevice [WTK 2.1]

| | |
|---|---|
| **Properties** | |
| Trace Options | |
| Profiles | MIDP-2.0 |
| Default Device | QwertyDevice |
| Configuration | CLDC-1.1 |
| Emulator Class Path | C:\Sun\jstudio\Mobile04q3\emulator\wtk21\lib\cldcapi11.jar;C... |
| Emulator Home | C:\Sun\jstudio\Mobile04q3\emulator\wtk21 |
| Emulator Name | J2ME Wireless Toolkit 2.1 |
| Heap Size | |
| Identifying Name | QwertyDevice [WTK 2.1] |
| **Expert** | |
| Preverify Executable | {emulator.home}{/}bin{/}preverify -classpath {emulator.class... |
| Emulator Executable | {emulator.home}{/}bin{/}emulator {classpath.configuration} {... |
| Preferences Executable | {emulator.home}{/}bin{/}prefs |
| Utilities Executable | {emulator.home}{/}bin{/}utils |
| Class Path Configuration | -classpath {filesystems} |
| **Device** | |
| Stub Class Path | C:/Sun/jstudio/Mobile04q3/emulator/wtk21/wtklib/emptyapi.zip |
| Screen Height | 235 |
| Security Domains | untrusted,trusted,minimum,maximum |
| Screen Width | 636 |
| Touch Screen | false |
| Device Class Path | C:/Sun/jstudio/Mobile04q3/emulator/wtk21/lib/midpapi20.jar,C... |
| Color Screen | true |
| Device Description | QwertyDevice |
| Available API's | C:/Sun/jstudio/Mobile04q3/emulator/wtk21/lib/cldcapi10.jar,C... |
| Color Depth | 8 |

**QwertyDevice [WTK 2.1]**
QwertyDevice

Figure 50: Setting the default emulator

### 18.1.4 Step 4: Create the Application Package

Select a node in the mounted D:\myprojectfolder\src to enable Nokia Developer's Suite 3.0 for J2ME™ tools if they are disabled. Then select:

*Tools | Nokia Develop's Suite for J2ME ™ | New Application Package...*

Set the required attributes and generate the files with **Generate**. To make the JAR and JAD files visible on the *Filesystems* tab, create them in the suggested directory, in this case D:\myprojectfolder. (For more information on this tool, see Chapter 7: Create Application Package.)

The MIDlet Suite (MyMIDlet.jar) and Application Descriptor (MyMIDlet.jad) you created are not displayed in the *Explorer* window because they are not at the same level as the source code. To see the created JAD and JAR files in the *Explorer* window, you can mount the root directory of your source directory. In this example project it would mean mounting D:\myprojectfolder (Figure 51).

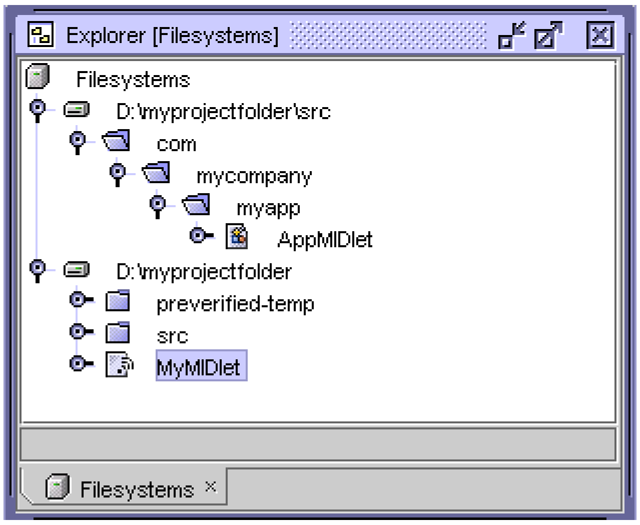


Figure 51: MIDlet JAR and JAD displayed in the Explorer window

### 18.1.5 Step 5: Test your MIDlet Suite in the emulator

When the Application Package is created, test your MIDlet Suite in the emulators. Select:

*Tools | Nokia Developer's Suite for J2ME ™ | Start Emulators...*

You can now start the selected emulators by clicking **Emulate** and they will run your MIDlet.

# 19 Using the Tools with Eclipse

Nokia Developer's Suite 3.0 for J2ME™, Eclipse edition offers two features that enable MIDP and Personal Profile application development in Eclipse. The SDK Plug-in enables using MIDP/Personal Profile (PP) SDKs with Eclipse, while NDS integration offers Nokia Developer's Suite 3.0 for J2ME™ functionalities for the Eclipse user. These functions are not integrated with each other and some actions can be performed with both. Operations performed in one plug-in do not affect other plug-in settings. E.g. if you set a default SDK with the NDS integration, the default SDK that has been set with the SDK plug-in will not be changed.

See the Installation and Configuration Guide [7] for instructions on how to integrate Nokia Developer's Suite 3.0 for J2ME™ with Eclipse. If you want to integrate Nokia Developer's Suite 3.0 for J2ME™ with Eclipse after installation, copy the contents of the NDS_INST_DIR\bin\eclipse folder to the Eclipse's plugins and features root directory.

## 19.1 SDK Plug-in Use With Eclipse

SDK plug-in is an independent plug-in that enables you to run and debug applications (MIDP and Personal Profile) in Eclipse.

**Note : To be able to use this feature you need to be in the Java Perspective in Eclipse. The perspective can be opened: *Window* | *Open Perspective* | *Java* .**

### 19.1.1 Creating a New Project

It is recommended that you add Nokia SDKs to work with Eclipse integration before you start creating new projects. Select *Add new Nokia SDK* icon from the tool bar. Browse the SDK's folder. Add the selected SDK by clicking **OK**.

Once you have added MIDP and PP SDKs to Eclipse, a suitable SDK is suggested by default when you create a new MIDP or PP project.

To create a new Project:

- Choose *File* | *New* | *Project...* in the Java Perspective. Then choose *Java* | *MIDP Project (Nokia SDK Plug-in) / Personal Profile Project (Nokia SDK Plug-in)* and click **Next** to continue.
- Enter a name for the project. If you choose a directory for the Project content, make sure the project name is the same as the name of the content directory. You can, for example, select the directory NDS_INST_DIR\examples\FormMIDlet directory or use the default directory. Click **Next** to continue.
- Select the SDK you wish to use for compiling and preverifying the MIDP / PP project. If the SDK is not displayed in the drop-down list, click *Browse...* and select the root directory of the SDK (e.g. c:\Nokia\Devices\Nokia_Prototype_SDK_2_0). Click **Next** to continue.
- Make sure that the project source folder on build path and default output folders are set correctly (e.g. MyProject\src and MyProject\bin) on the *Java Settings* | *Source* tab. Click **Finish** to continue.

**Note : Use only one source folder for your MIDlet / PP application. Nokia Developer's Suite 3.0 for J2ME™ only supports one source folder at a time. Files added to any other folders are not recognized.**

### 19.1.2 Changing the SDK Used for Compilation and Preverification

- Select your project in the Package Explorer and choose *Project* | *Properties* .
- Select SDK Plug-in properties and click *Browse...* Select the SDK root directory (e.g. C:\Nokia\Devices\Nokia_Prototype_SDK_2_0) to introduce an SDK to the Eclipse IDE as a new Java Runtime Environment (JRE).
- Click **OK** to approve the changes to the project properties.

### 19.1.3 Running an Application

When you wish to run the source of a MIDlet class / PP class in an SDK, select the class in one of the following ways:

- Select the class's .java file in the Package Explorer
- Select the .jad file (with MIDP applications) in the Package Explorer
- Select the project that includes the application in the Package Explorer
- Open the application in the editor

Then choose *Run* | *Run As* | *Nokia SDK* . This will create a launch configuration for the application. The default emulator starts up.

You can select another SDK for the launch configuration by editing the launch configuration. Choose *Run* | *Run...* The dialog for creating, managing, and running configurations opens. Select a launch configuration under Nokia SDK Plug-in in the configurations list. On the Nokia SDK Plug-in tab you can select, which SDK to use and also launch the SDK specific dialogs for preferences and utilities. Select the trace options you wish to use by selecting the appropriate check boxes.

**Note : The trace options are included in the Preferences menu of some SDKs. Click the Preferences button to access these settings.**

### 19.1.4 Debugging an application

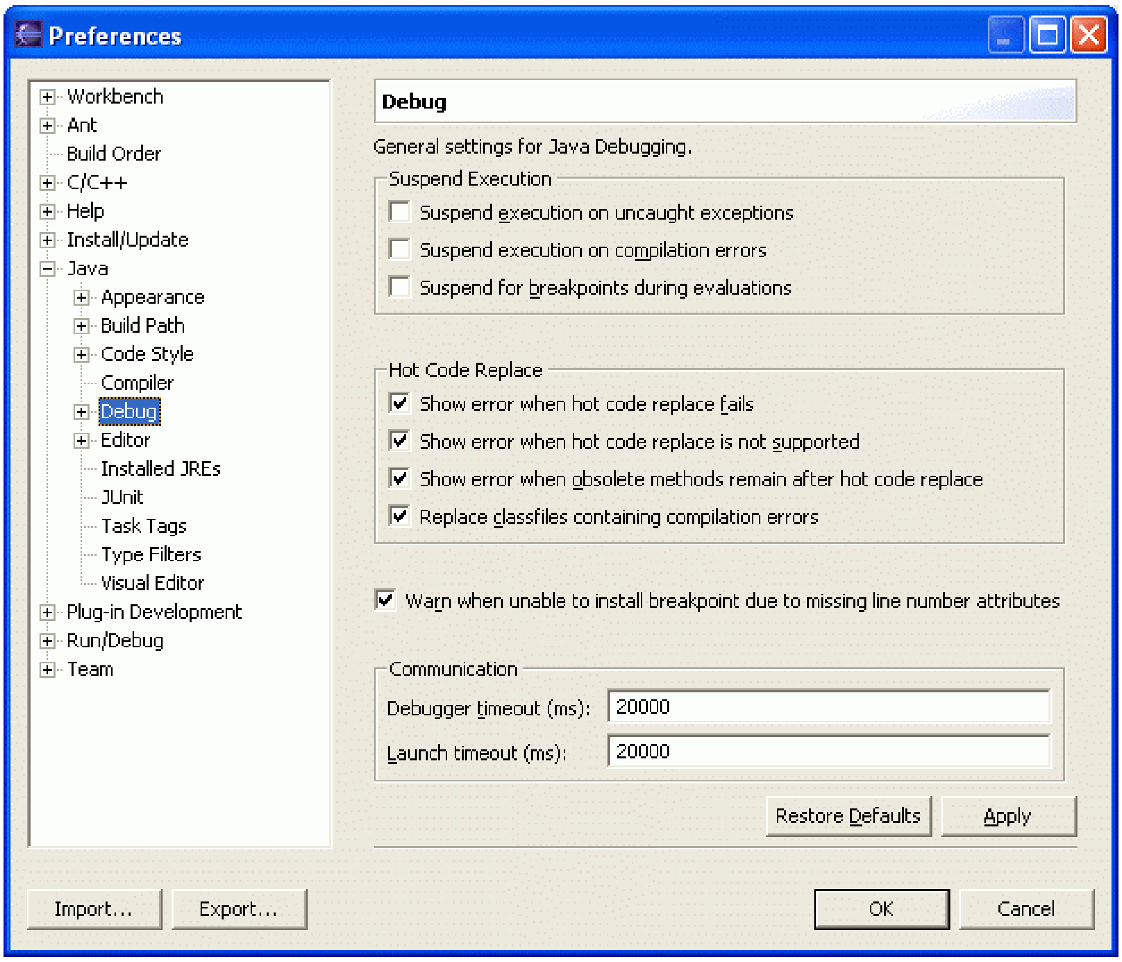
When you want to debug an existing application with the SDK, select a configuration you want to debug in one of the following ways:

- Select the class's .java file in the Package Explorer
- Select the .jad file (with MIDP applications) in the Package Explorer
- Select the project that includes the MIDlet in the Package Explorer
- Open the application in the editor

Add breakpoints to the application. Select to debug the configuration e.g. from *Run* | *Debug as* | *Nokia SDK*. The Debug perspective opens in the Eclipse.

To modify configuration settings, choose *Run* | *Debug...* The dialog for creating, managing, and running configurations opens, and you can perform similar actions to the ones described in the Chapter "Running an Application".

**Note : We recommend that you disable all Suspend Execution options from *Window* | *Preferences* | *Java* | *Debug*. We also recommend that you increase the debugger timeout (ms) setting to 20 000 when debugging with SDKs. (see Figure 52)**


Preferences

- Workbench
- Ant
- Build Order
- C/C++
- Help
- Install/Update
- Java
  - Appearance
  - Build Path
  - Code Style
  - Compiler
  - Debug
  - Editor
  - Installed JREs
  - JUnit
  - Task Tags
  - Type Filters
  - Visual Editor
- Plug-in Development
- Run/Debug
- Team

**Debug**

General settings for Java Debugging.

Suspend Execution
- [ ] Suspend execution on uncaught exceptions
- [ ] Suspend execution on compilation errors
- [ ] Suspend for breakpoints during evaluations

Hot Code Replace
- [✓] Show error when hot code replace fails
- [✓] Show error when hot code replace is not supported
- [✓] Show error when obsolete methods remain after hot code replace
- [✓] Replace classfiles containing compilation errors

[✓] Warn when unable to install breakpoint due to missing line number attributes

Communication

Debugger timeout (ms): 20000

Launch timeout (ms): 20000

Figure 52: Recommended settings for the Window | Preferences dialog

## 19.2 Using Nokia Developer's Suite from Eclipse

**Note : To be able to use these tools you must be in the Java Perspective in Eclipse. You can open the perspective as follows: *Window* | *Open Perspective* | *Java*.**

The functionalities of Nokia Developer's Suite 3.0 for J2ME™ can be accessed from Eclipse. You can access them from the *Tools* | *Nokia Developer's Suite* menu that includes the following options:

- Select your project and choose **New Class...** to create classes for the project.
- Select your project and choose **New Application Package...** to create a MIDlet suite (.jad and .jar files) or a Personal Profile application (JAR file).
- Select a JAD file in the Package Explorer view and then choose **Sign Application Package...** to sign a MIDlet.
- Select a JAD or JAR file in the Package Explorer view and then choose **Deployment...** to deploy a MIDlet or a Personal Profile application to a device.

- Choose **Configure Emulators...** to add or remove SDKs and to specify the SDK that Nokia Developer's Suite uses for creating an application package.
- You can start an application by selecting a JAD or JAR file and then selecting **Start Emulators...**
- Choose **Audio Converter...** to create a ring tone from a MIDI file.
- Select **DRM Editor...** to control how a phone user can use and re-distribute contentssuch as a MIDlet.
- Select **Web Services Client Tool...** to generate Java stub classes from WSDL files.
- Choose **Check for Updates...** to update Nokia tools and SDKs.
- Choose **Help...** to access the Nokia Developer's Suite online help.
- If you select **Provide Feedback to Nokia...** , you can send feedback to Nokia.
- Choose **Modules...** to view information about Nokia Developer's Suite modules.
- Choose **About...** to view license and version information.

### 19.2.1 NDS Console

NDS Console displays Nokia Developer's Suite for J2ME messages when you are using, e.g., the deployment function.

You can open the NDS Console view by choosing *Window* | *Show View* | *Other...* from the menu bar. Select Nokia Developer's Suite for J2ME \ NDS Console in the Show View dialog.

### 19.2.2 UI Designer

Nokia UI Designer is an additional Nokia Developer's Suite 3.0 for J2ME™ functionality that can be used to design MIDP User Interface layouts, tiled layers for games and screen flow diagrams. In Eclipse, Nokia UI Designer is an additional in-place editor for .java or .flw files.

To start the UI Designer, right-click one of the Form, Layer Manager class or Screen Flow Designer source files you have created in the Package Explorer. Choose *Open With* | *Nokia UI Designer* from the pop-up menu. Select UI Designer tab from the opened editor pane. You can switch between the editor and the UI Designer by clicking the tabs during development. It is also useful to open Nokia UI Designer Resources window (see Chapter 19.2.3: Nokia UI Designer Resources).

To create a new screen flow diagram, select *File* | *New* | *Other...* | *Nokia Developer's Suite for J2ME* | *Flow Diagram file*

### 19.2.3 Nokia UI Designer Resources

The Nokia UI Designer Resources view is used with the UI Designer editor. This view displays the items and resources that have been added to the Form, Layer Manager or Screen Flow Designer.

To open the Nokia UI Designer Resources view, choose *Window* | *Show View* | *Other...* from the menu bar. Select Nokia Developer's Suite for J2ME | Nokia UI Designer Resources in the Show View dialog that opens.

## 19.3 MIDlet Examples

Nokia offers a few example MIDlets for MIDP 1.0 and MIDP 2.0 with Nokia Developer's Suite 3.0 for J2ME™.

The following MIDP 1.0 MIDlet examples are located in the NDS_INST_DIR\examples directory:

- Boids
- Chat
- MediaSampler

The following MIDP 2.0 MIDlet examples are located in the NDS_INST_DIR\examples directory:

- Hawk
- Spaceworld

The following additional examples are located in the NDS_INST_DIR\examples directory:

- TrafficLight (UI Designer CustomItem example)
- WebServiceSample

To use the example MIDlets,

Choose *File* | *New* | *Project...* in the Java Perspective to create a new project. Then choose *Java* | *MIDP Project (Nokia SDK Plug-in)* and click **Next** to continue.

Browse for one of example directories, e.g., NDS_INST_DIR\examples\Hawk to access the project contents and enter the directory name, e.g. Hawk, for the project. Click **Next** to continue.

Select the SDK you want to use for running the MIDP project. Click **Next** to continue.

Make sure that the source folder on build path and the default output folder are correctly set (e.g. Hawk/src and Hawk/classes) on the Source tab. (Later you can add the source folders to the Source Tab by choosing *Project* | *Properties* | *Java Build Path*). Click **Finish** to complete.

To run the MIDlet with an SDK, select a .jad file in the Package Explorer and choose *Run* | *Run As* | *Nokia SDK* from the menu bar. The SDK opens with the MIDlet.

## References

[1] http://java.sun.com

[2] http://java.sun.com/products/midp/

[3] http://java.sun.com/products/personalprofile/

[4] http://java.sun.com/products/cldc/

[5] http://java.sun.com/products/cdc/

[6] Mobile Information Device Profile 2.0 (Final release)
http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html

[7] Nokia Developer's Suite for J2ME™ Installation and Configuration Guide.
http://www.forum.nokia.com

[8] J2ME Web Services Specification (Final release)
http://jcp.org/aboutJava/communityprocess/final/jsr172/index.html

[9] WS-I Basic Profile 1.0
http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

# Build Test Sell

Developing and marketing mobile applications with Nokia

### Go to Forum.Nokia.com

Forum.Nokia.com provides the tools and resources you need for content and application development as well as the channels for sales to operators, enterprises, and consumers.

Forum.Nokia.com

### Download tools and emulators

Forum.Nokia.com/tools has links to tools from Nokia and other industry leaders including Borland, Adobe, AppForge, Macromedia, Metrowerks, and Sun.

Forum.Nokia.com/tools

### Get documents and specifications

The documents area contains useful white papers, FAQs, tutorials, and APIs for Symbian OS and Series 60 Platform, J2ME, messaging (including MMS), and other technologies. Forum.Nokia.com/devices lists detailed technical specifications for Nokia devices.

Forum.Nokia.com/documents
Forum.Nokia.com/devices

### Test your application and get support

Forum Nokia offers free and fee-based support that provides you with direct access to Nokia engineers and equipment and connects you with other developers around the world. The Nokia OK testing program allows your application to enjoy premium placement in Nokia sales channels.

Forum.Nokia.com/support
Forum.Nokia.com/ok

### Market through Nokia channels

Go to Forum.Nokia.com/business to learn about all of the marketing channels open to you, including Nokia Tradepoint, an online B2B marketplace.

Forum.Nokia.com/business

### Reach buyers around the globe

Place your applications in Nokia Tradepoint and they're available to dozens of buying organisations around the world, ranging from leading global operators and enterprises to regional operators and XSPs. Your company and applications will also be considered for the regional Nokia Software Markets as well as other global and regional opportunities, including personal introductions to operators, on-device and in-box placement, and participation in invitation-only events around the world.

Forum.Nokia.com/business