JBuilder™ Mobileset 개발자 가이드

Chapter 1

Introduction

Jbuilder 모바일셋 개발자 가이드는 여러분이 어떻게 Jbuilder의 통합 개발 환경을 이용하여 J2ME를 사용하는 모바일 디바이스용 MIDP 애플리케이션을 생성하고 아카이브 하는지를 설명하고 있다.

이 문서는 프로젝트와 파일들을 어떻게 다루고, 유저 인터페이스를 어떻게 제작하는지, 어떻게 컴파일하고, 실행하고, 디버깅하는지를 알려주며, 여러분의 다양한 MIDP 애플리케이션을 아카이브 하는지를 알려주고 있다.

모바일셋 개발자 가이드는 단순히 Jbuilder를 이용하는 개념적인 측면에서 설명을 다루고 있으므로, 좀 더 자세히 Jbuilder IDE 에 관하여 알고 싶다면, 핵심 Jbuilder 문서들을 참조하기 바란다.

이 문서는 단지 여러분에게 J2ME와 자바 프로그래밍을 어떻게 하는지를 보여주는 역할을 하게 될 것이다. 그러나 이러한 정보들에 대한 적절한 리소스의 링크가 Introduction 마지막 부분에 있으니 이를 잘 참조하기 바란다.

주의 이 문서에 보이는 에뮬레이터, 휴대폰 UI 이미지들은 예제를 위한 목적으로 작성된 것이므로 특정 디바이스에 동일하게 적용하지 않기를 바란다.

목 차

Jbuilder 모바일셋 개발자 가이드는 다음의 장들로 구성되어 있다.

해 보게 된다.

• **Appendix B, "Tutorial: StopWatch MIDlet"**

J2ME MIDP 애플리케이션 개발에 있어서 몇 가지 기본적인 개념들을 탐색해 보게 된다. 여러분에게 Canva를 그리는 법, 색상을 세팅하는 법, 사각형에 색을 채우는 법, 폰트를 세팅하는법 등등에 관해 알려 주게 된다. 또한 다중 Displayables을 이용하는 MIDlet을 생성하는 예제를 보여 준다. 마지막 섹션에서는 여러분이 어떻게 저장하고 RMS 데이터베이스로부터 필요한 정보들을 어떻게 추출하는지를 보여준다.

리소스 링크

• Java™ 2 Platform, Micro Edition (J2ME™ Platform), at http://java.sun.com/j2me/

• CLDC and the K Virtual Machine (KVM), at http://java.sun.com/products/cldc/

• Mobile Information Device Profile (MIDP), at http://java.sun.com/products/midp/

• J2ME™ Connected Limited Device Configuration (CLDC), at http://www.sun.com/software/communitysource/j2me/cldc/

• JDC J2ME™ Tech Tips, at http://developer.java.sun.com/developer/J2METechTips/

• Java™ 2 Platform, Micro Edition, Frequently Asked Questions, at http://java.sun.com/j2me/faq.html

• Articles on Wireless Technologies, at http://developer.java.sun.com/developer/technicalArticles/wireless/

• *Sams Teach Yourself Wireless Java with J2ME in 21 Days* by Michael Morrison, at http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0672321424&vm=&from=SWP279

• *Core J2ME Technology and MIDP* by John W. Muchow, at http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0130669113&vm=&from=SWP279

• *Instant Wireless Java™ with J2ME™* by Paul Tremblett, at http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0072191759&vm=&from=SWP279

• *Mobile Information Device Profile for Java™ 2 Micro Edition (J2ME):Professional Developer's Guide* by C. Enrique Oritz, Eric Giguere, at http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0471034657&vm=&from=SWP279

• Forum Nokia Java Developer Section, at http://americas.forum.nokia.com/java/default.asp

JBuilder™ Mobileset 개발자 가이드

문서번호 : IMOBILE-JBM001

Chapter 2

Mobileset 3.0의 새로운 점

Mobileset 3.0은 몇 가지 다음 분야에서 새로운 특징과 향상된 점을 가지고 있습니다.

- 벤더 디바이스 지원
- 설치 업데이트
- 단일화된 에뮬레이터 인터페이스 지원
- MIDP 스크린 매니저
- 클래스 파일

벤더 디바이스 지원

- Sun J2ME Wireless Toolkit 1.0.3과 1.0.4
- Nokia Developer's Suite for Java™ 2 Platform, Micro Edition
- Siemens Mobility Toolkit(SMTK)
- Sprint PCS Wireless Toolkit for the Java Platform

설치 업데이트

여러분이 모바일셋의 풀 설치 동안 SUN의 J2ME Wireless Toolkit 1.0.4 가 여러분이 Jbuilder를 설치하였던 디렉토리 내의 J2mewtk 라는 폴더에 설치되게 된다. J2ME Wireless Toolkit JDK 가 모바일셋과 함께 설치될 때 자동으로 여러분에 맞게끔 세팅되게 된다. 만약 여러분이 이를 각각 분리하여 설치하고자 한다면, 여러분은 수동으로 이를 세팅해 주어야 하며, Jbuilder 내의 JDK 다이얼로그 박스를 설정해 주어야만 한다. (Tools|Configure JDKs).

단일화된 에뮬레이터 인터페이스 지원

모바일셋 3은 UEI(Unified Emulator Interface) 스펙을 지원하는 J2ME JDK를 위한 UEI 이용을 지원해 준다. UEI 는 여러분이 다중 써드파티 모바일디바이스를 지원하고자 할 때, 호환가능한 디바이스를 위해 일반적인 라이브러리 API들을 이용함으로써, 여러분의 개발을

단순화할 수 있게 해 준다. The Sun J2ME Wireless Toolkits 1.0.3과 이후 버전은 UEI를 지원한다. UEI 인터페이스로의 변화는 선택된 에뮬레이터 인터페이스를 기반으로 하고, 서로 다른 라이브러리들이 프로젝트를 컴파일 하는데 사용될 것이라는 것을 의미한다. 여러분이 UEI를 지원하는 J2ME JDK를 설정할 때, 여러분은 JDK 다이알로그 박스 설정 창의 Micro 탭에서 Default Color Phone와 같은 타겟 디바이스를 명세하게 된다.

여러분이 타겟 디바이스를 선택할 때, 라이브러리 API는 여러분의 프로젝트에서 선택한 JDK에 자동으로 탑재되게 된다. 기타 동일한 라이브러리 API를 공유하는 써드 파티 디바이스들은 여러분의 애플리케이션 내에서 사용할 수 있게 된다.

프로젝트 속성에서, 여러분은 타겟 디바이스와 관련된 일반적인 라이브러리 API에 의해 지원되는 디바이스들의 목록에서 여러분의 프로젝트를 테스팅할 특정 에뮬레이터 디바이스를 선택할 수 있다.

런타임 속성 다이알로그 박스에서 에뮬레이터 디바이스는 프로젝트 속성 다이알로그 박스에 있는 Run 페이지에 여러분의 프로젝트에 대한 런타임 설정을 선택함으로써 접근할 수 있다. (Project|Project Properties|Run)

JDK 다이알로그 박스 설정에서 마이크로 탭에 있는 타겟 디바이스에 의해 정의된 동일 라이브러리들을 공유하는 디바이스들은 타겟 디바이스 리스트에 보여지게 된다. 만약 희망하는 디바이스가 리스트되지 않으면, Tools|Configure JDKs 에 가서 타겟 디바이스를 선택하면 된다.

MIDP 스크린 매니저

MIDP 스크린 매니저는 여러분이 MIDlet과 관련하여 MIDP Displayable을 다루고 이해할 수 있도록 도움을 주게 된다. MIDP 스크린 매니저는 주어진 MIDlet 과 관련된 displayable에 관한 개념적인 모습들을 제공하게 된다.

- MIDP 스크린 매니저는 편집기 안에서 오픈된 MIDlet에 스크린 탭을 클릭함으로써 접근할 수 있다.
- MIDP 스크린 매니저는 어떤 MIDP Dispalable이 오픈된 MIDlet 과 관련되어 있는지를 결정하기 위해 여러분의 프로젝트에 .java 파일들을 해부해 가면서 편집기의 컨텐츠 Pane 에서 관련된 Displayable과 MIDlet의 표현을 해 나가게 된다. 서로 다른 형태의 Displayable(Alert, Form, List, TextBox, Canvas)들은 스크린 매니저와 스트럭쳐 페인에 있는 아이콘에 의해 표시된다.
- MIDlet과 Displayable 사이의 호출(Call)은 배열(Arrow)에 의해 표현된다. 배열은 하나의 MIDP Displayable 에서 다른 Displayable 로의 흐름을 나타내어 주며 배열은 주어진 Displayable 맨 아래에서부터 시작하게 된다.
- 여러분은 Displayable 간의 관계를 보다 보기 좋게 하기 위해 스크린 매니저에서 MIDlet과 MIDP Displayable들을 다시 정리할 수 있다. 여러분의 스크린 매니저

레이아웃은 하나의 Jbuilder 세션에서 또 다른 Jbuilder 세션으로 유지시켜 나갈 수 있다.

- 스크린 매니저에서 하나의 Displayable을 더블 클릭함으로써 편집기에 Displayable에 대한 소스 파일을 열어 볼 수 있다. 여러분은 Search 메뉴를 이용하거나 툴바의 Back 버튼을 눌러서 Displayable에 대한 소스 파일이 열린 이후에 스크린 매니저로 돌아 갈 수 도 있다.
-

Note 스크린 매니저가 데이터를 보여주기 위해서는 Command 파라미터는 반드시 일정해야 한다. (label, screen type, and priority) 그렇지 않으면, 스크린 매니저는 "<unparsable>"을 보이게 된다. 만약 다른 파라미터들이 일정하지 않으면 그들은 커맨드 인스펙터에서 "<unparsable>"이라고 디스플레이 하게 된다.

Class file obfuscation

MobileSet 3은 여러분의 클래스 파이들을 Jbuilder SE 와 Enterprise에서 아카이브 과정의 일부분으로 흐릿하게 만들 수 있다. Jbuilder는 obfuscator RetroGuard v. 1.1을 지원하며 모바일셋3에서 이를 이용할 수 있게 해 준다.

Note Implementation에 따라 RetroGuard 를 포함하는 몇몇 obfuscator들은 여러분의 MIDlet Suite 클래스 파일에 의해 언급된 모든 클래스들을 로드하게 될 것이다.
만약 언급된 클래스들이 네이티브 메소드를 가지고 있다면, obfuscator들은 이러한 네이티브 메소드와 관련된 공유 라이브러리를 로드하려고 할 것이다. 이는 MDIP API 클래스들이 네이티브 메소드를 이용하기 때문에, 공유 라이브러리들은 해석되지 못하고 이런 obfuscator들은 여러분의 MDIlet Suite 을 흐릿하게 만들어 주지 못할 것이다.
이런 문제들에 대해 알아보기 위해서는 MIDP API (emptyapi.zip)의 버전에 대해 알아볼 필요가 있으며, 이 라이브러리들을 obfscuator 클래스 경로에 추가시켜 주어야 한다. emptyapi.zip은 여러분이 Mobileset 과 함께 J2ME Wirelsess Toolkit 을 설치할 때, <Jbuilder home>\j2mewtk\wtklib에서 찾아보거나 SUN의 웹사이트에서 "Obfuscating a J2ME[tm] MIDlet suite" page (http://wireless.java.sun.com/midp/questions/obfuscate/)을 찾아 다운로드 할 수 있다.
emptyapi.zip을 Jbuilder의 obfscuator 클래스 경로에 추가시키기 위해서는, Tools|Configure Obfuscator에서 Obfuscator를 선택하고, 클래스 탭을 선택하고 Add 를 눌러 emptyapi.zip 파일을 찾아 주면 된다.

JBuilder™ Mobileset 개발자 가이드

문서번호 : IMOBILE-JBM001

Chapter 3

Mobileset 개발 환경

개 요

JBuilder Mobileset은 이미 설치되어 있는 JBuilder 위에 재 설치되는 오픈 툴이다. Mobileset은 JBuilder와 함께 구동 되나, JBuilder Personal Edition에서는 몇몇 기능들(JDK Switching, Archive Builder)이 구동 되지 않는다.

Mobileset에 대한 JBuilder의 운영 환경의 설정은 추가적인 Tab과 기존의 다이얼로그 박스에 있는 옵션, 두개의 Wizard(MIDlet Wizard, MIDP Displayable Wizard)을 제외하고 일반적인 Java 개발 환경에 대한 설정과 동일할 뿐 아니라 Jbilder Professional, Enterprise, Archive Builder에서는 MIDlet Suite와 이와 관련된 JAD 파일을 생성할 수 있다.

여러분이 JBuilder Mobileset을 설치하게 되면, Mobileset에 관련된 문서들은 기존의 JBuilder 문서에 같이 설치되게 되며, Help Viewer의 목차 마지막 부분에 관련 내용이 추가되게 된다. 도움말이 필요하면, F1 키를 누르거나 다이얼로그 박스 안에 있는 Help 버튼을 누르기 바란다.

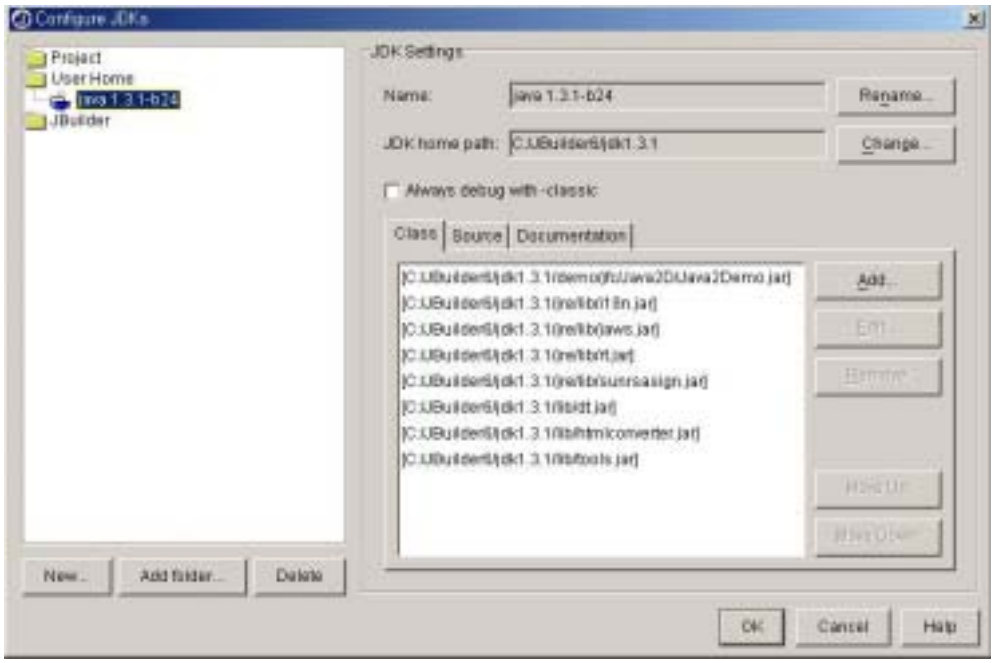
여러분은 View | Browse Class를 선택함으로써 클래스에 관련된 참고 문서들을 열어볼 수 있다.

JBuilder는 이미 기본적으로 설치되고 설정되어 있는 JDK를 가지고 있으나, 여러분이 새로운 프로젝트에 추가하고자 하는 새로운 JDK를 JBuilder에 추가할 수 있습니다. 자신의 PC에 새로운 JDK를 추가로 설치하고자 합니다면, 다음과 같이 설치하기 바랍니다.

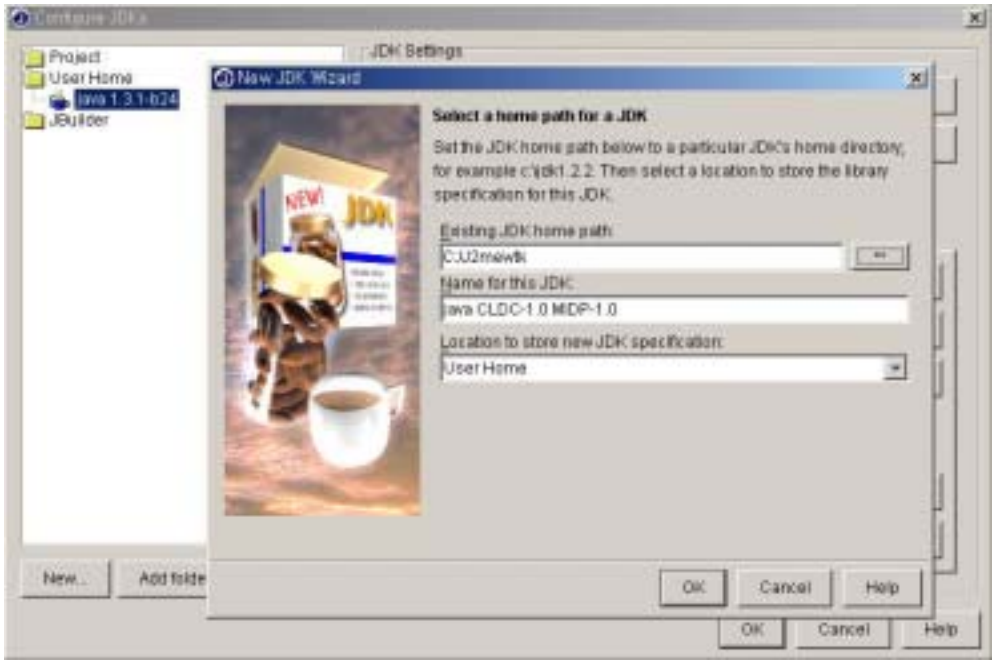
새로운 JDK 세팅

여러분은 JBuilder Professional과 Enterprise에서 하나 이상의 JDK를 정의해주고 프로젝트 내에서 JDK의 교환이 가능하다.

1.새로운 MIDP/CLDC JDK를 추가하기 위해 Tools | Configure JDKs를 선택하여 JDK 설정 다이얼로그 박스를 연다.



2. New JDK Wizard을 열기 위해 New를 클릭합니다.

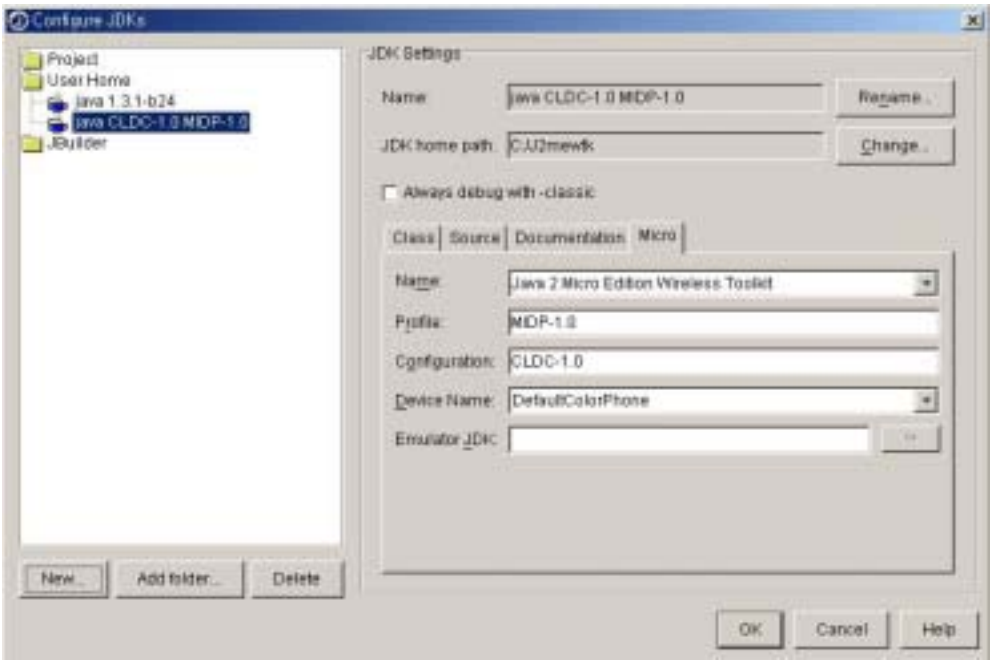


3.새로운 JDK Wizard 안에 있는 생략 버튼을 눌러 JDK가 설치되어 있는 곳을 찾습니다.

4.만약 JDK의 이름을 바꾸고자 합니다면 이름을 변경합니다.

5.새로운 JDK Spec을 저장하기 위한 장소를 선택하고자 합니다면, Drop-Down 메뉴를 클릭합니다.

6.OK를 누르면, 새로운 JDK 가 추가되어 다이얼로그 박스에 나타나게 됩니다. 오른쪽에 새로 추가된 JDK에 관련된 정보와 기반이 되는 CLDC와 MIDP이 버전을 보여주는 Micro 탭이 추가되었음을 확인할 수 있습니다.



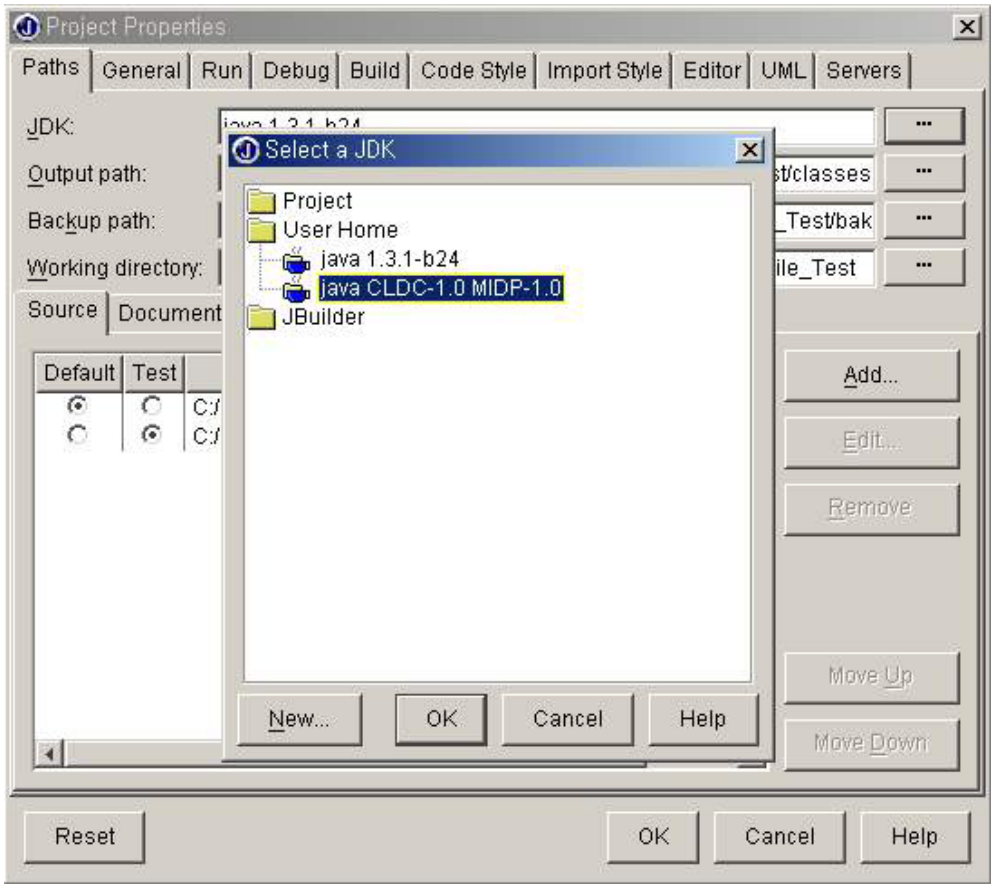
7. OK를 눌러 Configure JDKs 다이얼로그박스를 닫습니다.

※추가정보를 원한다면, http://www.borland.com/JBuilder/download/addons.html를 방문하여, JBuilder에 관련된 문서 중 *Building Application with JBuilder*를 참고하기 바랍니다.

위의 방법들은 Jbuilder 6에서 새로운 JDK를 세팅하는 방법에 대한 것입니다. 그러나, Jbuiler 7에서는 자동으로 세팅되어 있으므로, 한 단계씩 따라 하면서 제대로 세팅되어 있는지 확인해 보시기만 하면 됩니다.

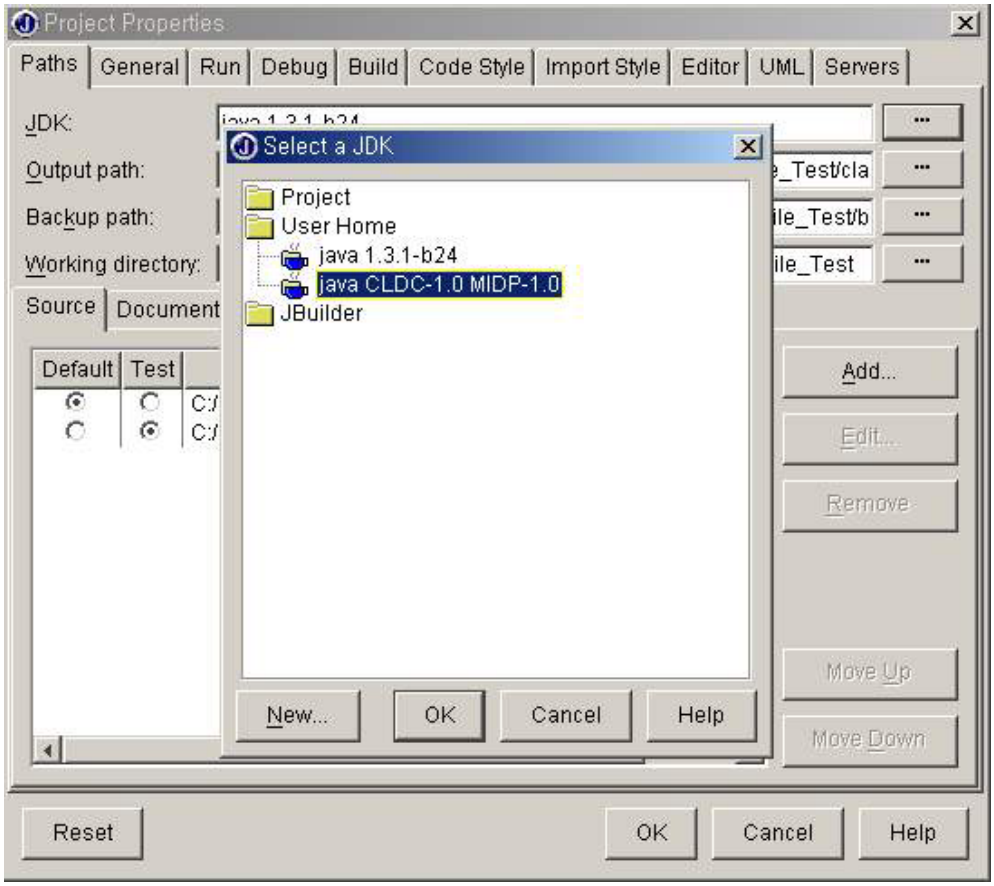
■ 새로운 프로젝트를 위한 MIDP/CLDC JDK 지정

여러분이 새로운 프로젝트를 생성할 때 MIDP/CLDC JDK를 지정하기 위해 Project | Project Properties를 선택하여 새로운 JDK를 지정하면 됩니다.



■ 기존의 프로젝트에 대한 MIDP/CLDC JDK 지정

기존의 프로젝트에 MIDP/CLDC JDK를 지정하기 위해서는 Project | Project Properties를 선택하여 Paths 페이지에서 JDK 필드의 생략 버튼을 눌러 JDK를 선택합니다.

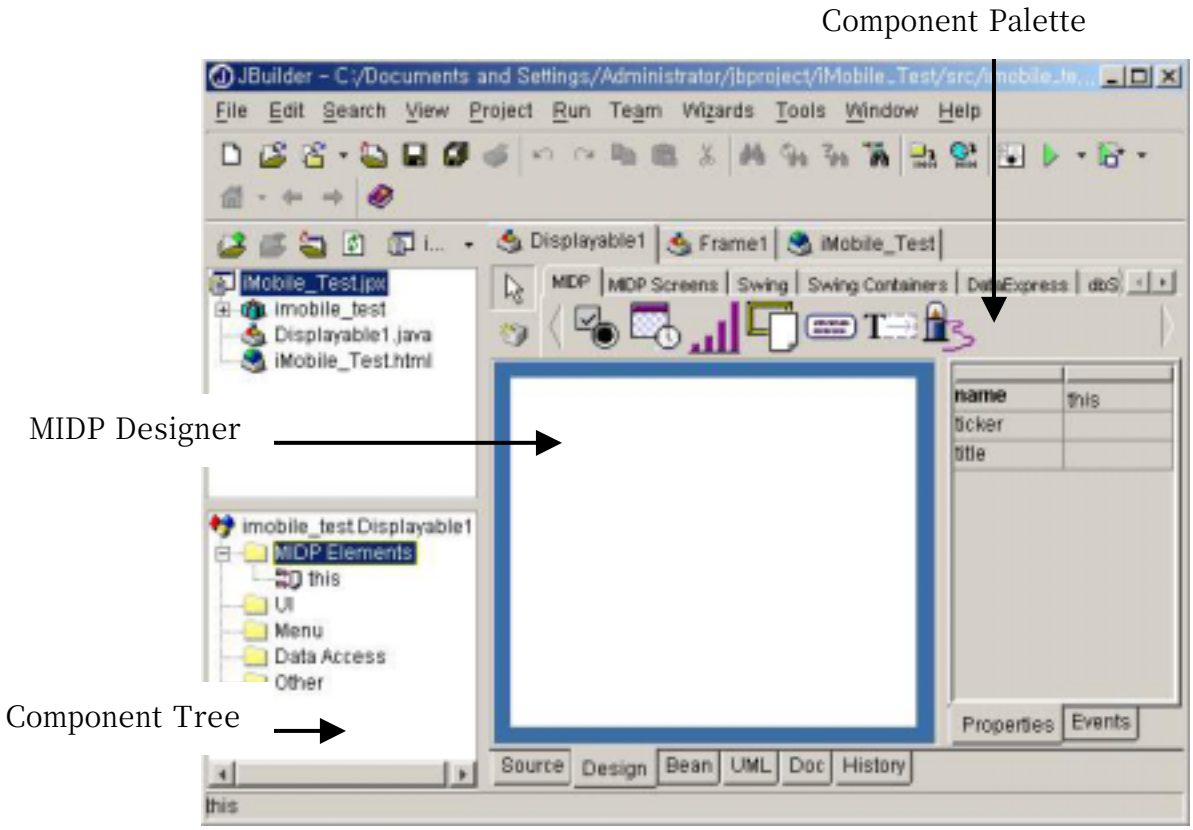


■ **모든 프로젝트에 대해 Default MIDP/CLDC JDK 지정**

모든 프로젝트에 대해 Default MIDP/CLDC JDK를 지정하기 위해서는 Project | Default Project Properties를 선택한 후, Paths 페이지에서 JDK 필드에 해당하는 생략 버튼을 누른 후 JDK를 선택합니다.

MIDP User Interface 디자인

여러분은 MIDP 디자이너를 이용하여 JBuilder에서 비쥬얼하게 사용자 인터페이스를 디자인할 수 있습니다. MIDP 디자이너를 열기 위해서는 AppBrowser에 디스플레이 가능한 파일을 열어 Design 탭을 클릭하면 됩니다



Important

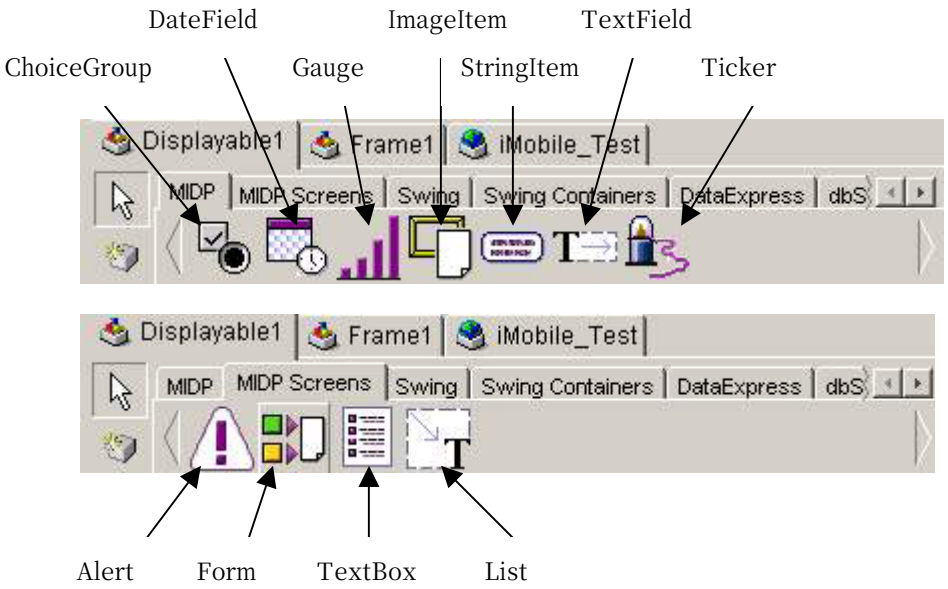
MIDP 디자이너는 단지 Displayable 클래스를 디자인 하기 위해 고안된 것이며, 가상적으로 MIDlet 객체를 디자인 하는데 이용되지는 않는다. Displayable 클래스는 일반적인 Java application에 있는 Frame이나 Panel 컨테이너처럼 디바이스의 스크린을 보여주기 위한 UI를 포함하고 있습니다.

여러분은 JBuilder MIDlset wizard나 MIDP Displayable wizard를 이용하여 여러분의 MIDlet을 위해 Displayable 클래스를 생성할 수 있습니다. 한번 여러분이 Displayable을 생성하면 여러분은 MIDP 디자이너를 이용하여 사용자 인터페이스를 비쥬얼하게 커스터마이징할 수 있습니다.

만약 여러분이 여러분의 MIDlet에 추가적인 Displayable 클래스들을 추가하고자 합니다면, MIDP Displayable Wizard를 이용하여 여러분의 프로젝트에 클래스들을 쉽게 추가할 수 있습니다. 새로운 Displayable 클래스들을 추가한 후에, 여러분은 그들을 메인 MIDlet 클래스에 Hook 해줘야 합니다.

MIDP UI 컴포넌트들은 실질적인 JavaBeans이 아니기 때문에 MIDP 디자이너는 비쥬얼하게 컴포넌트들과 함께 운용하기 위해 표준적인 UI 디자이너로부터 변경된 것입니다. 여러분이 디자이너에 MIDP 컴포넌트를 추가하면, MIDP Elements라 불리는 폴더 아래에 컴포넌트 트리에 보이게 됩니다.

JBuilder Mobileset은 컴포넌트 팔레트에 이미 MIDP UI 컴포넌트가 실려 있습니다.



■ MIDP UI 디자이너에 관한 추가 정보를 원하다면, CH6의 “MIDP UI 생성” 부분을 참조하기 바랍니다.

MIDP Application 의 컴파일, 실행, 디버깅

JBuilder에서 MIDP application의 컴파일, 실행, 디버깅은 JBuilder에서 일반적인 자바 프로그램의 컴파일, 실행, 디버깅 방법과 동일하다.

Chapter 4

Creating and Managing MIDP Project

JBuilder를 이용한 개발은 Project 파일 개념에 맞추어져 있습니다. Project 파일은 해당 프로젝트 안의 모든 파일의 목록과 프로젝트의 속성을 모두 내포하고 있으며, 프로젝트 템플릿, 기본 경로, 클래스 라이브러리, 연결 설정까지 모두 포함하고 있습니다.

JBuilder는 이처럼 여러분이 프로젝트를 Load, Save, Build, Run 할 때의 모든 정보들을 이용하게 됩니다. 프로젝트 파일은 여러분이 프로젝트의 속성을 변경하거나 세팅하거나 파일을 제거하거나 추가하기 위해 JBuilder 개발 환경을 이용할 때마다 변경되게 됩니다. 여러분은 이 모든 것들을 프로젝트 Pane에서 찾아 볼 수 있습니다.

여러분이 MIDP 응용 프로그램을 생성할 때 여러분은 MIDP Wizard(MIDlet wizard, MIDP Displayable wizard)에 접근하기 전에 우선 J2ME MIDP/CLDC JDK를 이용하게 될 프로젝트 파일을 생성해야 합니다.

새로운 프로젝트를 시작하기 위해서는 파일, 디렉토리, 경로 등을 JBuilder의 프로젝트 wizard를 이용하여 자동으로 생성하면 됩니다.

■ 프로젝트 Wizard를 이용한 새로운 프로젝트 생성

중요 사항

프로젝트 Wizard를 열기 위해서는 JBuilder 메뉴에서 File | New Project를 선택하면 됩니다. 여러분은 이미 설치된 J2ME MIDP/CLDC JDK를 가지고 있고, 새로운 MIDP 프로젝트를 생성하기 전에 JBuilder에서 설정을 이미 마쳤으므로 새로운 설정을 적용할 필요는 없습니다. (JDK 세팅 참조 – 페이지 3-2)

프로젝트 생성, 프로퍼티 세팅에 관한 사항은 Appendix에서 설명된 Tutorial을 참고하기 바란다.

Chapter 5

Building MIDP applications

컴파일링

Jbuilder에서 MIDP 클래스를 컴파일하는 것은 한가지 예외를 제외하고 일반적인 자바 개발 환경에서와 동일하다. J2ME개발에 있어 클래스는 실행하기 전에 반드시 우선 검증되어야 한다. Jbuilder는 컴파일 시 여러분의 프로젝트 내의 모든 클래스를 우선 검증하게 된다.

여러분의 프로젝트를 컴파일 하기 위해서는 Make 또는 Rebuild 를 이용하시면 되고 실행할수 있는 곳은 세 곳이 있다.

1. 프로젝트 메뉴
2. 툴바
3. 프로젝트 페인 안에 선택된 파일위에 마우스를 놓고서 오른쪽 클릭

컴파일링 타입

통합개발환경(IDE) 하에서 컴파일을 위한 커맨드는 다음의 사항들을 내포하고 있다.

- Make 커맨드(Project|Make Project)는 선택된 노드와 변경되어 import 된 파일들을 컴파일한다. 편집/재컴파일 과정에 있어 일반적인 반복을 되풀이 할 경우, Make가 적당한 커맨드이다.
- Rebuild 커맨드(Project|Rebuilde Project)는 모든 프로젝트 소스파일과, 변경되었다 하더라도 import된 파일들을 모두 컴파일한다. 그래서 Make커맨드보다는 시간이 오래 걸린다.
- Run 커맨드 (Run|Run Project)는 여러분의 애플리케이션을 실행시킨다.

컴파일이 가능한 것

다음과 같은 프로그램의 부분들이 컴파일 될 수 있다.

- 전체 프로젝트
- 패키지
- 자바 파일

Jbuilder에서 컴파일 되는 파일들의 위치를 알아 보려면, Jbuilder 입문서인 “How Jbuilder constructs paths”와 “Where are my files?” 를 참조하기 바란다.
Jbuilder 에서의 컴파일에 관한 더 많은 정보를 원한다면, “Building Application with Jbuilder” 안에 있는 “Compiling Java programs”를 참조하기 바란다.

실행

MIDP에 유연한 디바이스들은 자바 애플리케이션 디스크립터(JAD : Java Application Descriptor)파일과 JAR 파일을 활용하는 자바 VM을 지원하게 된다. JAD 파일은 JAR 파일 내부의 MIDlet의 이름을 명명하고 MIDlet에 관해 필수 정보나 추가 정보들을 명기해준다. 또한, JAR 파일의 크기와 런타임시 MIDlet을 통과하는 파라미터들을 포함하게 된다. JAD 파일은 단지 MIDlet suite 내부의 다중 MIDlet의 이름을 명기해 주기만 한다.
여러분이 Jbuilder 안에서 MIDlet 을 테스트 하게 될 때, 여러분은 서로 다른 프로젝트의 각각의 MIDlet을 실행하거나, 다중의 MIDlet을 선택하여 동시에 실행하기도 하고, MIDlet suite의 다중 MIDlet을 테스트 하기 위해 JAD 파일로부터 한번에 실행하기도 할 것이다.

MIDlet 실행

MIDlet 파일의 실행

Jbuilder 내부에서 여러분의 애플리케이션을 실행하기 위해서는 툴바에서 Run을 누르거나 F9을 누르시면 실행됩니다. 만약 여러분의 프로젝트가 다중 MIDlet을 가지고 있다면, 여러분은 MIDlet 모두를 선택하여 동시에 Run 시킬 수 있습니다. 이 때 다중으로 선택하기 위해서는 프로젝트 페인에서 여러가지 MIDlet을 shift 키를 누른 상태에서 마우스로 선택하시면 됩니다.

Chapter 6

Creating a MIDP user interface

Appendix 참조

문서번호 : IMOBILE-JBM001

JBuilder™ Mobileset 개발자 가이드

문서번호 : IMOBILE-JBM001

Chapter 7

MIDP 데이터베이스 프로그래밍

개요

다양한 handheld 기기의 저장 용량은, 작은 메모리에 의해 극도로 제한되며, 각 기기마다 필요 용량은 매우 다르다. 이러한 이유로, 데이터를 저장하기 위한 J2SE에 쓰이는 일관적인 저장 구조는 J2ME MIDP 기기에 부적당하다.

MIDlet을 종료할 때, 객체나 객체의 상태가 파괴되는 것을 막으려면, 데이터베이스 정보를 저장하기 위한 비-휘발성의 장소가 필요하다. J2ME MIDP는 레코드 저장소로 데이터를 저장하고 받기 위한, Record Management System(RMS)라는 간단한 레코드 지향의 데이터베이스를 사용합니다. RMS는 MIDlet이 종료된 후에도 일관적인 레코드의 집합인 MIDP 레코드를 저장합니다. 다시 MIDlet을 실행한 후에 이 레코드 저장소에서 정보를 얻어낼 수도 있습니다.

RMS 사용하기

RMS를 사용하기 위해서는 javax.microedition.rms 패키지를 임포트합니다 .J2ME RMS 프로그래밍 하는 방법에 관해서는 이 문서가 제공해 주는 범위 이상의 내용이다. 그러나 Mobileset의 StockTracker 샘플에서 RMS를 사용하는 예제가 제공되고 있습니다. Appendix B, “Tutorial: StopWatch MIDlet” 를 참조합니다. MobileSet 샘플들은 <JBuilder>/samples/Mobile 디렉토리에서 볼 수 있습니다. RMS에 관한 추가적인 리소스의 리스트는 아래와 같습니다.

• API documentation for the javax.microedition.rms package.

• “MIDP Database Programming using RMS: a Persistent Storage for MIDlets”, at http://developer.java.sun.com/developer/technicalArticles/wireless/midpdatabase/

• “Record Management System Basics”, in the J2ME Tech Tips at http://developer.java.sun.com/developer/J2METechTips/2001/tt0220.html#tip2

• “Using Record Stores Efficiently”, at
http://developer.java.sun.com/developer/J2METechTips/2001/tt0319.html#tip1
• “Enumerating, Filtering and Sorting MIDP Record Rtores”, at
http://developer.java.sun.com/developer/J2METechTips/2001/tt0622.html#tip2

Chapter 8

MIDlets 아카이브

기기에 MIDlet을 배포하기 전에 반드시 JAR 파일로 클래스들을 묶어야 하며 JAR 파일 안의 manifest 파일을 변경해야 합니다. Manifest 파일은 MIDlet 프로그램과 JAR 파일 안의 각 MIDlet 파일에 관한 특정한 속성들을 포함해야 합니다.
또한 반드시 JAR 파일의 내용을 설명하는 외부의 JAD(Java Application Descriptor) 텍스트 파일을 생성해야 합니다. 이 파일은 요구되거나 선택적인 MIDlet 속성을 포함하고 실행시에 MIDlet으로 파라미터를 전송합니다.

• 만일 JBUILDER Personal을 사용하고 있습니다면, Nokia를 제외한 모든 JDK를 위해 명령 라인에서 수동으로 이러한 파일들을 생성하여야만 합니다. JAR과 JAD 파일의 생성에 관해 더 자세한 정보를 알고 싶다면 8-2페이지의 "Archiving MIDlets from the command line"을 보라.
만일 Nokia JDK를 사용하고 있습니다면, JAR과 JAD파일을 생성하기 위해 J2ME 의 Nokia 개발툴을 사용할 수 있습니다. J2ME를 위한 Nokia 개발툴은 Tools 메뉴에서 확인할 수 있습니다. 이것들을 사용하는 것에 관해서는 Nokia 도움말 문서를 참조하라.
• 만일 JBUILDER Professional 이나 Enterprise를 사용하고 있습니다면, 세가지 파일 모두를 자동적으로 생성해주는 Archive Builder를 사용할 수 있습니다. 이 마법사의 각 단계들은 필요한 정보들을 기재하게 할 것입니다. 페이지 8-7의 "Archiving MIDlets with the Archive Builder" 을 보라.
중요 생성한 JAR 파일은 반드시 사전 검증된 클래스만을 포함해야 합니다. 만일 JBUILDER에서 Archive Builder로 JAR 파일을 생성합니다면, 이는 자동으로 이루어질 것입니다. 만일 jar을 이용하여 JAR 파일을 생성합니다면, preverifier에 의해 진행된 클래스들만을 포함하도록 해야 할것입니다. 일반적인 JAR 파일과 Manifest 파일에 관하여 더 자세한 정보를 원합니다면 "Deploying Java programs" in *Building Applications with JBuilder*. 안의 "About Java Archive Files (JAR)"와 "About the Manifest file"을 보기 바란다.

명령어 라인에서 MIDlet 아카이브하기

만일 Archive Builder를 포함하고 있지 않은 JBUILDER 퍼스널에서 실행하고 있습니다면, JBUILDER 밖의 명령 라인에서 어플리케이션을 아카이브 해야 합니다. 하지만, 만일 Nokia 개발 도구를 사용하고 있습니다면, 필요한 파일들을 생성하기 위해 JBuilder Personal 안의 아카이브 툴을 사용할 수 있습니다. 페이지 8-16에 있는 "Archiving MIDlets with the Nokia Developer's Suitefor J2ME™"을 보라.

명령어 라인에서 아카이브 하는 과정은 다음과 같은 기본 단계로 구성됩니다.

1 JBUILDER 내에서 사전 검증된 클래스들을 생성하기 위해 프로젝트를 컴파일하고 실행합니다. 다음과 같은 과정을 수행합니다.

1 Project Properties 대화 상자의 Run|MIDlet 페이지의 Running 옵션 이전에 Compile 항목을 체크합니다. (기본적으로 체크되어 있습니다.)

2 모든 파일을 저장합니다.

3 프로그램을 실행시킨다. 이러한 단계를 거치면 사전 검증된 클래스들을 형성하고, 프로젝트 내에서 그것들을 위치시킨다.

2 jar 수행 중 기본적인 manifest 파일의 내용에 추가할 MIDlet 속성을 포함하는 manifest 파일을 생성합니다.

3 JAR 내에서 하나의 파일에 manifest 파일의 내용을 추가시키기 위해서 m 옵션을 사용한 jar 명령으로 JAR 파일을 생성합니다.

4 JAR 파일내에 이미 포함되지 못한 어떠한 리소스들을 추가하여 JAR 파일을 업데이트 하기 위해 u 옵션을 사용한 jar 명령을 다시 실행합니다.

5 어플리케이션 메니저가 MIDlet을 실행시킬 시에 사용하는 MIDlet 속성을 포함하는 JAD 텍스트 파일을 생성합니다.

Manifest 파일 생성하기

jar 명령은 JAR 파일내의 META-INF/MANIFEST.MF 로 저장되는 텍스트 파일인 기본 manifest를 생성합니다. 하지만 이 manifest 파일은 아카이브시에 MIDlet과 관련된 모든 것을 포함하지는 않는다. MIDlets 아카이브할 때, MIDlet 속성을 포함하기 위해 기본 manifest 파일을 변경해야 합니다.

Example of a Manifest file which includes MIDlet attributes:

MIDlet 속성을 포함하는 Manifest 파일의 예제이다.

Manifest-Version: 1.0

MIDlet-Name: UIDemoMIDlet

MIDlet-1: UIDemoMidlet, /icons/ivanIcon.png,

com.borland.JBuilder.samples.micro.uidemo.UIDemoMidlet

MIDlet-Icon: /icons/ivanIcon.png

MIDlet-Version: 1.0

MIDlet-Vendor: Borland Software Corporation

MicroEdition-Configuration: CLDC-1.0

MicroEdition-Profile: MIDP-1.0

jar 명령의 m 옵션은 JAR 파일 생성할 때, 텍스트 파일로부터 기본 manifest에 정보를 추가하도록 해 줍니다. 그런 까닭에, JAR 파일 생성 전에 다음 형식을 사용하여 jar 명령에서 지정할 수 있는 추가 정보와 함께 외부의 manifest 파일을 생성할 필요가 있습니다.

jar cmf manifest-addition jar-file input-file(s)

MIDlet 속성 요소를 나열하는 외부의 텍스트 파일을 생성하기 위해서는 표준의 텍스트 에디터를 사용합니다. 이 속성들에 관한 설명들이 아래에 기술되어 있습니다.

MIDlet manifest 속성들

MIDlet JAR 파일 내의 manifest 파일은 MIDlet의 속성을 정의하는 *name-value* 줄을 포함하여야 합니다. 이 속성들의 몇 몇은 필수이며, 다른 것들은 선택적이다.

Manifest 파일을 위한 MIDlet 속성에는 아래와 같은 것들이 있습니다.

Required attributes	
MIDlet-<n>	The name, icon, and primary class of each MIDlet in the MIDlet Suite. Enter one line for each MIDlet, with the values separated by commas.
	Name — Used to identify the MIDlet to the user on the display. The name MIDlet-1 refers to the first (and sometimes only) MIDlet in the JAR file. Successive MIDlets are assigned numbers incrementally, such as MIDlet-2, MIDlet-3, and so on.
	Icon — Name of the icon image (PNG) within the JAR file to display beside the name of the MIDlet.
	Class — The class name of the MIDlet. The class must have a public constructor with no arguments.
Optional attributes	
MIDlet-Icon	The name of the icon image (PNG) within the JAR file used to represent the MIDlet Suite.
MIDlet-Description	A description of the MIDlet.
MIDlet-Info-URL	A URL that describes the MIDlet Suite further.
MIDlet-Data-Size	The minimum number of bytes of persistent data storage required on the device for the MIDlet to run. The default if not specified is zero.

Required attributes	
MIDlet-Name	The name of the MIDlet Suite that identifies the MIDlets to the user.
MIDlet-Version	The version number of the MIDlet Suite in the format <major>.<minor>.<micro>. The default micro revision number is zero.
MIDlet-Vendor	The organization providing the MIDlet Suite.
MicroEdition-Profile	The name and version of the J2ME profile required to run the MIDlet Suite. The format must be in the format MIDP-1.0.
MicroEditon-Configuration	The name and version of the J2ME configuration required to run the MIDlet Suite. The format must be in the format CLDC-1.0.

참고

• “Understanding the Manifest”, at http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html

• “Modifying a Manifest File”, at http://java.sun.com/docs/books/tutorial/jar/basics/mod.html

JBuilder 외부에서 JAR 파일 생성

JAR 파일을 생성하기 위해서는

1 CLASSPATH 환경 변수 내에 jar 가 포함되도록 합니다. 이를 확인하기 위해서는, jar -version 을 타이핑하도록 합니다. 만일 원하는 결과가 아니라면, classpath 에 포함되도록 필요한 작업들을 수행하라.

2 MIDlet에 의해 요구되는 모든 파일을 포함하게 될 MIDlet을 위해 특별한 아카이브 디렉토리를 생성하라. 이는 선택적이지만, 아카이브 과정을 보다 다루기 쉽게 해 줍니다. 실행을 위해 MIDlet에 요구되는 올바른 디렉토리 구조를 형성합니다.

3 사전 검증된 클래스 파일들을 프로젝트의 <output path> 디렉토리부터 아카이브 디렉토리가 되는 classes 하위 디렉토리로 이동시킨다.

팁 만일 MIDlet이 필요로 하는(이미지와 같은) 추가적인 자원들을 이 디렉토리로 옮기고자 합니다면, JAR 프로세스내에서 추가적인 단계를 줄일 수 있으며 이는 첫번째 아카이브 패스에 포함됩니다. 그렇지 않으면 JAR 프로세스를 실행시키고, 이것들을 포함시키기 위해 u 옵션을 사용하여 JAR 파일을 갱신하기 위한 두번째의 JAR 프로세스를 실행시켜야만 합니다. http://java.sun.com/docs/books/tutorial/jar/basics/update.html 에서 “Updating a JAR”을 참고하라.

4 MIDlet에 필요한 속성들을 포함하는 manifest 파일을 생성합니다. 8-2 페이지의 “Creating the manifest file”을 참고하라.

5 다음의 형식을 사용하여 프로젝트의 상위 디렉토리의 명령 라인에서 jar를 실행시킨다.

jar cmf manifest-addition jar-file input-file(s)

팁 해당하는 디렉토리와 하위 디렉토리의 모든 파일을 포함하기 위해 input-file(s) 위치에 * 문자를 사용할 수도 있습니다.

예를 들어, MIDlet 튜토리얼이라 불리는 어플리케이션을 아카이브 합니다고 가정하자.

1 현재 디렉토리는 MIDletTutirial의 상위 디렉토리이다.

2 midletattributes 이라 불리는 manifest 텍스트 파일을 생성합니다.

3 생성하고자 하는 JAR 파일은 MIDletTutorial.jar 가 될 것입니다.

이와 같은 명령어로 JAR 파일을 생성합니다.

jar cmf midletattributes MIDletTutorial.jar *

참고

• “Using JAR Files: The Basics”, from *The Java Tutorial* at http://java.sun.com/docs/books/tutorial/jar/basics/index.html.

JBuilder 외부에서 JAD 파일 생성

JAD(Java Application Descriptor) 파일은 JAR 파일을 외부의 .jad 확장자를 지니는 표준의 텍스트 파일이다. manifest와 같이 MIDlet Suite 안의 MIDlet을 설명하는 속성들의 나열로 구성되어 있습니다. JAD 파일은 JAR 파일안의 MIDlet을 관리하는 어플리케이션 관리 소프트웨어로서 사용됩니다.

모든 JAD 파일에 사용되도록 미리 정의된 MIDlet 속성 set이 존재합니다. 이들 속성 중 몇가지는 manifest와 같이 공유됩니다. MIDlet-Name, MIDlet-Vendor, MIDlet-Version 속성들은 manifest 와 JAD 파일 모두에서 동일한 값이어야 합니다. 공유되는 다른 것들 중에서 JAD파일의 것은 manifest 파일의 것을 오버라이드합니다.

MIDlet 속성으로서 JAD 파일의 예는 아래와 같습니다.

MIDlet-1: UIDemoMidlet, /icons/ivanIcon.png,
com.borland.JBuilder.samples.micro.uidemo.UIDemoMidlet
MIDlet-Icon: /icons/ivanIcon.png
MIDlet-Jar-Size: 32127
MIDlet-Jar-URL: Midp_UIDemo.jar
MIDlet-Name: UIDemoMIDlet
MIDlet-Vendor: Borland Software Corporation
MIDlet-Version: 1.0

이 JAD 파일과 함께 하는 manifest 파일을 보기 위해서는 페이지 8-3의 “Example of a Manifest file which includes MIDlet attributes:” 을 참고하도록 합니다. Manifest 파일과 함께, JAD 파일은 필수, 선택 속성 모두를 가지고 있습니다. 이는 또한 MIDlet으로 전달되는 파라미터들을 포함합니다.

JAD 파일 속성들

JAD 파일을 위한 MIDlet 속성들은 아래와 같습니다.

Required attributes	
MIDlet-Name	The name of the MIDlet Suite that identifies the MIDlets to the user.
MIDlet-Version	The version number of the MIDlet Suite in the format major.minor.micro. The default micro revision number is zero.
MIDlet-Vendor	The organization providing the MIDlet Suite.
MIDlet-Jar-URL	The URL from which the JAR can be downloaded.
MIDlet-Jar-Size	The size of the JAR file in bytes. (This information is used by the application manager to determine if JAR file can be downloaded to the device.)
MIDlet-<n>	The name, icon, and primary class of each MIDlet in the MIDlet Suite. Enter one line for each MIDlet, with the values separated by commas. Name — Used to identify the MIDlet to the user on the display. The name MIDlet-1 refers to the first (and sometimes only) MIDlet in the JAR file. Successive MIDlets are assigned numbers incrementally, such as MIDlet-2, MIDlet-3, and so on. Icon — Name of the icon image (PNG) within the JAR file to display beside the name of the MIDlet. Class — The class name of the MIDlet. The class must have a public constructor with no arguments.

Optional attributes	
MIDlet-Icon	The name of the icon image (PNG) within the JAR file used to represent the MIDlet Suite.
MIDlet-Description	A description of the MIDlet.
MIDlet-Info-URL	A URL that describes the MIDlet Suite further.
MIDlet-Data-Size	The minimum number of bytes of persistent data storage required on the device for the MIDlet to run. If not specified, the default is zero.
[MIDlet specific attributes]	Attributes not starting with "MIDlet-" that are related to specific MIDlets.

Archive Builder로 MIDlets 아카이브

The Archive Builder is a feature of JBuilder Professional and Enterprise.

Archive Builder는 JBuilder Professional 과 Enterprise에서 사용하실 수 있습니다.

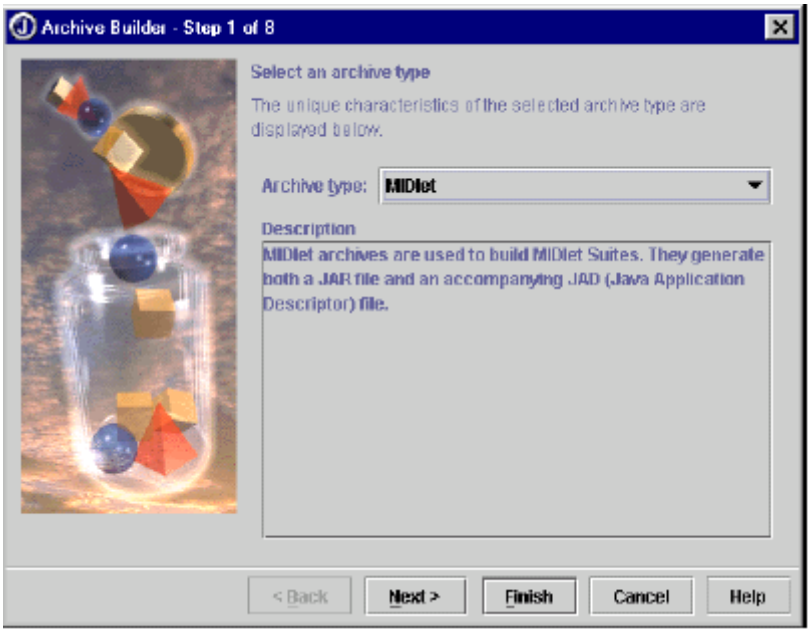
JBuilder Archive Builder는 JAR, manifest 그리고 JAD 파일 생성을 포함한 모든 아카이브 과정을 지원합니다. Archive Builder는 배포를 위한 MIDlets을 패키지화하는 과정을 눈에 띌 정도로 빠르게 해 줍니다. 이는 적당한 manifest 파일을 이용하여 JAR 파일 내에 파일들을 아카이브하기 전에 프로젝트에서 필요한 모든 파일을 찾아내어 JAR 파일의 내용을 커스터마이즈 할 수 있도록 해 줍니다. 또한 이에 수반하여 생성되는, 마법사에서 입력된 정보를 기반으로 한 JAD(Java Application Descriptor)파일을 생성해 줍니다.

아래는 JBuilder Archive Builder를 이용하여 아카이브 과정을 수행하는 과정이다. MIDlet 튜토리얼에서 생성된 MIDlet은 이 예제에서는 사용되지만, 이러한 단계로 어떤 생성할 MIDlet에도 적용할 수 있습니다.

중요 처음으로 Archive Builder를 실행하기 전에, 프로젝트를 위한 클래스 파일을 생성하기 위해 프로젝트를 컴파일 해야 합니다. Archive Builder는 항상 최근에 생성된 클래스 파일들을 사용하지만, 어떠한 것이 최근의 것인지는 체크하지 않는다. 만일 컴파일 이후에 소스를 변경했다면, Archive Builder를 실행하기 전에 다시 프로젝트를 컴파일 하도록 해야 합니다.

1 Archive Builder를 열기 위해 JBuilder 메뉴에서 Wizards|Archive Builder 를 선택합니다.

Figure 8.1 Archive Builder, Step 1

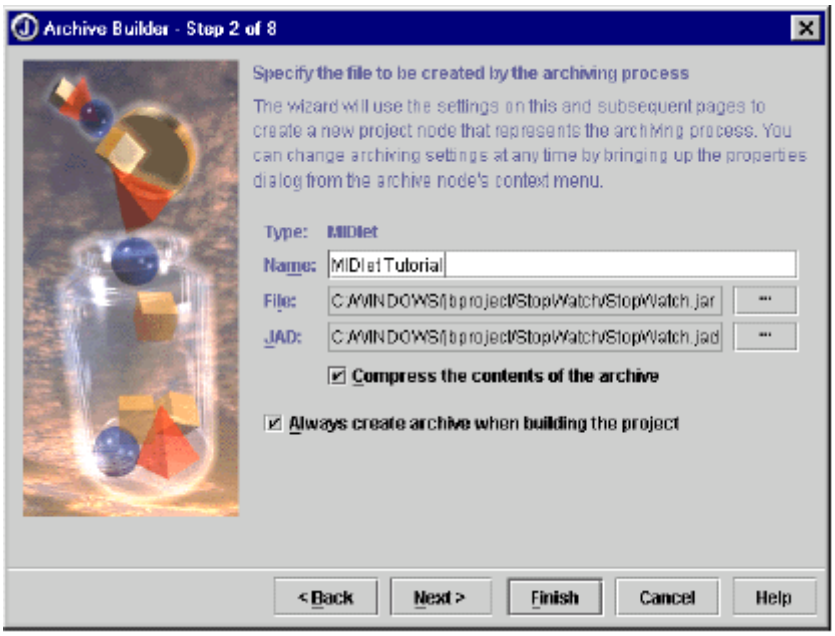


2 1단계에서는 Archive Type으로 MIDlet을 선택합니다. 이것은 manifest와 JAD을 동반하는 JAR 파일로 구성된 MIDlet Suite 을 생성할 것입니다.

주의 JAD 파일이나 JAR manifest 파일의 기본적인 정보를 사용하여 자동적으로 설정되도록 하려면, 마법사의 어떤 단계에서도 Finish 버튼을 누를 수 있습니다.

3 2단계로 가기 위해 Next를 누른다.

Figure 8.2 Archive Builder, Step 2



4 2단계에서는 MIDlet Suite을 위한 이름을 쓴다.

5 JAR과 JAD의 이름, 위치를 변경하고자 하지 않는다면 그냥 내버려둔다. 변경하기 위해서는 JAR 파일을 위한 브라우저를 열고 선택하면 됩니다. JBuilder는 자동적으로 패키지 이름과 같은 JAR 파일 이름을 부여하며,, JAR 파일에 JAD 파일의 이름과 위치를 매치시킨다.
경고 전혀 문제 없이 JAD 파일의 이름을 바꿀 수 있습니다. 하지만 JAD 파일로의 패스를 변경하게 됩니다면, 그것은 JAD 내의 상대적인 패스로부터 절대적인 패스로의 MIDlet-Jar-URL 속성을 변경시키게 될 것입니다. 이것은 실제 기기에 어플리케이션을 배포할 때 문제를 발생시키는데, 왜냐하면 기기의 파일 구조와 컴퓨터 상의 파일 구조가 서로 맞지 않기 때문이다.

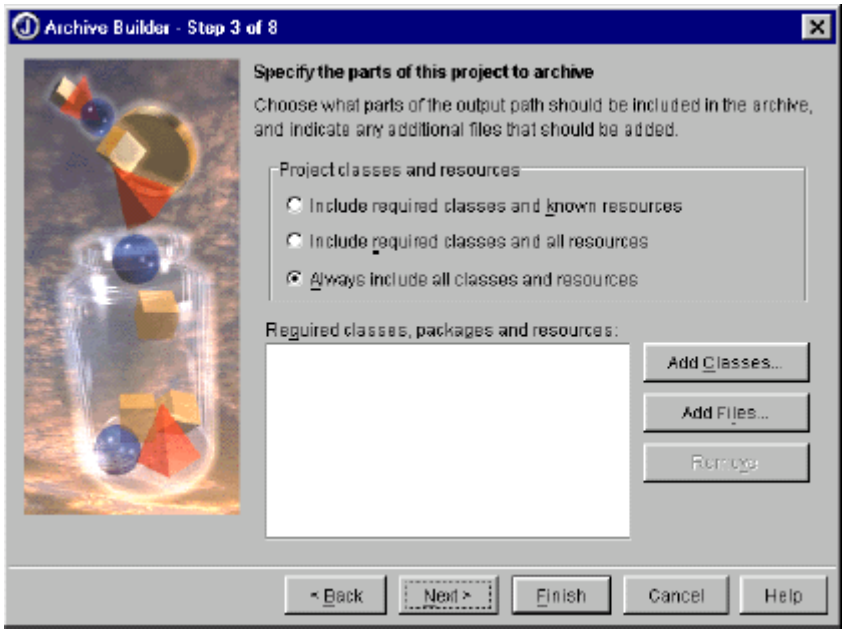
6 Compress The Contents Of The Archive 를 체크된 상태로 합니다. 제한된 기기로 다운로드를 해야 하므로, JAR 파일을 압축해야 합니다. 또한 프로젝트를 생성할 때 Always Create Archive를 체크된 상태로해서, 프로젝트를 만들거나 완성할 때마다 JAR 파일이 재생성되도록 합니다.

팁 만일 마법사의 각 단계에 포함된 것들에 관해서 자세한 내용을 알고 싶다면, 각 단계의

하단에 위치한 Help 버튼을 클릭합니다.

7 3단계로 가기 위해 Next 버튼을 누른다.

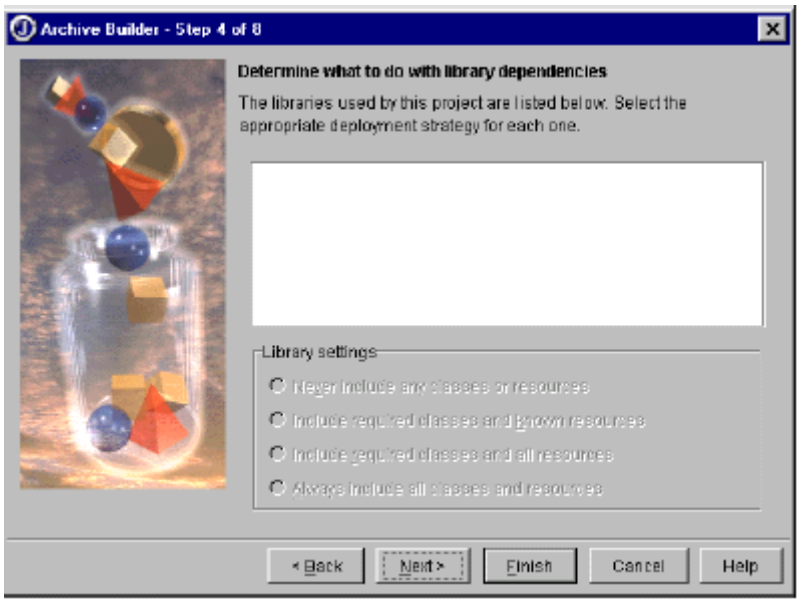
Figure 8.3 Archive Builder, Step 3



8 아카이브 할 때 프로젝트 outpath상의 어떤 파일들이 포함되어야 할 지를 지정합니다. 필요상 추가되는 파일들을 지정하기 위해 Add Classes 나 Add Files 을 클릭합니다.

9 4단계로 가기 위해 Next 버튼을 클릭합니다.

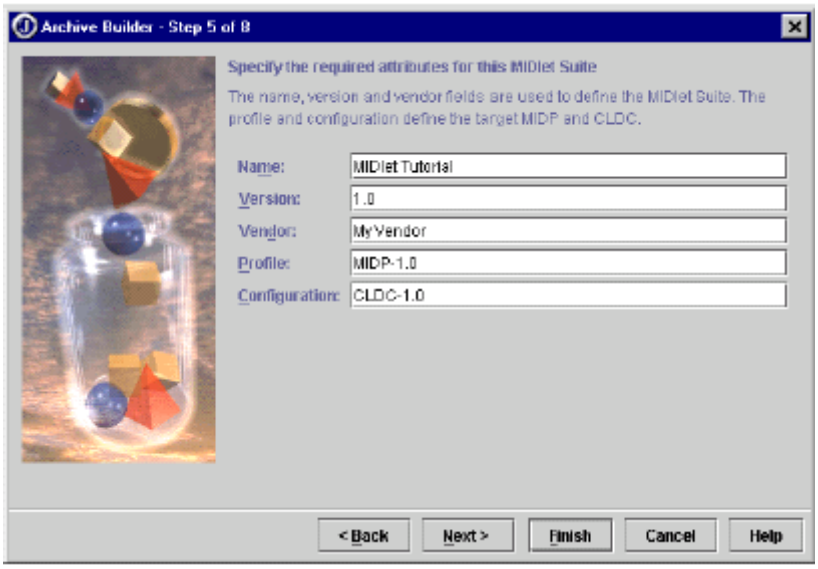
Figure 8.4 Archive Builder, Step 4



10 만일 열거되어 있는 어떤 라이브러리가 존재합니다면, 4단계에서는 라이브러리에 종속적으로 해야 할 일을 지정합니다. 각각에 대해 개개의 배포 전략을 고를 수 있습니다.

11 5단계로 가기 위해 Next를 누른다.

Figure 8.5 Archive Builder, Step 5



12 5단계에서는 MIDlet Suite를 위한 필수 속성을 지정합니다.

중요 모든 필드는 이 단계에서 값을 갖는다. 그리고 Version, Profile 그리고 Configuration 은 다음의 형식을 사용해야만 합니다. :

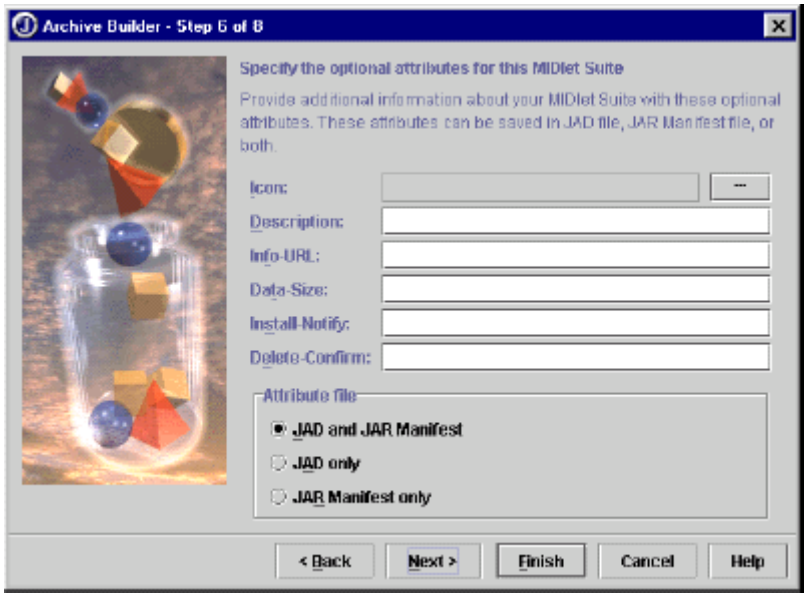
Version: <major>.<minor>.<micro> (For example 1.0)

Profile: <profile-major>.<minor>.<micro> (For example MIDP-1.0)

Configuration: <configuration-major>.<minor>.<micro> (For example CLDC-1.0)

13 6단계로 이동하기 위해 Next를 누른다.

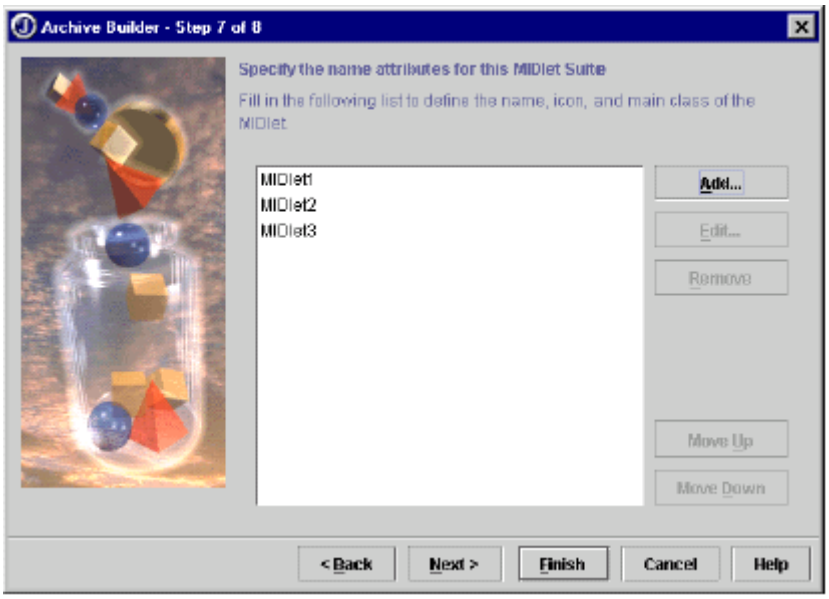
Figure 8.6 Archive Builder, Step 6



14 6단계에서는 MIDlet Suite를 위한 선택적인 속성들을 입력합니다. 이러한 속성들에 관해 더 자세히 알고 싶다면 Help 버튼을 클릭하면 됩니다. Install-Notify 와 Delete-Confirm 는 단지 인터넷 서버로부터 기기로 설치되는 아카이브를 생성하고자 할 때만 적용되는 OTA (Over The Air) Provisioning 옵션들이다. 또한 MIDlet 속성들을 JAD 파일과 JAR manifest 파일 모두 혹은 그들 중 하나에만 쓸 것인지를 선택할 수 있습니다.

15 7단계로 이동하기 위해 Next 버튼을 클릭합니다.

Figure 8.7 Archive Builder, Step 7



16 7단계에서는 MIDlet Suite내에 어떠한 MIDlet들이 포함될 것인가를 지정하고, MIDlet-n 속성을 입력합니다.

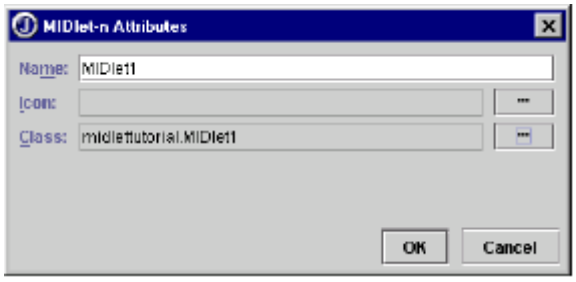
Archive Builder는 프로젝트로부터 사용가능한 모든 MIDlet들을 기본적으로 포함하도록 합니다. 이는 또한 각 MIDlet에 자동적으로 Name과 class MIDlet-n 속성값을 채워넣는다. 만일 MIDlet을 나타낼 아이콘을 원합니다면, 명시적으로 그 속성을 지정할 필요가 있습니다. 리스트로부터 MIDlet을 제거하거나 새로운 것을 추가하거나, 그 순서를 바꿀 수도 있습니다. MIDlet-n 속성 정보는 manifest와 JAD파일에 필요하며 기기상의 Suite내의 모든 MIDlet들이 실행 가능 하도록 해 줍니다. 이는 또한 Suite내의 MIDlet들이 서로 의사소통을 하거나 JAR 파일내의 자원들을 공유할 수 있게 합니다.

MIDlet을 위한 속성 변경

1 리스트에서 MIDlet을 선택합니다.

2 MIDlet-n 속성 대화 상자를 열기 위해 Edit를 클릭합니다.

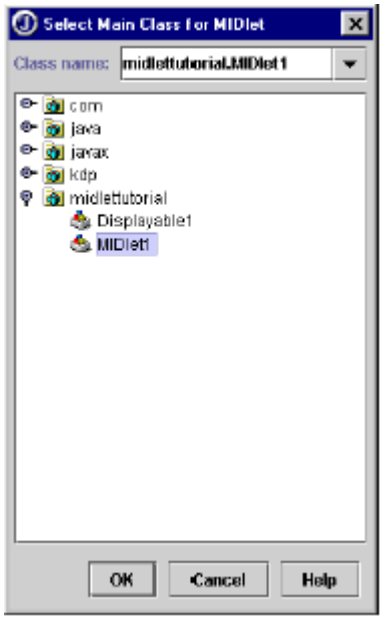
Figure 8.8 Archive Builder, MIDlet-n Attributes dialog box



3 원합니다면 MIDlet의 이름을 바꾸자. MIDlet 이름은 사용자가 MIDlet을 식별하는 데 사용됩니다.

4 Icon 필드의 오른쪽에 있는 버튼을 클릭합니다. 이 MIDlet을 위한 아이콘으로서 PNG(.png) 이미지를 선택합니다. 이는 선택적이다. 이 아이콘은 기기에서 MIDlet 이름 옆에 나타난다.

5 Class 필드 오른쪽에 위치한 버튼을 클릭합니다. MIDlet의 메인 클래스로 이동합니다. (예를 들어 midlettutorial.MIDlet1). Archive Builder는 필드에 path/class 이름을 입력합니다. 이 정보는 필수적이다.



6 MIDlet-n 속성 대화 상자로 돌아가기 위해 OK를 클릭합니다.

7 MIDlet Suite 내에 어떤 추가적인 MIDlet이 필요하다면 각 MIDlet에 적당한 Name과 Class를 바꾸어 가면서, 이 과정을 반복합니다.

리스트로 MIDlet을 추가하기

Add를 클릭하고, MIDlet-n 속성 대화 상자 내의 Name, Icon 그리고 Class 속성들을 입력하고 OK를 클릭합니다.

주의 MIDlet은 반드시 MIDlet Suite에 추가시키기 위해 프로젝트 내에 있어야 합니다.

리스트에서 MIDlet 제거하기

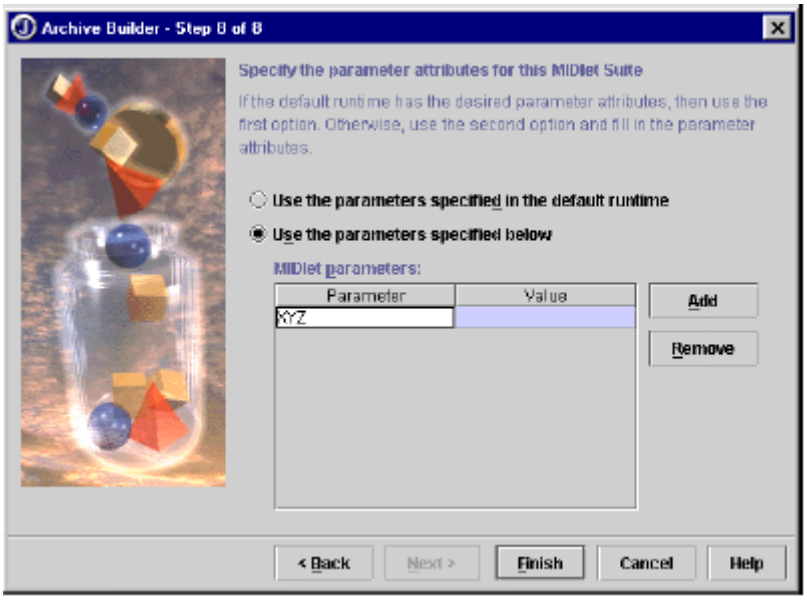
MIDlet Suite에서 제외하고 싶은 MIDlet을 선택하고 Remove를 클릭합니다.

MIDlet의 순서 변경하기

7단계에서 나열된 MIDlet들의 순서는 JAD와 manifest파일 내의 순서를 결정하고, 또한 기기에 나타나는 순서를 정해줍니다. 리스트내에서 MIDlet을 다른 위치로 이동시키고 싶다면, Move Up 또는 Move down을 클릭합니다.

17 끝나면 8단계로 이동하기 위해 Next를 클릭합니다.

Figure 8.10 Archive Builder, Step 8



18 기본적인 실행 파라미터 속성을 사용할 지 또는 새로운 상세를 사용할 것인지를 가리킨다. 이러한 파라미터들은 실행시에 MIDlet에 전달됩니다.

• 기본 실행시에 지정된 파라미터들을 사용합니다. 기본 실행 파라미터는 Run|Configurations 와 Project|Project Properties 메뉴하에서 볼 수 있는 Run|MIDlet 페이지에 설정되어 있습니다.

• 다음과 같이 지정된 파라미터를 사용합니다. 만일 기본적인 실행 환경이 원하는 파라미터를 가지고 있지 않다면 Add를 클릭하고 첫째, 둘째 등등의 칼럼에 파라미터 이름을 입력합니다. 다른 파라미터를 위한 추가 작업을 합니다.

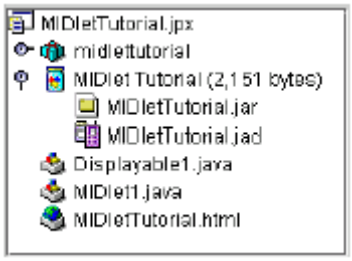
19 다 되었으면 Finish를 클릭합니다.

20 다시 한 번 완성하고 저장합니다.

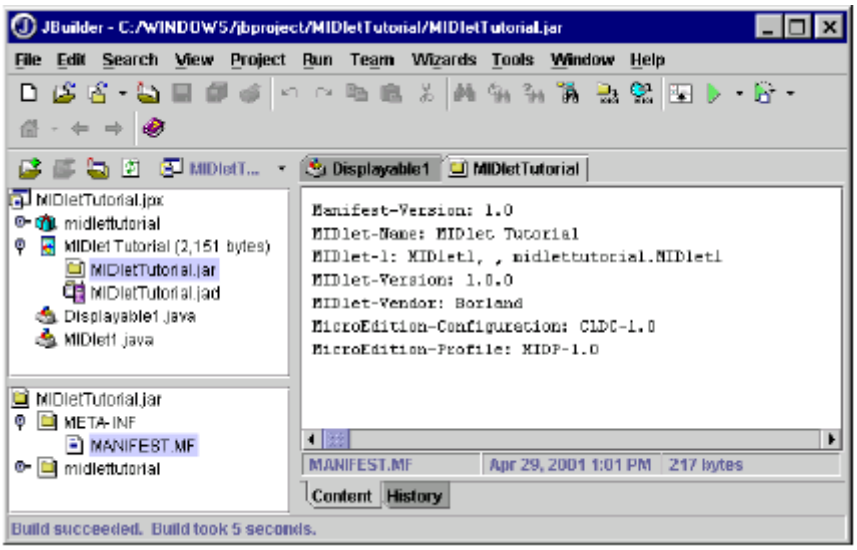
아카이브 파일의 새로운 노드가 MIDlet Tutorial 라 불리는 프로젝트 판에 추가되지만, JAR나 JAD 파일을 위한 노드는 전혀 볼 수 없습니다.

프로젝트를 다시 컴파일하기 위해 Rebuild  를 클릭하고 프로젝트 판에 그러한 노드를 생성하고 나서 모두 저장을 클릭합니다. 

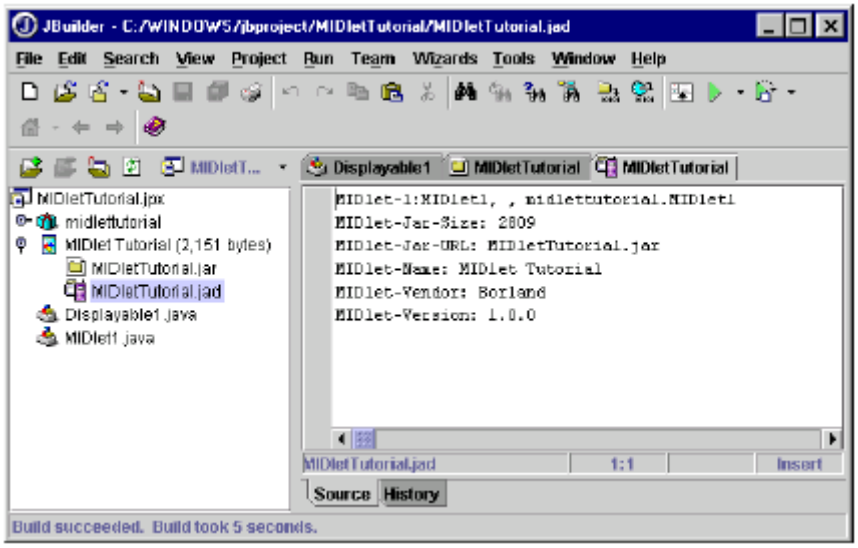
이제 MIDlet Tutorial **을 위한 아카이브 노드를 갖게 되었다.** MIDletTutorial.jar **과** MIDletTutorial.jad **을 위한 노드를 보기 위해** MIDlet Tutorial **을 더블클릭합니다.**



MIDletTutorial.jar 을 더블클릭하고 content 화면에 나타나는 manifest 파일의 내용을 체크합니다. 마법사에서 입력한 MIDlet에 관한 모든 정보를 확인합니다. 또한 structure 화면에 JAR 파일의 내용이 어떻게 기술되어 있는 지도 확인합니다.



MIDletTutorial.jad 를 더블클릭합니다. JAD 파일의 내용을 연구합니다. 다시 마법사에서 입력한 정보들을 확인하고 그와 manifest 파일 간의 유사성과 차이점을 확인해 본다. JAD 파일은 JAR 파일 내에 몇 몇 정보를 복제하고 또한 JAR 파일의 크기와 같은 예처럼, 추가적인 정보를 더해야 합니다.



J2ME™ 노키아 개발자 Suite으로 MIDlet으로 MIDlet 아카이브 하기

J2ME™에서 Nokia 개발자 Suites는 MIDlet Suite를 생성하기 위한 도구들을 포함합니다. 이 툴 셋을 열기 위해 J2ME™의 Tools|The Nokia Developer's Suite을 선택합니다. 이 도구에 관한 자세한 내용은 Nokia Developer's Suite 문서를 참조합니다.

JBuilder™ Mobileset 개발자 가이드

문서번호 : IMOBILE-JBM001

Chapter 9

Over The Air (OTA) Provisioning

Appendix 참조

문서번호 : IMOBILE-JBM001

Chapter 10

JBUILDER 모바일셋 샘플

MobileSet은 <JBuilder>\samples\Mobile\ 에 몇 가지 MIDP 샘플을 제공하고 있습니다. JBUILDER 안에 있는 이러한 샘플을 열어 실행시키기 위해서는 아래의 지시를 따르도록 합니다.

중요 특정한 JDK를 사용하는 샘플을 실행시키는 경우, 반드시 JBUILDER에 JDK를 먼저 설정한 후에 프로젝트에서 사용하는 기본 JDK로 지정해야 합니다.

JBUILDER에서 MIDP 샘플 실행

MIDlet 파일로부터 실행하기

주 MIDlet 파일로부터 샘플을 실행하기 위해서는 다음의 단계를 따릅니다.

1 File|Open Project. 선택

2 <JBuilder>\samples\Mobile 디렉토리로 이동

3 실행시키고자 하는 디렉토리로 이동한 다음 샘플을 위한 프로젝트 파일을 엽니다.

4 Project|Project Properties를 선택하고, Paths 페이지에서 JDK 필드 옆의 버튼을 클릭하고, 이 MIDlet을 실행하기 위해 사용하고자 하는 MIDP/CLDC JDK 을 선택합니다. 그리고 Project Properties 대화상자를 닫습니다.

5 툴바의 Run  버튼을 클릭하거나, 프로젝트 판 내의 주 MIDlet .java 파일에서 오른쪽 클릭하고 Micro Run  을 선택합니다.

JAD 파일로부터 실행하기

JAD 파일로부터 샘플을 실행시키려면 다음 지시를 따릅니다.

1 File|Open 프로젝트를 선택합니다.

2 <JBuilder>\samples\Mobile 디렉토리로 이동합니다.

3 실행하고자 하는 샘플 프로젝트 파일(.jpx)을 엽니다.

4 Project|Project Properties를 선택하고, Paths 페이지에서 JDK 필드 옆의 버튼을 클릭하고, 이 MIDlet을 실행하기 위해 사용하고자 하는 MIDP/CLDC JDK 를 선택합니다. 그리고 Project Properties 대화상자를 닫습니다.

5 Run 페이지로 전환하여 New 버튼 눌러 새로운 실행 환경을 만듭니다. 기존의 설정 사항을 선택하고 Edit 버튼을 클릭합니다. Runtime properties 대화상자가 나타납니다.

6 Runtime properties 대화상자의 Run 페이지에서 MIDlet 탭을 클릭합니다.

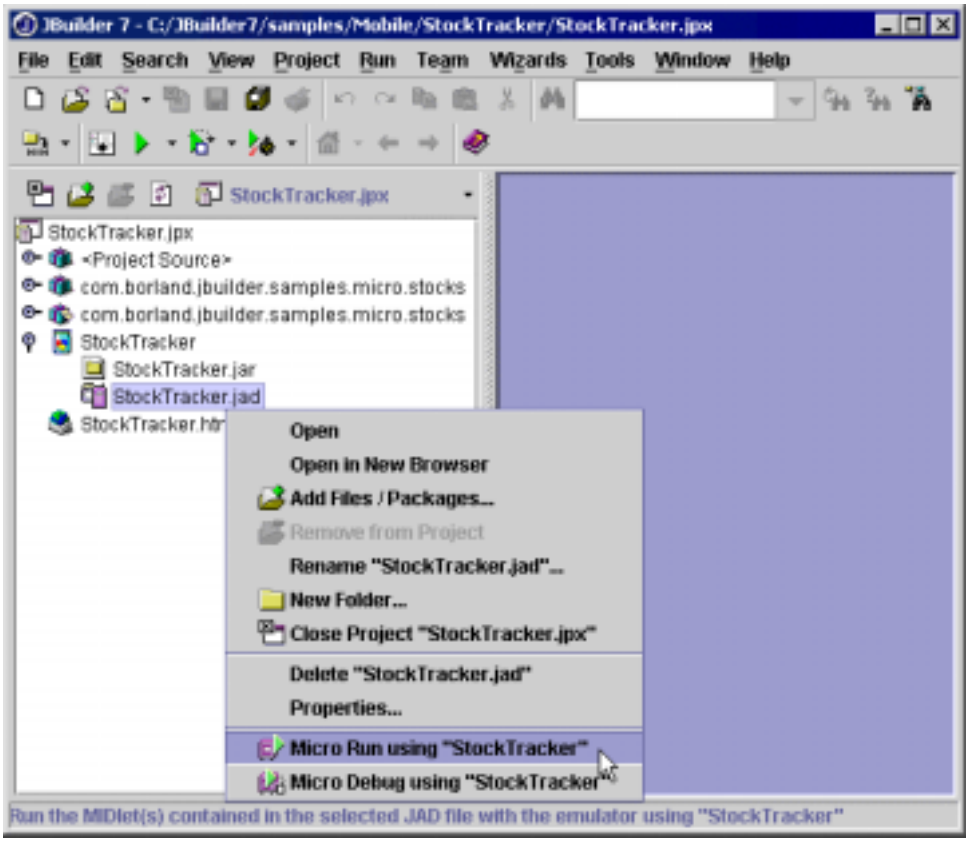
7 JAD 파일 옵션을 선택하고 그 옆의 … 버튼을 눌러 파일 브라우저 창을 엽니다.

8 프로젝트의 JAD 파일로 이동하여 선택한 후 OK 버튼 클릭합니다.

9 OK 눌러 Project properties 대화 상자를 닫습니다.

10 툴바의 Run  버튼을 클릭합니다.

주의 만일 JAD 파일이 프로젝트에 추가되었다면, 손쉽게 프로젝트 판의 JAD 파일을 오른쪽 클릭하고 Micro Run  을 선택합니다. 프로젝트에 JAD 파일을 추가하기 위해서는 Add Files/Package  버튼을 사용합니다.



에뮬레이터 사용

에뮬레이터는 MIDlet 이름을 보여주며 실행됩니다. 사용하는 디바이스에 따라 파일을

탐색하고 선택하는 방법이 달라집니다. 이에 대한 정보는 디바이스 문서를 참고 하시기 바랍니다.

모바일셋과 함께 제공되는 샘플

에뮬레이터가 열리고 MIDlet 이름이 나타날 것입니다. 에뮬레이터에서 파일을 선택하거나 이동하는 방법은 사용하는 기기에 따라 다릅니다. 이 정보를 위해서는 기기 문서를 참조하시기 바랍니다.

Hello World

The Hello World 샘플은 아주 간단한 Hello World 어플리케이션으로서 하나의 MIDlet을 작성하는데 필요한 내용들을 담고 있습니다.

HelloWorldMidlet이 Main MIDlet 클래스입니다. 이것이 MIDlet을 초기화 하고 HelloWorldForm 인스턴스를 만들어 HelloWorld 텍스트를 디스플레이 합니다.

이 샘플이 담긴 디렉토리는 <JBuilder home>\samples\Mobile\HelloWorld이며 프로젝트 파일은 <JBuilder home>\samples\Mobile\HelloWorld\ 아래에 위치하는 HelloWorld.jpx 파일입니다.

MIDP UI Demo

MIDP UI 데모 샘플은 MIDP API 스펙에서 사용 가능한 UI 요소들에 대한 간단한 데모입니다. 이는 몇가지의 서브 데모로 구성되어 있습니다. 각각의 세부 데모들은 특정 MIDP UI 요소들에 대한 디스플레이와 관련된 내용입니다. 그 구성을 보자면

• **Canvas**: MIDP에서 사용 가능한 로우 레벨 드로잉 API 사용법에 대한 내용을 다룹니다.

• **Label**: 스크린에 문자열이나 이미지를 담는 세가지 방법에 대한 내용을 담고 있습니다.

• **DateField**: DateField 객체의 세가자 모드를 보여줍니다. : Date Only, Time Only, Date/Time.

• **Alert**: 정의된 모든 alert 타입을 보여줍니다. 각각의 Alert는 두가지 방법으로 나타납니다. 5초 후에 사라지는 Alert와 사용자의 조작을 통해서만 사라지는 모달(modal) Alert가 있습니다.

• **ChoiceGroup**: ChoiceGroup 타입의 특성을 다룹니다. 특성에는 explicit (오직 선택이 가능할 경우)와 multiple (복수 선택이 가능할 경우)이 있습니다. (폼에 담겨진)하나의 스크린에 다수의 ChoiceGroups도 가능합니다. 주의할 것은 MIDP API 스펙에서는 ChoiceGroup의 타입을 생성자 인수를 통해서만 설정하도록 하고 있습니다는 것입니다. 즉 ChoiceGroup 타입에 대한 setter 프로퍼티는 존재하지 않습니다.

• **Gauge**: MIDP에서 사용 가능한 두가지 gauges 타입을 가지고 있습니다. interactive와 non-interactive가 이에 해당하며 MIDP API 스펙에서는 Gauge의 타입을 생성자 인수를

통해서만 설정하도록 하고 있습니다. 즉 Gauge타입에 대한 setter 프로퍼티는 존재하지 않습니다.

• **ImageItem**: 이미지를 PNG (.png) 포맷으로 보여줄 수 있습니다.

• **List**: List는 세가지 타입을 가지고 있습니다. explicit (오직 선택이 가능할 경우), implicit (라디오 버튼과 같은 비주얼한 요소가 없으며 하나만 선택할 경우), multiple (복수 선택이 가능할 경우). ChoiceGroup과는 달리 한번에 하나만의 리스트만 디스플레이 가능하다. 다른 요소들과 마찬가지로 MIDP API 스펙에서는 생성자 아규먼트를 통해서만 List 타입 설정이 가능하게끔 되어 있습니다. 그러므로 List 타입에 대한 setter 프로퍼티는 존재하지 않습니다.

• **TextBox**: 사용자가 수정 가능한 텍스트 박스를 보여줍니다. 오직 한번에 하나의 텍스트 박스만이 보여질 수 있습니다.

• **TextField**: 길지 않은 텍스트 문자열을 입력/편집할 수 있는 사용자 정의 텍스트 필드를 보여줍니다. 폼에 담기는 하나의 스크린에 복수의 TextField가 들어갈 수 있습니다.

• **Ticker**: 텍스트가 가로 방향으로 스크롤되게 합니다. Screen을 상속 받은 객체들에 세팅 될 수 있으며 이러한 객체들로는 List, Form, Alert, TextBox가 있습니다.

UIDemoMidlet 이 Main MIDlet 클래스이며 MIDlet을 초기화 하고 UIDemoList를 인스턴스화 해서 서브 데모 항목을 보여줍니다.

이 샘플이 위치한 디렉토리는 <JBuilder home>\samples\Mobile\Midp_UIDemo 이며 프로젝트 파일은 <JBuilder home>\samples\Mobile\Midp_UIDemo\ Midp_UIDemo.jpx 입니다.

Stock Tracker

The Stock Tracker 샘플은 간단한 stock tracker로서 사용자로 하여금 새로운 주식 항목을 추가하거나 제거하거나 업데이트 간격을 조정할 수 있는 기능을 가지고 있습니다.

이 데모 프로그램에서는 다음과 같은 개념을 다룹니다.

• 원격 서버로 네트웍 연결을 하는 방법.

• MIDlet 세션과 MIDP Record Management System (RMS)을 이용하여 데이터를 저장하는 방법.

StockTracker.java 파일이 MIDlet 메인 클래스이며 원격 서버 주소에 관한 정보와 RMS를 통해 정보를 읽고 저장하는 기능을 담고 있습니다.

StockTrackerDisplay.java 파일이 이 MIDlet's의 메인 디스플레이 내용을 담고 있으며 주식 심볼과 가격을 보여주는 ticker와 주식 심볼을 추가하거나 제거하거나 가격 업데이트 주기 설정이 가능한 리스트를 가지고 있습니다.

StockList.java 는 간단한 헤시 테이블로서 세션동안의 주식 심볼을 저장합니다.
AddStock.java, DeleteStocks.java, SetOptions.java 파일은 Displayable 클래스로서 추가, 삭제, 업데이트와 관련된 핸들링 부분을 담고 있습니다.
이 샘플이 있는 디렉토리는 <JBuilder home>\samples\Mobile\StockTracker이며 프로젝트 파일은 <JBuilder home>\samples\Mobile\StockTracker\StockTracker.jpx입니다.

Stop Watch

Stop Watch 샘플에서는 다음과 같은 내용을 다룹니다.

- Canvas에 대한 기초적인 이해.
- Canvas에 그림을 그리는 방법과 색 세팅하는 법, 사각형에 색을 입히는 법, 폰트 설정, 문자열을 그리는 방법.
- MIDlet에 다수의 Displayable 객체를 추가하고 이동하는 방법.
- Record Management Store (RMS) 레코드 저장소를 이용한 정보 저장과 추출 방법.

이 샘플이 들어있는 디렉토리는 <JBuilder home>\samples\Mobile\StopWatch, 이며 프로젝트 파일은 <JBuilder home>\samples\Mobile\StopWatch\StopWatch.jpx 입니다.

MobileBookStore

Overview

MobileBookStore 샘플은 J2ME 테크롤로지를 클라이언트 단에, J2EE 테크놀로지를 미들/백엔드 단에서 활용하는 예를 보여주는 간단한 어플리케이션입니다. 어떠한 방식으로 J2ME 디바이스에서 MIDP를 이용하여 J2EE의 클라이언트로서 동작하는지 보여주고 있습니다.
J2ME 클라이언트는 웹 티어의 Java 서블릿과 HTTP 연결을 통해 통신하며 Java 서블릿은 JavaBean을 통해서 EJB 티어의 EJB에 담긴 비즈니스 로직에 접근 합니다. 그리고 EJB 티어에서는 JDBC API를 통해 데이터베이스 어플리케이션에 접근하게 되어 있습니다. 이 샘플 어플리케이션에서 사용자는 ISBN을 입력하여 데이터베이스에서 서적을 검생하고 구입하게끔 되어 있습니다.

Description

MobileBookStore 어플리케이션에는 두개의 독립된 JBuilder 프로젝트가 담겨 있습니다. J2ME 클라이언트 프로그램은 BookStoreJ2MEClient.jpx 파일이며 J2EE 서버 프로그램은 BookStore.jpx 파일입니다.

클라이언트 프로그램은 MIDlet으로서 핸드폰(cell phone)의 에뮬레이터를 사용하게끔 되어 있습니다. 사용자는 핸드폰을 통해 서적에 대한 ISBN 번호를 입력합니다. 그러면 MIDlet은 검색 요청을 HttpConnection을 통해 MobileBookStore 서버로 전달하게 되고 검색 결과를 다시 받아서 휴대폰에 보여주게 됩니다. 사용자가 서적을 구입한 후에는 updateInStock 요청이 서버로 전달되고 서버에서는 데이터베이스의 InStock 양을 업데이트 하게 됩니다.
서버 프로그램은 Borland Enterprise Server에 호스트 되게끔 되어 있으며 BES는 웹 컨테이너에서 서블릿을 수행하고 여기서 EJB 컨테이너의 JavaBean과 통신할 수 있게 되어 있습니다.

Running the MobileBookStore sample

MobileBookStore 샘플을 실행하기 위해서는 우선 서버 프로그램을 수행해야 합니다. 그리고 나서 클라이언트 프로그램을 수행합니다. 아래 내용은 그렇게 하기 위한 절차를 설명하고 있습니다. 서버 설정, 서버 수행, 클라이언트 수행 방법입니다.

Configuring and running the server

주의 Borland Enterprise Server 5.0.2 가 설치되어 있어야 하며 JBuilder7과 연동할 수 있도록 세팅되어 있어야 합니다. 만약 준비가 되어 있지 않다면 우선 BES를 설치하셔야 합니다. 다음은 JBuilder7 Enterpirse Edition에서의 작업 순서 입니다.

1 Tools|Configure Server…를 선택합니다. Borland Enterprise Server 5.0.2를 선택하고 서버를 설치한 디렉토리를 지정합니다.

Note 만약 처음으로 JBuilder와 BES 세팅을 하셨다면 JBuilder를 재시작 하셔야 합니다. 만약 샘플 프로젝트 시행 전에 Borland Enterprise Server 5.0.2 설정을 하셨다면 project properties 의 server configuration에서 서버 설정을 refrash합니다.

a Project | Project Properties 클릭합니다.

b Server 탭을 클릭합니다.

c Single server for all services in project 항목을 선택하고 server selected 부분 옆에 있는 … 버튼을 클릭합니다.

d Edit or Select Server 대화 상자에서 Borland Enterprise Server AppServer Edition5.0.2 항목 선택하고 오른쪽 항목의 Enable Server 체크 박스를 선택하고 OK 클릭합니다.

e Project Properties 대화 상자에서 OK 버튼 클릭합니다.

Visibroker Smart Agent의 디폴트 포트 번호는 14000 입니다. 여러분은 이 번호를 수정하여 작업하실 수 있으며 Tools|Enterprise Setup 메뉴 선택하고 CORBA 탭 항목에서 SmartAgent 포트 번호를 변경 하실 수 있습니다.

2 Borland Enterprise Server Management Agent를 구동시킵니다. (Tools| Borland Enterprise Server Management Agent.)

3 다음과 같이 Run Configuration을 수정합니다.

a Run|Configurations 클릭합니다.

b Runtime Properties 대화 상자를 열기 위해 Run 페이지의 Edit 버튼을 클릭합니다. 그리고 Server 탭을 클릭합니다.

c VM parameter 항목의 -Dvbroker.agent.port 값을 위에서 설정한 값으로 바꿉니다. 예를 들어 포트 번호를 15000으로 바꾸기 위해서는 VM parameter 항목을 다음과 같이 수정합니다:

-Dvbroker.agent.port=15000

d database URL 을 수정하여 샘플 데이터베이스 파일을 가리키도록 합니다. MobileBookStore 데이터베이스 (Bookstore.jds).

1 JBuilder7의 프로젝트 창에서 EJB module 노드(bookstore)를 확장합니다.

2 JDBC 1 DataSource 노드를 확장시킵니다.

3 data source definition (DataSource)항목에 더블 클릭합니다.

4 URL 필드를 수정하여 다음과 같은 위치에서 데이터베이스를 지정합니다. Bookstore.jds 는 <JBuilder home>\samples\Mobile\MobileBookStore\BookStore 디렉토리에 있습니다.

4 JBuilder의 툴바에서 Run 버튼 옆의 드랍 다운 화살표 클릭하여 BookStoreServer 선택하여 서버를 실행시킵니다. 이렇게 하면 서버가 실행되고 서블릿 화면이 JBuilder의 컨텐트 창에 나타납니다.

만약 여러분께서 "HTTP Status 500 - No Context configured to process this request," 에러를 보게 됩니다면 서버 로딩이 완전히 끝난 후에 툴바 옆의 Refresh 버튼을 클릭하시기 바랍니다.

또한 BookStoreServelet.java 파일에 마우스 오른 클릭하여 나오는 단축 메뉴에서 Web Run using "BookStoreServer."를 클릭해도 서블릿을 띠울 수 있습니다.

자, 이제 MobileBookStore 서블릿이 구동되며 J2ME 클라이언트가 입력한 값을 받을 준비가 다 되었습니다.

Running the client program

1 <JBuilder home>\Samples\Mobile\MobileBookStore\BookStoreJ2MEClient 디렉토리에서 BookStoreJ2MEClient.jpx 파일을 엽니다. .

2 현재 프로젝트에 대한 JDK 설정이 J2ME MIDP/CLDC JDK로 되어 있는지 확인합니다. 이는 Project|Project Properties선택하고 나오는 화면의 Path 탭에서 확인할 수 있습니다.

Note 어떻게 J2ME MIDP/CLDC JDK 설정을 하는지에 대한 내용은 "Setting up a JDK" on page 3-2.

3 Run|Configuration 선택.

4 Runtime Properties 대화 상자를 열어 Run 페이지의 Edit 버튼을 클릭합니다. 그리고나서 Main Class 필드 옆의 … 버튼을 클릭하여 com.borland.samples.micro.mobilebookstore.bookstoreJ2MEclient.BookStoreMIDlet 을 실행할 Main 클래스로 지정합니다. (또한 JAD 파일 라디오 버튼을 눌러 BookStoreJ2MEClient.jad 파일 선택하여 실행시킬 수도 있습니다.)

5 툴바의 Run 버튼 옆의 드랍 다운 버튼 클릭하고 BookStoreJ2MEClient 선택하여 클라이언트 프로그램을 실행시킵니다.

이 샘플에 대한 좀더 자세한 정보는 프로젝트와 함께 포함되어 있는 .html 노트를 참고하시기 바랍니다. 이 샘플이 위치하고 있는 디렉토리는 <JBuilder home>\samples\Mobile\MobileBookStore.

이 샘플에 대한 프로젝트 파일은 :

• <JBuilder home>\samples\Mobile\MobileBookStore\BookStore\BookStore.jpx

• <JBuilder home>\samples\Mobile\MobileBookStore\BookStoreJ2MEClient\BookStoreJ2MEClient.jpx.

JBuilder™ Mobileset 개발자 가이드

문서번호 : IMOBILE-JBM001

Appendix A

Tutorial : Creating and testing MIDlets

본 튜토리얼의 목적은 여러분에게 JBuilder를 이용하여 개발 및 디자인하는 툴들(마법사, 편집기, 컴파일러, MIDP 디자이너)을 이용하여 MIDlet을 생성할 수 있는 능력을 배양하기 위한 것입니다.

이 튜토리얼의 첫 단계를 통해 여러분은 간단한 사용자 인터페이스를 포함하는 기본 MIDlet 을 생성하시게 됩니다. MIDlet을 생성한 후, 여러분은 JBuilder 안의 에뮬레이터에서 MIDlet을 테스트 해 볼 수 있게 됩니다.

Note 본 튜토리얼 안에 있는 에뮬레이터와 폰UI 이미지는 예제를 위한 목적으로 사용되었을 뿐 특정 기기를 홍보하지는 않습니다.

First : MIDlet을 생성하기 위한 기본 단계는 다음과 같습니다.

1. Project 마법사를 이용하여 프로젝트를 생성합니다. 프로젝트는 프로젝트 파일과 프로젝트에 관해 갖추고 있어야 할 내용을 담고 있는 HTML 파일을 담고 있습니다.
2. MIDlet 마법사를 이용하여 MIDlet을 생성합니다. MIDlet 마법사는 MIDlet 클래스와 사용자 인터페이스를 보여주는 Displayable 클래스를 생성하게 됩니다.
3. MIDP 디자이너를 이용하여 사용자 인터페이스의 생성합니다.
4. 여러분의 프로젝트를 컴파일하고 저장합니다.
5. 에뮬레이터 안에서 MIDlet를 실행하여 테스트합니다.

Second : 본 튜토리얼의 두 번째 단계에서 여러분은 어떻게 프로젝트에 MIDlet을 추가, archive 빌더를 이용해서 어떻게 archive JAR와 JAD의 생성하는지를 알 수 있으며, 에뮬레이터에서 JAD 파일을 실행해 봄으로써 MIDlet suite을 테스트하게 됩니다.

1. MIDlet을 추가합니다.
2. 새로운 MIDlet을 위한 사용자 인터페이스를 디자인합니다.
3. MIDlet을 테스트합니다.
4. MIDlet suite을 archive 합니다.
5. JAD 파일로부터 실행합니다.

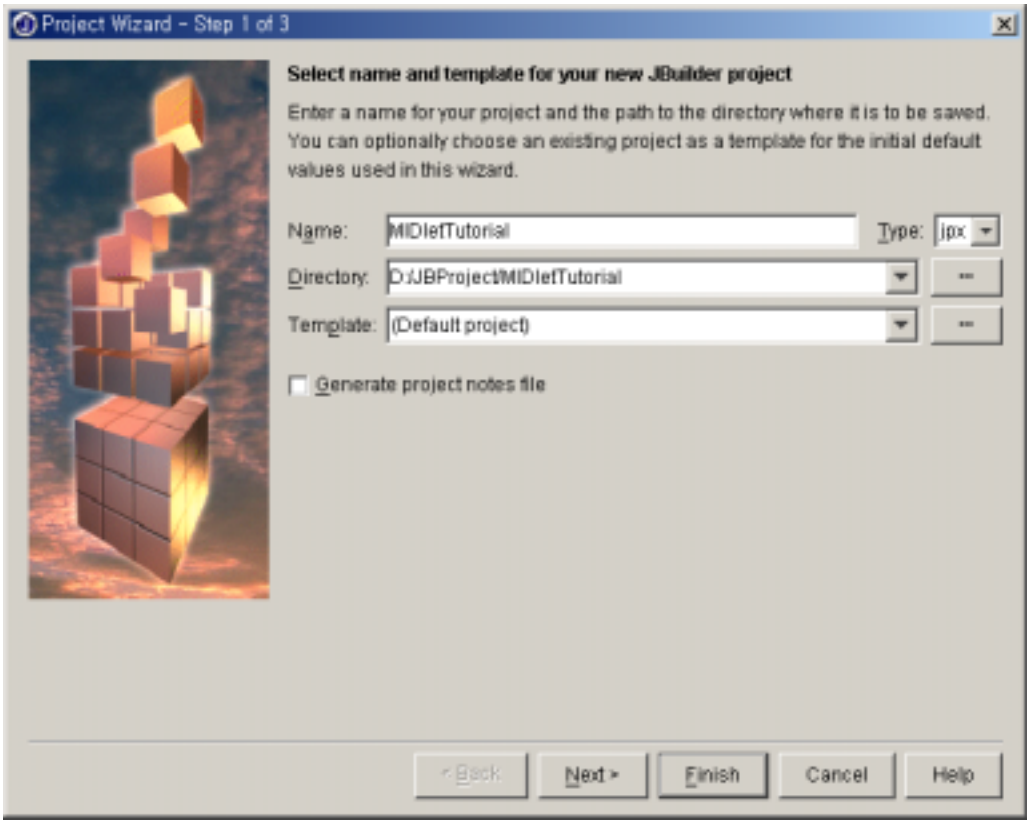
Important 이 튜토리얼은 여러분이 이미 필요한 JDK 가 설치되어 있고, JBuilder 가 제대로 설정되어 있습니다고 가정하고 진행하고 있습니다. 만약에 설정이 되어 있지 않다면, 페이지3-2에 잇는 JDK 세팅을 참조하기 바랍니다.

프로젝트 생성

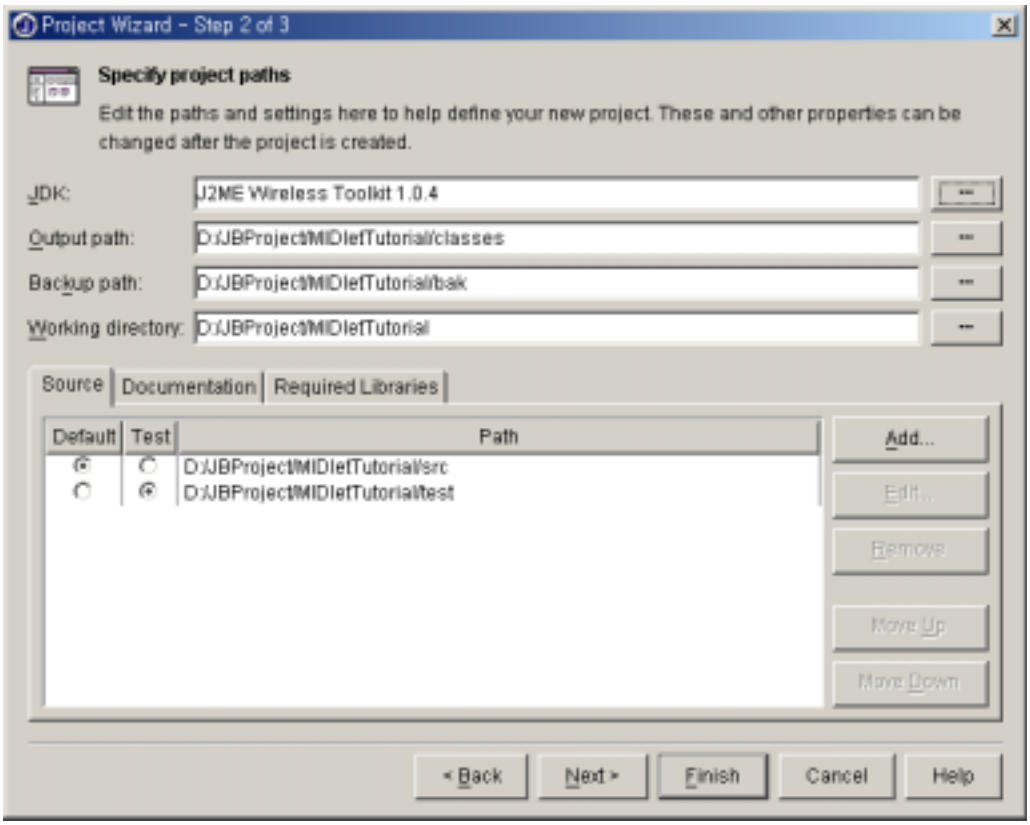
여러분이 JBuilder에서 MIDlet을 생성하기 전에 여러분은 우선 프로젝트를 생성해야만 합니다.

Note MDIet 마법사를 실행하기 위해서는 프로젝트가 열려 있어야 합니다.

1. File | New Project를 선택하여 프로젝트 마법사를 시작합니다.
2. 기본 프로젝트 이름을 “MIDletTutorial”로 바꿉니다. 프로젝트 이름에는 빈칸이 있으면 안 됩니다.
3. 나머지는 모두 기본값으로 두기 바랍니다. 만약 여러분이 “Generate project notes file”을 체크하게 되면 여러분이 생성한 프로젝트에 대해 기본적인 기록 사항들이 생성될 것입니다.

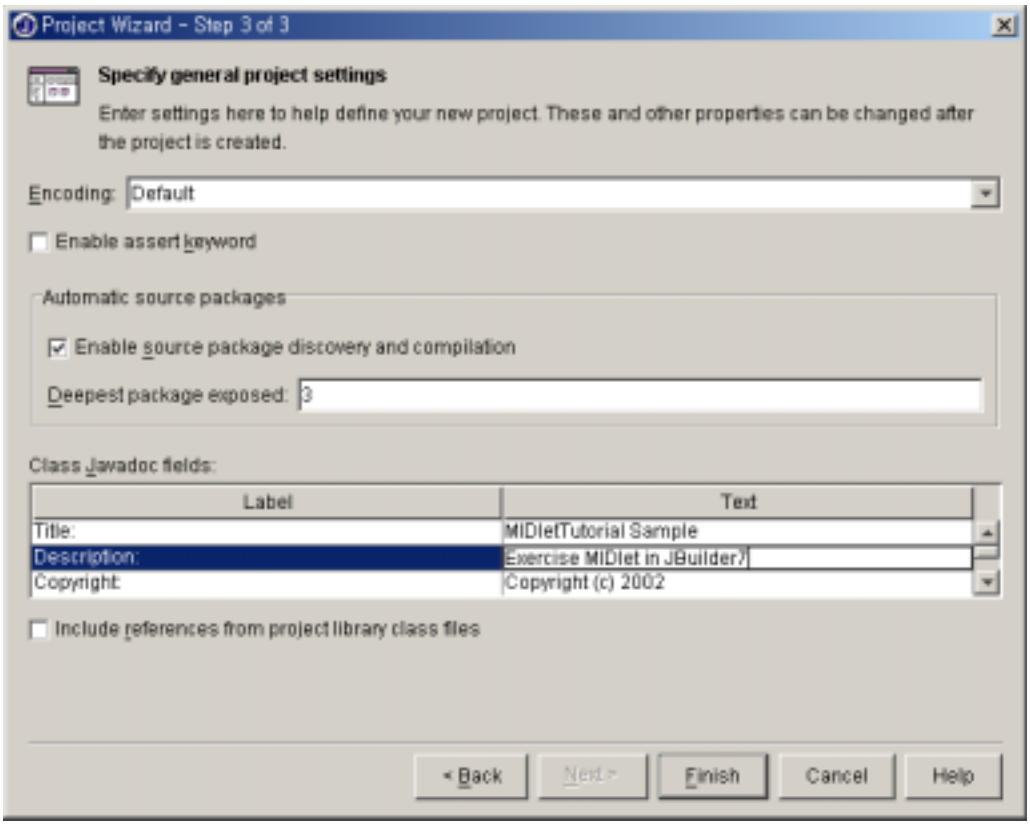


4. Next를 클릭합니다.
5. Step 2에서 프로젝트에 맞게 디렉토리와 경로를 수정해 줍니다.



6. 원하는 JDK를 찾기 위해  를 눌러 JBuilder에서 기본적으로 설치되는 JDK 이외 버전의 JDK를 찾습니다.
7. Next를 클릭합니다.
8. Step 3에서 여러분이 원하는 추가적인 세부사항을 입력합니다.
9. 프로젝트 생성을 위해 Finish를 클릭합니다.

여러분이 이 과정을 끝마치고 나면, MIDletTutorial.jpx 파일이 생성되고 이와 관련된 사항을 나타내는 MIDletTutorial.html파일이 생성됨을 볼 수 있습니다(위저드 1 단계의 체크 박스 항목을 선택했을 경우).

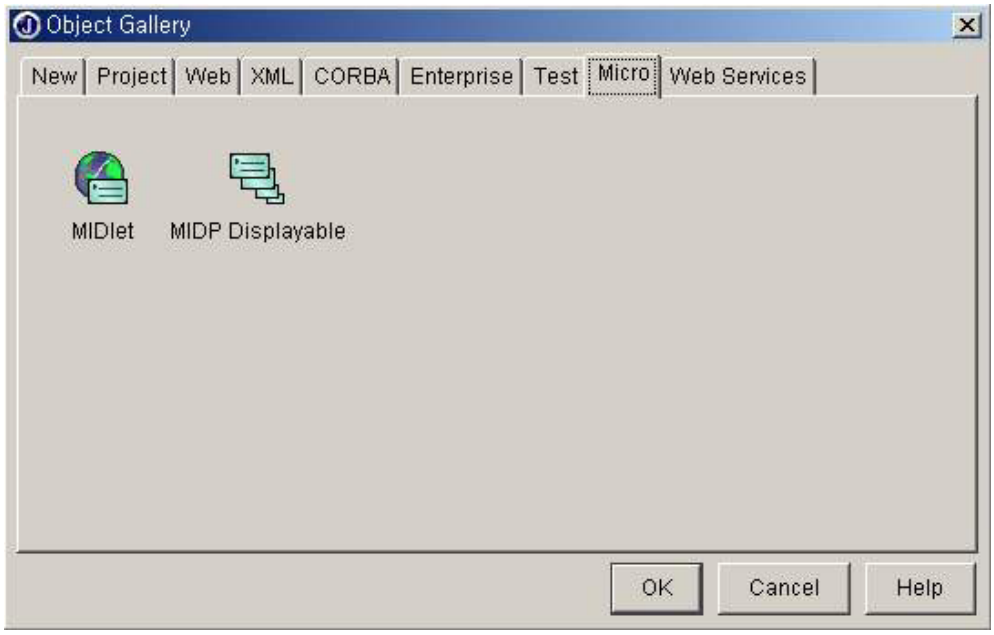


MIDlet1 생성

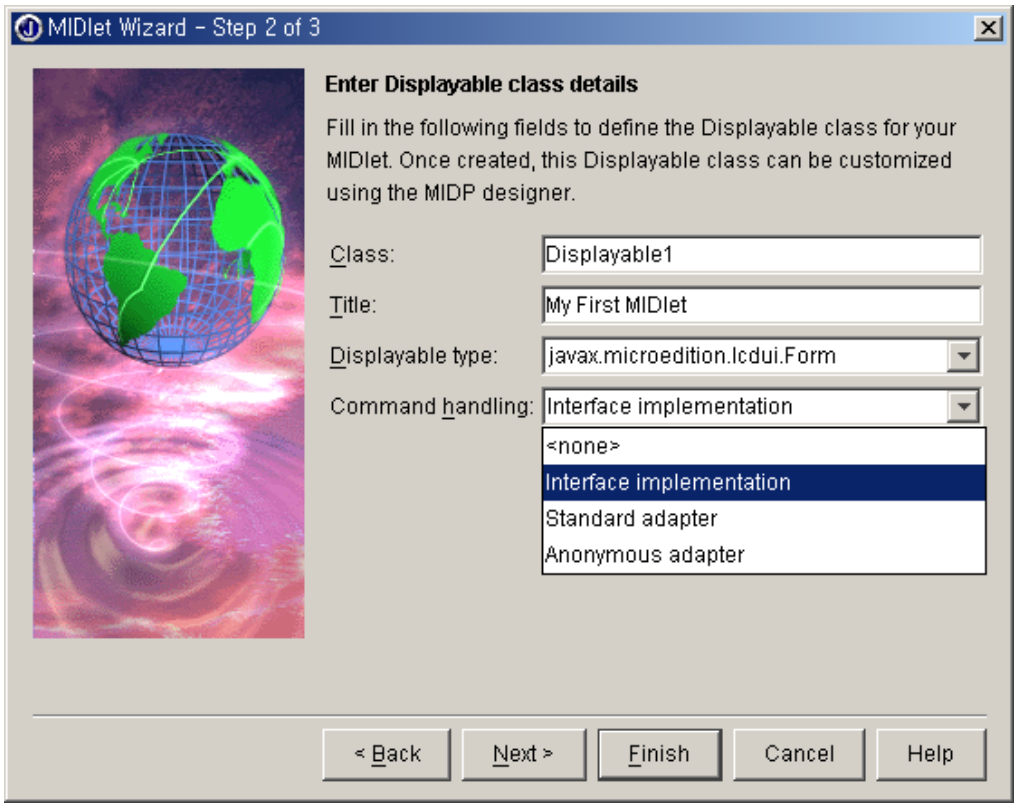
이제 여러분은 새로운 프로젝트를 생성하여 열어 놓았으니 새로운 MIDlet을 생성해 볼 것입니다.

1. File | New 를 선택하여 Object 갤러리를 오픈합니다.
2. Micro 탭을 클릭하여 MIDlet 아이콘을 더블 클릭하여 MIDlet 마법사를 오픈합니다.

※ Micro 탭의 Object의 활성화를 위해서는 MIDP/CLDC JDK를 추가 설치해 주어야 합니다. (CH3장의 튜토리얼을 참고하기 바랍니다.)

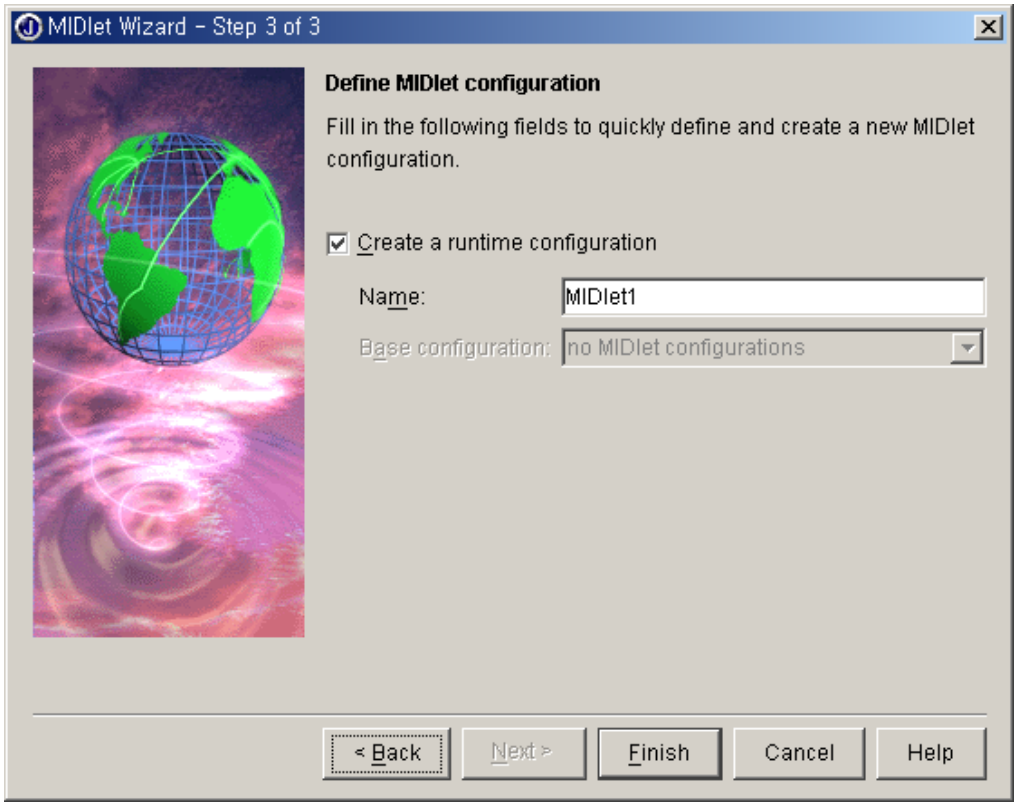


3. Step 1에서 MIDlet 클래스의 이름은 그대로 MIDlet1으로 놔두고 “Generate Header ComMent”를 체크하고 Next를 클릭합니다.
4. Step 2에서 클래스 이름은 Displayable1으로 놔두고, 타이틀만 “My First MIDlet”으로 변경합니다. 이 타이틀은 여러분이 모바일 기기에 여러분의 MIDlet을 실행할 때 MIDlet의 윗 부분에 나타나게 됩니다.



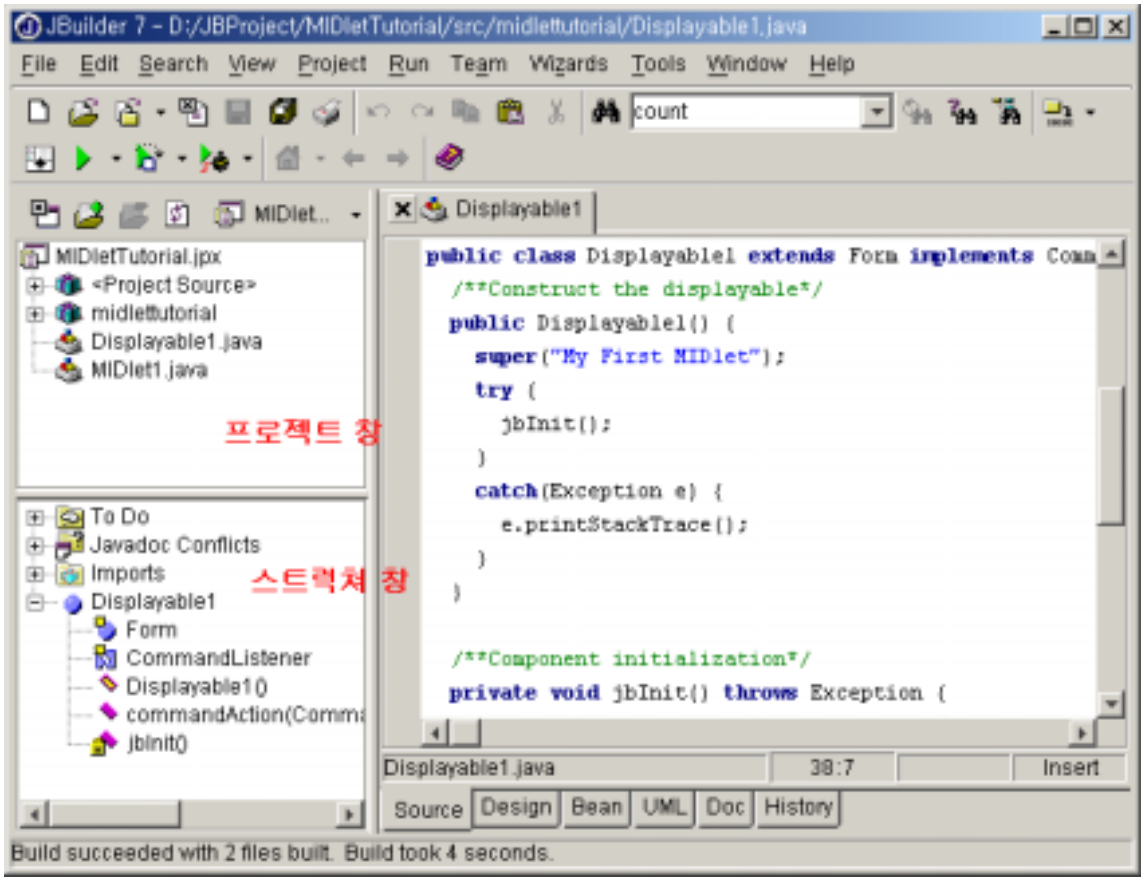
Displayable type은 확장한 클래스를 의미합니다. Displayable type이 javax.microedition.lcdui.form으로 기본 설정되어 있나 확인합니다. J2ME 스크린은 자체적으로 보여줄 수 없기 때문에 여러분은 Form, Canvas, List, TextBox와 같은 서브 클래스를 이용해야 합니다. Form만이 사용자 인터페이스 요소들을 포함하는 클래스이기 때문에 우선 Displayable type에 Form을 선택합니다. 확인한 후에 Next 클릭합니다.

5. Step3 에서는 런타임 설정을 확인합니다. 아래 그림과 같이 Create a runtime configuration 체크 박스를 선택하고 런타임 컨피규레이션 이름을 디폴트인 MDIlet1으로 지정합니다. MIDlet을 실행시키기 위해서는 적어도 하나 이상의 런타임 컨피규레이션을 지정해야 합니다. 런타임 환경 설정에 대한 좀 더 자세한 내용은 Building Application with JBuilder 의 “Setting Runtime Configuration” 을 참고하시기 바랍니다.



6. Finish 버튼을 클릭합니다.

아래 그림과 같이 MIDletTutorial 프로젝트파일이 JBuilder7 의 프로젝트 창에 보이고 Displayable1 파일이 컨텐트 창에 나타나 편집과 디자인을 할 준비를 갖추게 됩니다. 스트럭쳐 창을 통해서는 Displayable1의 구성 요소들을 종류별로 확인하실 수 있습니다.



7. 툴바의 Save All 버튼 눌러 프로젝트의 모든 파일을 저장합니다.

MIDlet1 을 위한 사용자 인터페이스 디자인

여러분의 MIDlet을 위한 사용자 인터페이스를 생성하기 위해 여러분은 Form 까지 확장되는 Displayable1을 디자인 하게 될 것입니다.

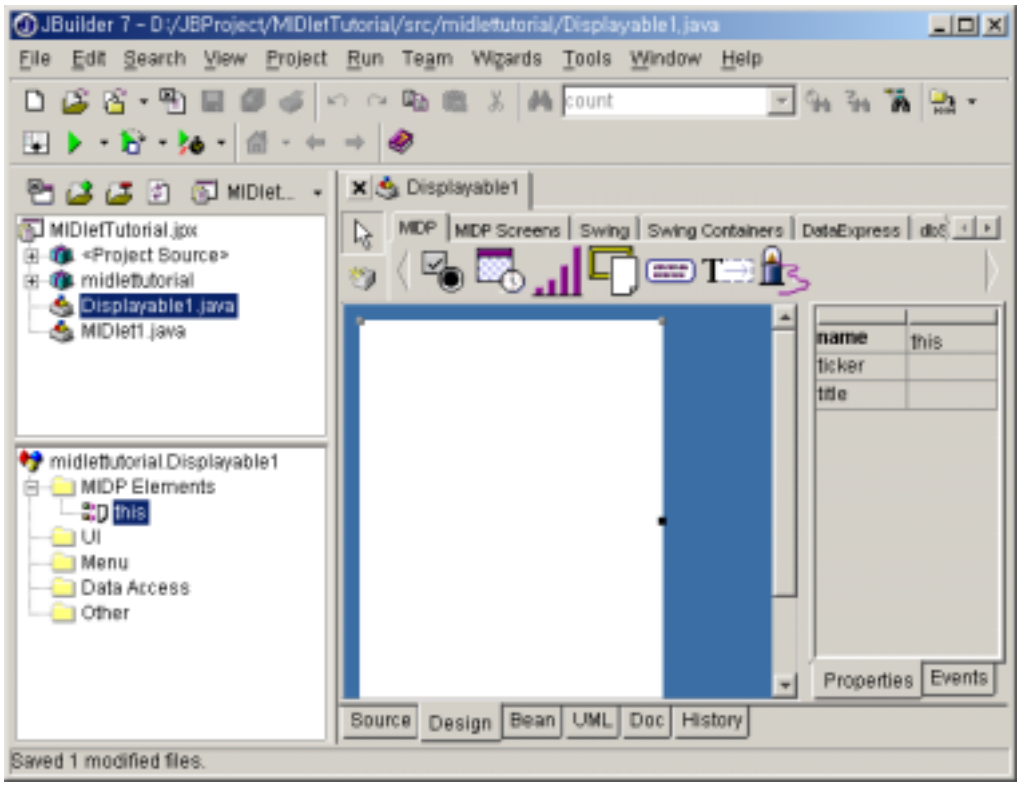
MIDP 는 유일하게 하나의 컨테이너 컴퍼넌트를 갖는데 그것이 바로 Form입니다. Form은 다중의 컴포넌트들을 포함할 수 있으나 그들의 위치를 컨트롤 할 수는 없습니다. 표준 Java와 달리, MIDP 디자인을 할 때, 여러분은 Layout manager를 이용할 수 없습니다. 그러나 여러분은 마우스를 이용하여 그들을 드래그 업 & 다운 함으로써 Form에 있는 컴퍼넌트들의 순서를 swap 할 수 있습니다.

여러분이 생성하고자 하는 MIDlet의 사용자 인터페이스에는 StringItem, DateField, Ticker 를 포함하게 될 것입니다.

Displayable1이 프로젝트 창에 열려 있을 것입니다 (만약, 그렇지 않다면 Diaplayable1.java를 더블 클릭하기 바랍니다.)

1. MIDP 디자이너를 열기 위해 Displayable1의 아래에 있는 Design 탭을 클릭합니

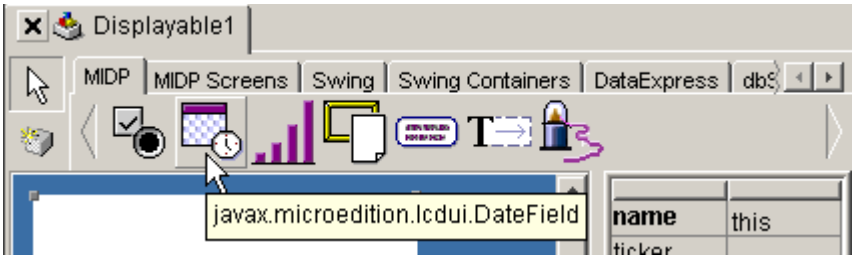
다.



MIDP 디자이너의 윗부분의 컴퍼넌트 팔레트를 보면, MIDP 스크린과 MIDP UI 컴퍼넌트를 위해 이미 세팅이 되어 있음을 볼 수 있습니다.

Form 은 MIDP 디자이너에 나타나 있고 왼편 아래에 있는 MIDP Element 폴더에는 “this”라고 나타나 있는 것을 볼 수 있습니다.

2. 팔레트에 MIDP 에 있는 컴포넌트들 위에 마우스를 가져가면 툴팁 기능이 작동하게 되어 몇 초후에 컴포넌틀르의 이름이 나타나게 됩니다. 이 중에서 StringItem, Datefield, Ticker를 찾기 바랍니다.

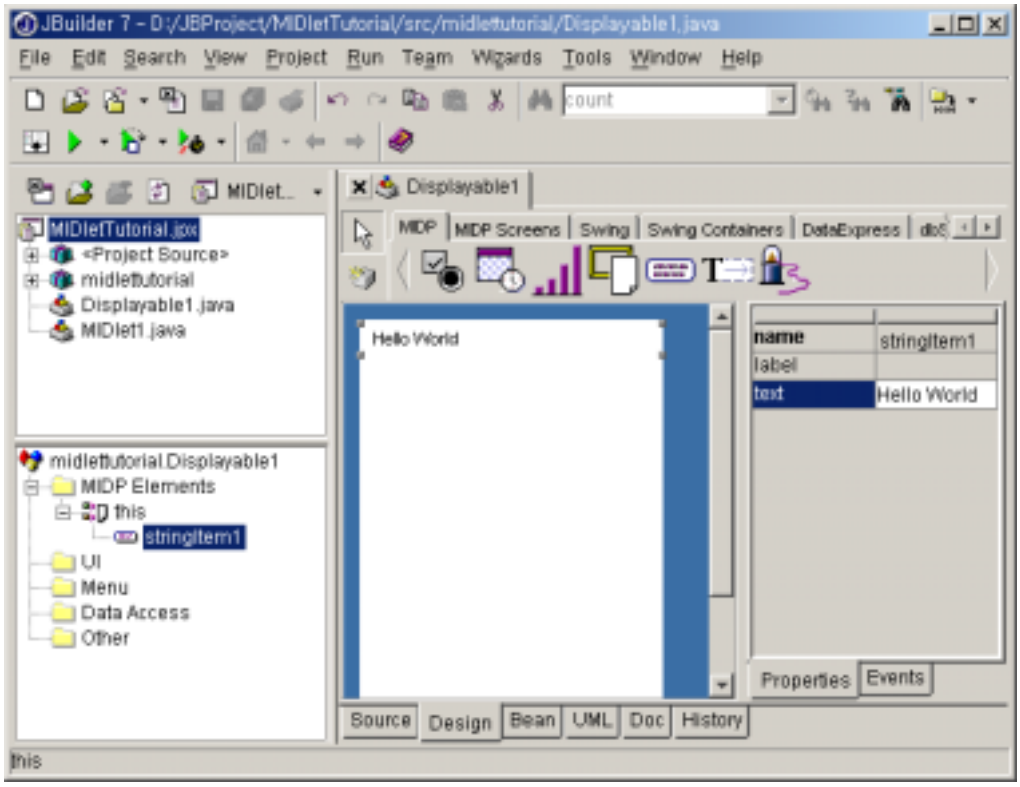


StringItem 추가하기

여러분이 가장 먼저 추가하고자 하는 아이템은 “Hello World”를 보여줄 StringItem

입니다.

1. 컴포넌트 팔레트의 MIDP 페이지에서 StringItem을 찾아 클릭한 후, Form 위에 아무 곳에서나 클릭하면 추가 됩니다. 그러면, Form 위에 추가된 StringItem이 왼쪽 아래에 있는 컴포넌트 트리에서 “this”라는 Form에 추가된 StringItem이 있음을 볼 수 있게 됩니다.
2. Inspector에서 StringItem1의 속성 중 Text를 Hello World로 변경하고 Enter 를 치면, 곧바로 디자이너에서 StringItem1의 위치에 여러분이 기입한 “Hello World”가 생기는 것을 볼 수 있습니다.



3. 소스 코드로 돌아 가기 위해 Source 탭을 클릭합니다. 여러분이 지금까지 디자이너에서 작업한 내용이 어떻게 JBuilder가 여러분의 클래스에 자동적으로 코드를 추가하게 되었는지를 살펴보기 바랍니다.

```
package midlettutorial;

import javax.microedition.lcdui.*;

public class Displayable1 extends Form implements CommandListener {
StringItem stringItem1;
```

```
    /**Construct the displayable*/
    public Displayable1() {
        super("Displayable Title");
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**Component initialization*/
    private void jbInit() throws Exception {
        // set up this Displayable to listen to command events
        stringItem1 = new StringItem("", "");
        stringItem1.setText("Hello World");
        setCommandListener(this);
        // add the Exit command
        addCommand(new Command("Exit", Command.EXIT, 1));
        this.append(stringItem1);
    }

    /**Handle command events*/
    public void commandAction(Command command, Displayable displayable) {
        /** @todo Add command handling code */
        if (command.getCommandType() == Command.EXIT) {
            // stop the MIDlet
            MIDlet2.quitApp();
        }
    }
}
```

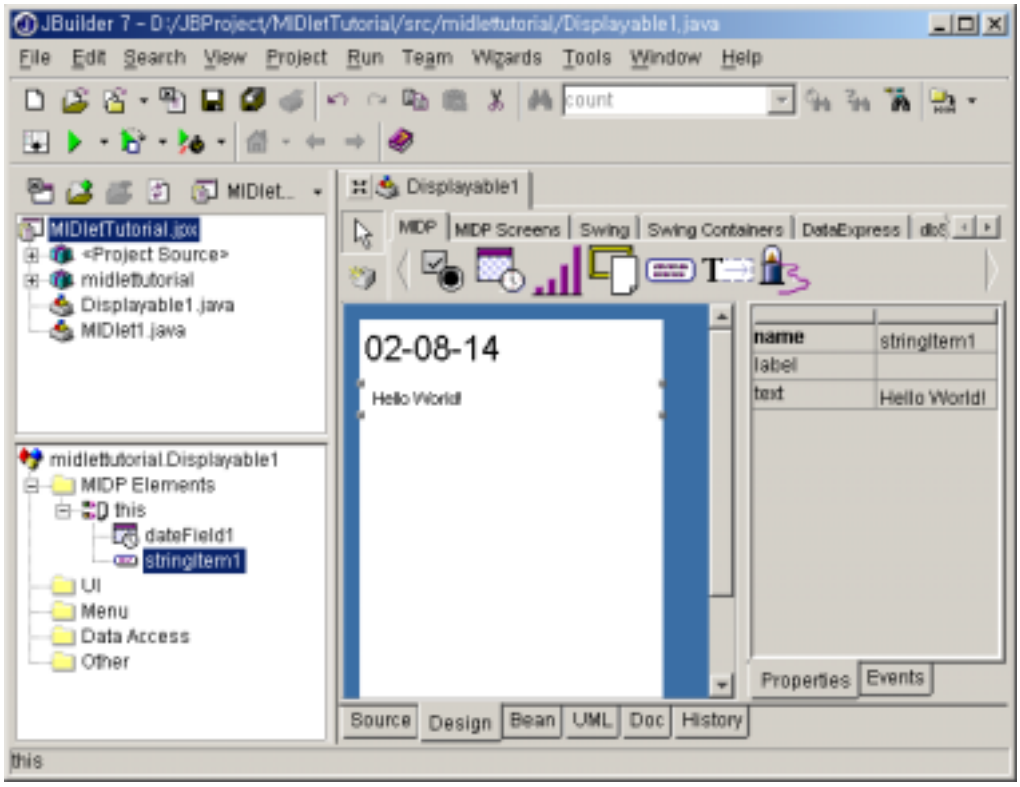
4.다시 디자이너로 돌아갑니다.

DateField 추가하기

다음으로 여러분은 디자인에 DateField를 추가하고, 위치를 StringItem과 바꾸게 될 것입니다.

1. 팔레트에서 DateField 컴포넌트를 클릭한 후 Form에 추가합니다.

2. StringItem을 선택하여 드래그 다운하여 DateField와 위치를 바꿉니다.



DateField와 StringItem의 위치를 맞바꾼 결과 그림은 위와 같습니다. 디자이너에서의 순서 변동에 따라 스트럭쳐 창에서 컴포넌트 트리 순서가 함께 바뀌게 됩니다.

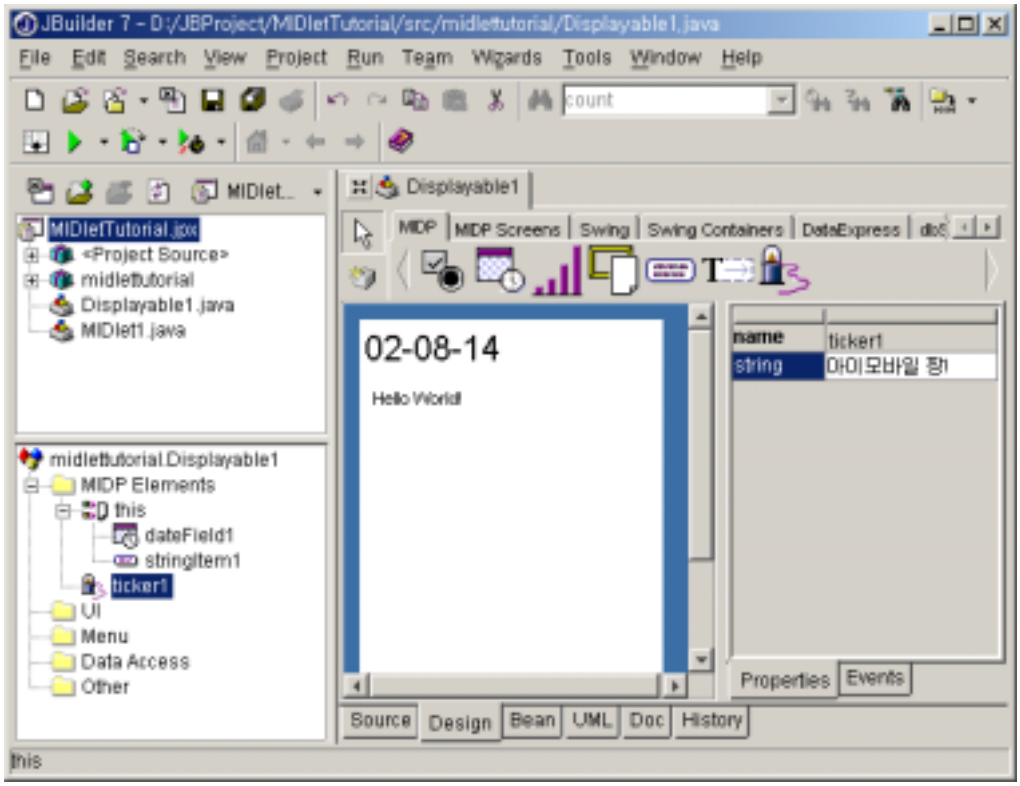
Ticker 추가하기

마지막으로 “아이모바일 짱 ! ” 이라는 텍스트를 디스플레이 하는 Ticker를 추가합니다. StringItem, DasteField 추가하기와 동일한 방법으로 추가하면 됩니다.

Ticker 컴포넌트는 StringItem, DateField처럼 Form 위에 나타나지 않습니다. Ticker는 단지 스크린의 속성일 뿐이므로 컴포넌트 트리에서만 확인할 수 있습니다. 또

한 여러분은 Ticker의 Display를 컨트롤할 수 없습니다. 이것은 단지 디바이스 스크린에 의해 컨트롤 됩니다.

1. 컴포넌트 팔레트에서 Ticker 컴포넌트를 찾아 클릭한 후 폼에 마우스 클릭하여 프로젝트에 추가합니다.
2. 아래 그림과 같이 Ticker1 컴포넌트의 String 프로퍼티 값을 “아이모바일 짱!” 으로 변경합니다.

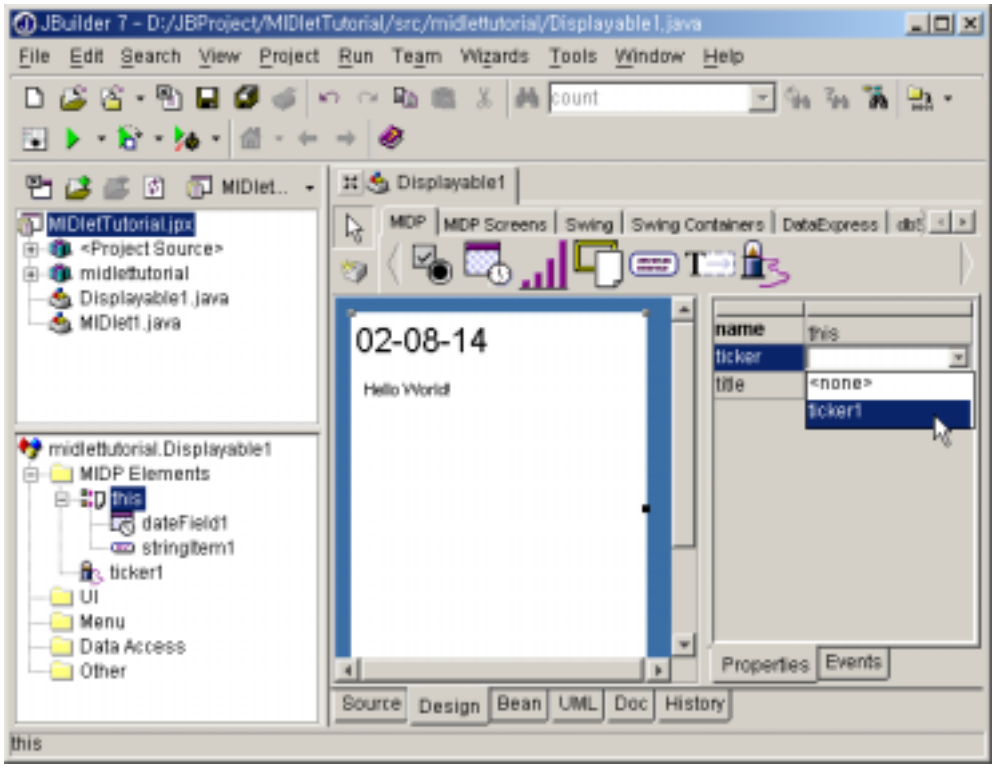


3. 컴포넌트 트리의 this를 선택한 후에 오른편의 Inspector에서 ticker 프로퍼티 값을 ticker1으로 변경합니다. 아래 그림처럼 하면 됩니다.

이와 같은 과정을 거치면 실제 소스 코드에 다음과 같인 라인이 추가 됩니다.

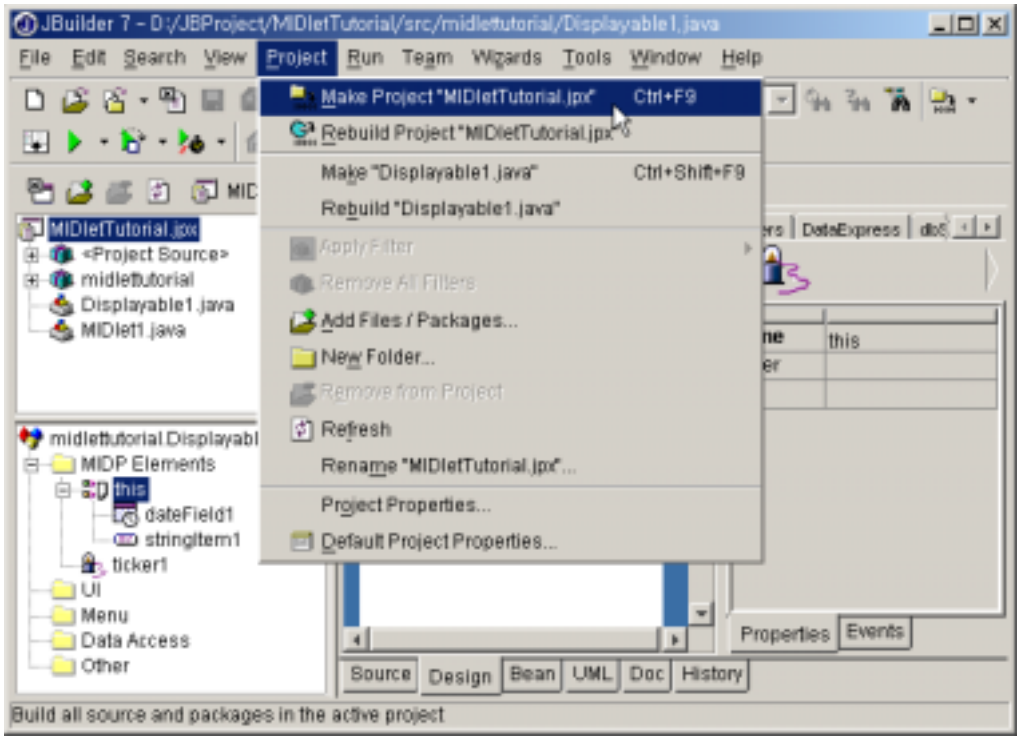
```
this.setTicker(ticker1);
```

이제 Displayable1 에 Ticker가 나타나게 되었습니다.



프로젝트 컴파일 및 저장하기

여러분의 프로젝트를 컴파일하기 위해 Project | Make Project “MIDletTutorial.jpx”를 선택하거나 Ctrl+F9을 누르면 됩니다.



MIDlet1 테스트하기

에뮬레이터에 MIDlet1을 실행해 봄으로써 여러분이 작성한 MIDlet1을 테스트 해 볼 수 있습니다.

1. MIDlet을 실행시키는 방법에는 다음과 같은 세가지 방법 있습니다
   - JBuilder 메뉴에서 Run | Run Project 선택하거나 F9을 누르면 됩니다.
   - 툴바의 Run 아이콘을  클릭합니다.
   - 프로젝트 창에서 MIDlet 파일에 마우스 오른 클릭하고 나오는 단축 메뉴에서 Micro Run  을 선택합니다.
2. 실행하면 에뮬레이터가 실행되면서 여러분이 작성한 MIDlet이 에뮬레이터에서 작동하는지 확인할 수 있습니다.

위 그림과 같이 나오는 까닭은 아직 날짜와 시각 세팅이 되어있지 않기 때문입니다. 에뮬레이터 조작을 통해 날짜와 시각을 세팅하면 다음 그림과 같이 나타나게 됩니다.

에뮬레이터 실행을 멈추기 위해서는 에뮬레이터의 Exit 문자 아래 버튼을 클릭하거나 에뮬레이터 창에서 File | Exit를 선택합니다.

추가적인 MIDlet 작성하기

여러분은 이제 추가적인 MIDlet을 프로젝트에 추가하고 나서 에뮬레이터에서 모든 MIDlet을 한 번에 테스트 하는 것을 보게 될 것입니다. 이것은 여러분이 MIDlet suite을 묶어서 JAD파일에서 실행을 함으로써 가능하게 됩니다.

먼저 여러분의 프로젝트에 두 개 이상의 MIDlet 을 생성합니다.

1. File | New 클릭하면 나오는 Object Gallery에서 MIDlet 마법사를 더블 클릭합니

다.

2. Default값으로 MIDlet2를 생성하기 위해 Finish를 누른다.
3. MIDlet3 를 생성하기 위해 Step1 과 Step2를 반복합니다.
4. 여러분의 프로젝트를 Rebuild 하고서 저장합니다.

새로운 MIDlet을 위한 UI 디자인하기

이 튜토리얼을 수행하기 위해 우리는 MIDlet2와 MIDlet3에 대한 각각의 Displayable에 UI 컴퍼넌트를 집어 넣게 될 것입니다. 이 튜토리얼의 목적은 에뮬레이터에 서로 구분이 용이한 세 개의 MIDlet을 만드는 데에 있습니다.

프로젝트에 이미지 추가하기

MIDlet2는 스크린에 이미지 파일을 보여주게 될 것입니다. 이 이미지는 MIDP 디자이너를 이용하여 UI를 추가할 수 있으나 여러분은 먼저 프로젝트 소스 디렉토리 안에 이미지를 복사해 넣어 두어야 합니다. 그리고 나서 JBuilder에 여러분의 프로젝트를 추가 시켜야 합니다.

1. File | Open File을 선택하여 <JBuilder>\samples\mobile\Midp_UIDemo\src \com\Borland\JBuilder\samples\micro\uidemo\ivanIcon.png를 확인하고 OK를 클릭합니다. JBuilder는 콘텐츠 창에 이 그래픽을 열게 될 것입니다.

2. File | Save As를 선택하고 디렉토리 필드에서  를 선택하여 여러분의 프로젝트의 소스 디렉토리를 찾습니다.

(Ex : <JBuilder home>\jbproject\MIDlettutorial\src\midlettutorial)

3. “Add Saved File to Project”에 체크하고 OK를 클릭합니다.이미지 파일은 프로젝트에 추가되면서, 프로젝트 Pane에 추가된 것을 볼 수 있습니다.

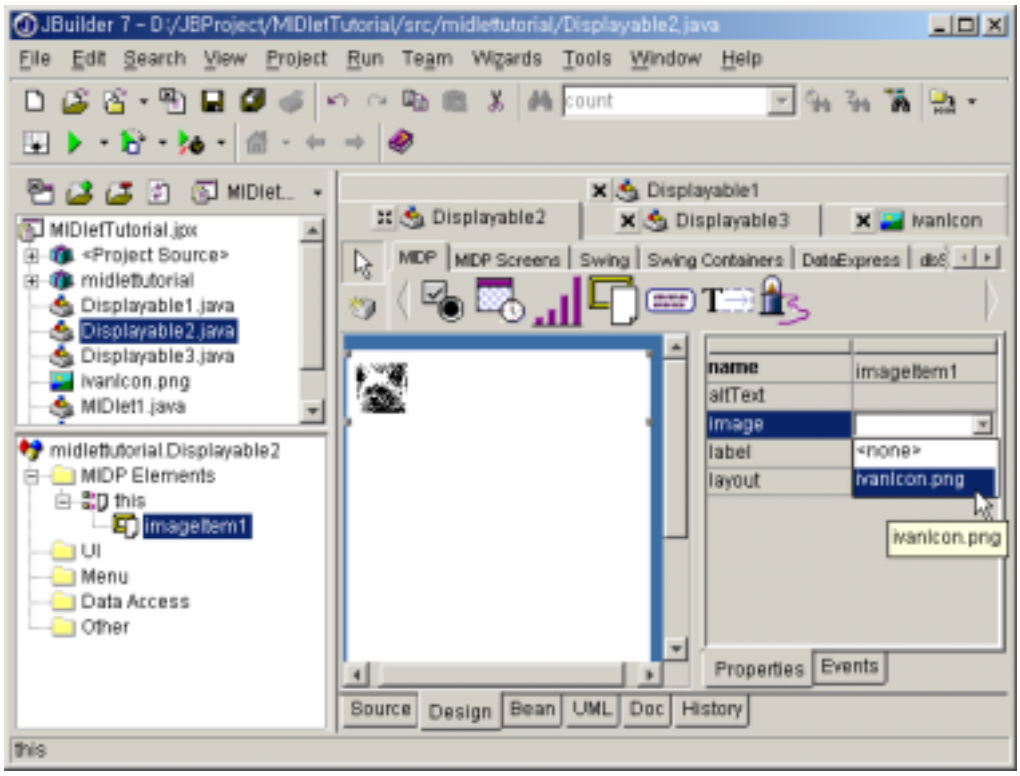
Important MIDP 스펙에 따라, 이미지는 반드시 .png 포맷 형태이어야 합니다.

Displayable2에 이미지 넣기

이제 이미지가 여러분의 프로젝트에 물리적으로 추가되었습니다. 여러분은 이미지를 MIDP 디자이너를 이용하여 UI를 추가할 수 있게 되었습니다.

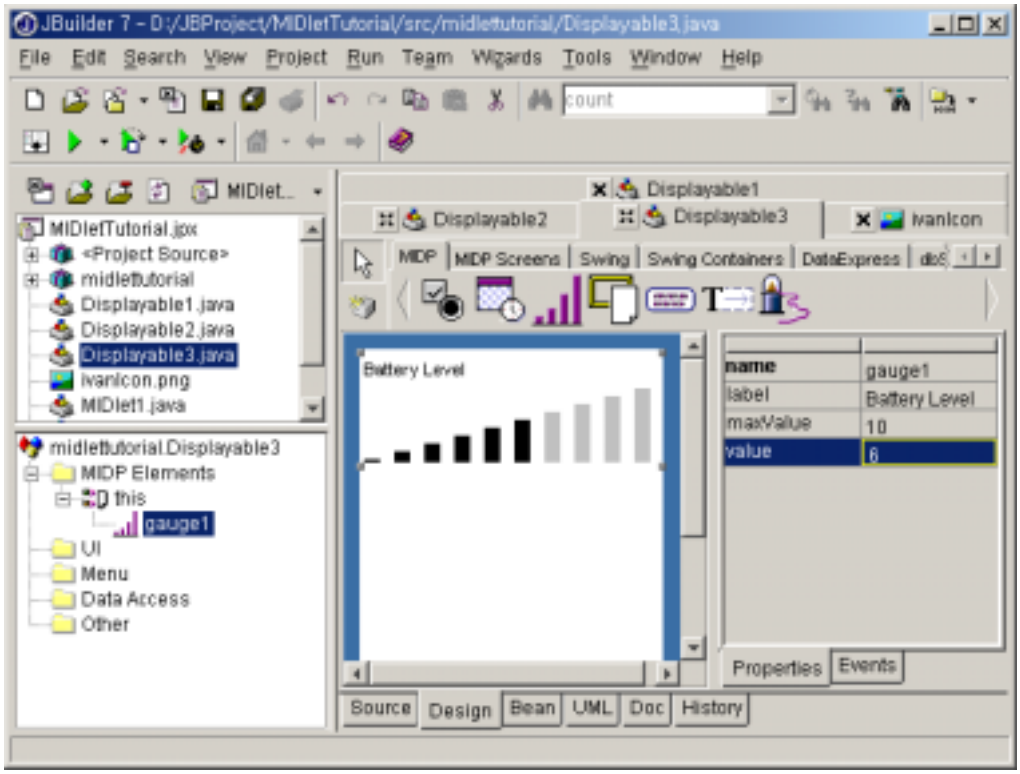
1. Displayable2에 대한 Design 탭을 누른다.

2. 컴퍼넌트 팔레트의 MIDP 페이지에서 ImageItem을 선택하여 추가합니다. 여러분은 이미지를 사용하게 될 때까지 아무런 변화를 볼 수 없을 것입니다.
3. Inspector에서 image 속성을 클릭하여 drop-down 리스트에서 IvanIcon.png를 선택합니다.



Displayable3에 게이지 추가하기

MIDlet3에 대해 우리는 MIDP 게이지를 추가하게 될 것입니다.
1. Displayable3의 디자인 탭을 클릭하여 UI 디자인에 MIDP 게이지를 추가합니다.
2. Label에 Battery Level 이라고 입력하고 maxValue 프로퍼티 값은 10으로 Value 프로퍼티 값은 6으로 세팅합니다. 아래 그림과 같습니다.

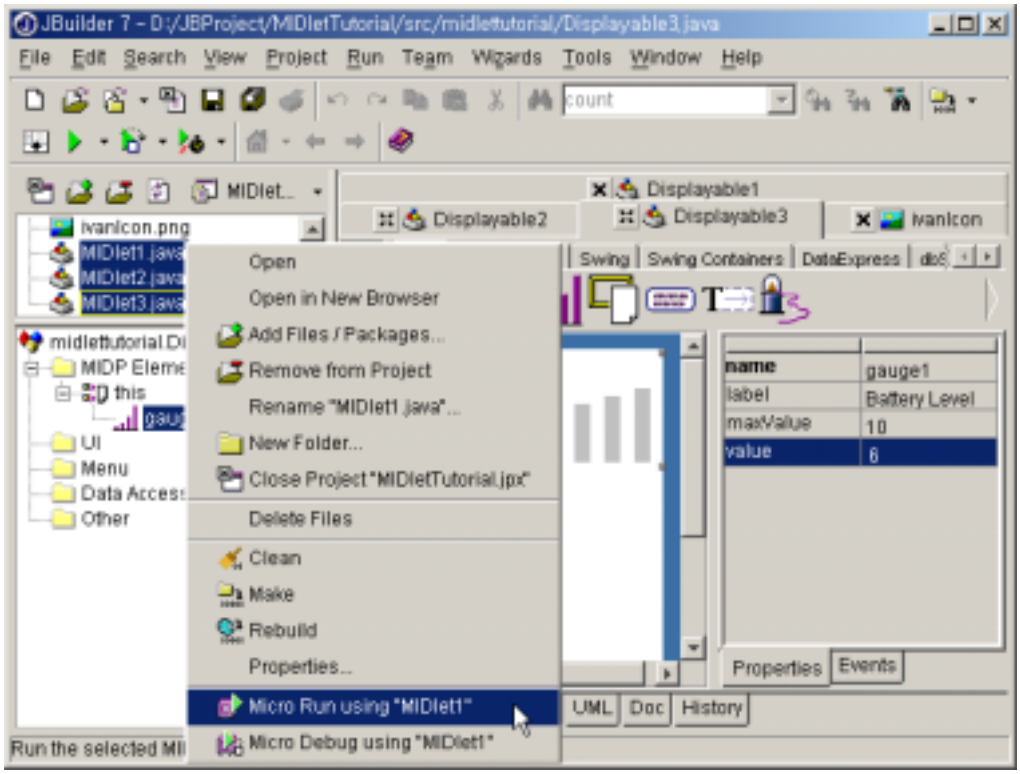


3. 리빌드 한 후 프로젝트를 저장합니다.

MIDlet 테스트 하기

MobileSet은 여러분이 에뮬레이터 내부에서 복수개의 MIDlet을 한번에 테스트 할 수 있게 해 줍니다.

1. MIDlet1을 클릭한 후, Shift키를누르고 MIDlet3까지 클릭하여 동시에 선택합니다.
2. 마우스 오른쪽 버튼을 눌러 Micro-Run을 실행합니다.



실행 결과입니다.

MIDlet suite Archive 하기

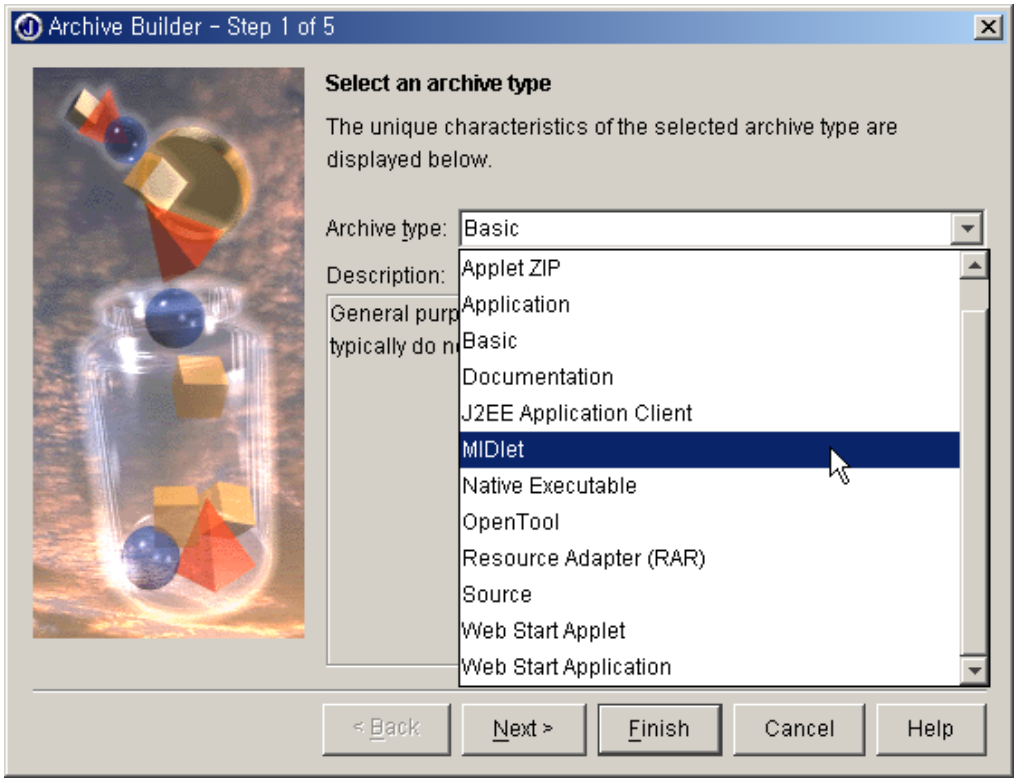
여러분이 개발한 MIDlet을 모바일 디바이스에 배치하기 위한 첫 번째 단계로 여러분은 우선 MIDlet suite을 모아서 JAD 파일을 에뮬레이터에서 테스트 해 봐야 합니다. 여러분이 실제로 MIDlet suite을 디바이스에 배치할 때에는 여러분은 단지 JAR, JAD 파일만 다운로드 받게 될 것입니다. JBuilder SE와 Enterprise는 Archive Builder 마법사를 이용하여 매우 쉽게 Archive 과정을 수행하게 해 줄 것입니다.

Note Archive Builder는 JBuilder SE와 Ent만의 특징 중의 합니다이다. 만약 여러분이 JBuilder Personal 버전을 가지고 있다면 이 기능을 테스트 해 볼 수는 없을 것입니다. Personal 버전에는 마법사 기능이 없기 때문에 수동으로 JAR, JAD 파일을 Command Line에서 생성해 줘야 합니다. 그러나 여러분이 만약 Nokia JDK를

이용하고 있고 Nokia Developer's Suite를 JBuilder에 설치하였다면, 이 툴을 이용하여 JBuilder Personal에서도 MIDlet suite를 Archive 할 수 있습니다.

JBuilder7 SE와 Enterprise Edition .에서 Archive Builder를 이용하려면 다음과 같이 하면 됩니다.

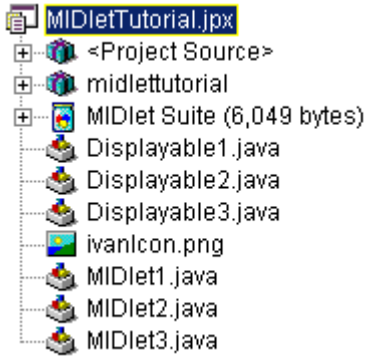
1. Wizards | Archive Builder를 선택합니다.
2. Archive Type에서 Drop-Down 화살표를 눌러 MIDlet을 선택합니다.



3. Next를 클릭합니다.
4. MIDlet suite 이름을 기입하고 마칩니다.

(JBuilder는 자동적으로 필요한 정보들을 모을 것입니다.)

5. 프로젝트를 Rebuild하게 되면 Project Pane에서 MIDlet suite가 보일 것입니다.

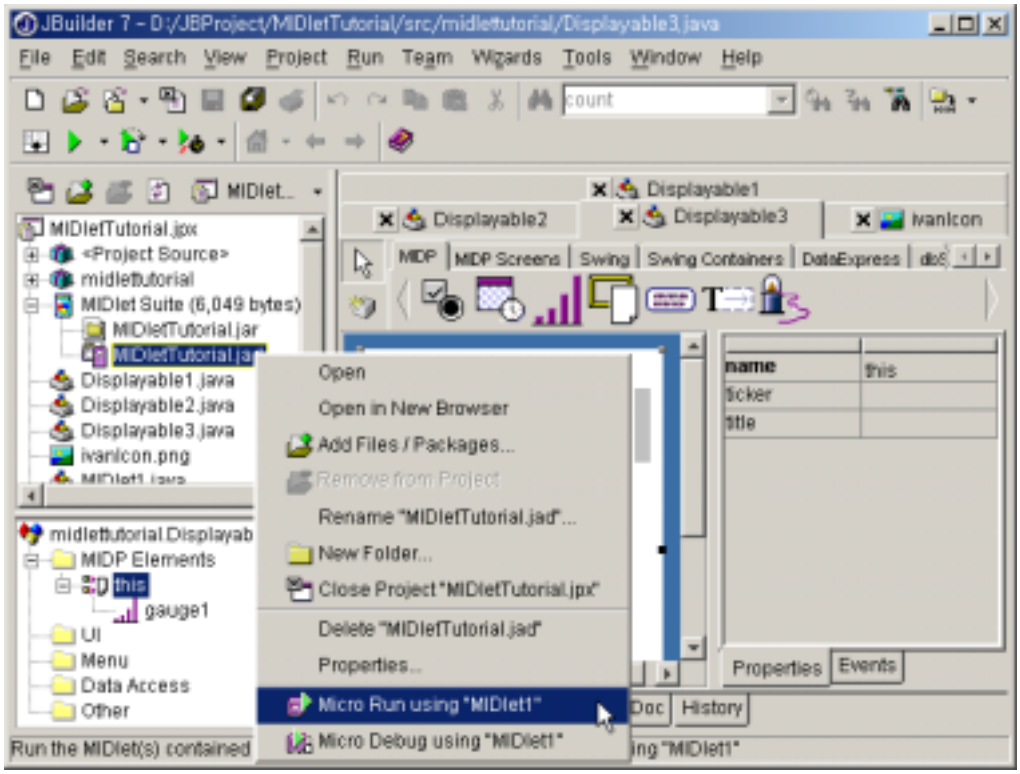


JAD 파일 실행하기

JAD 파일로부터 MIDlet suite를 테스트하는 방법은 두 가지가 있습니다.

Using MicroRun

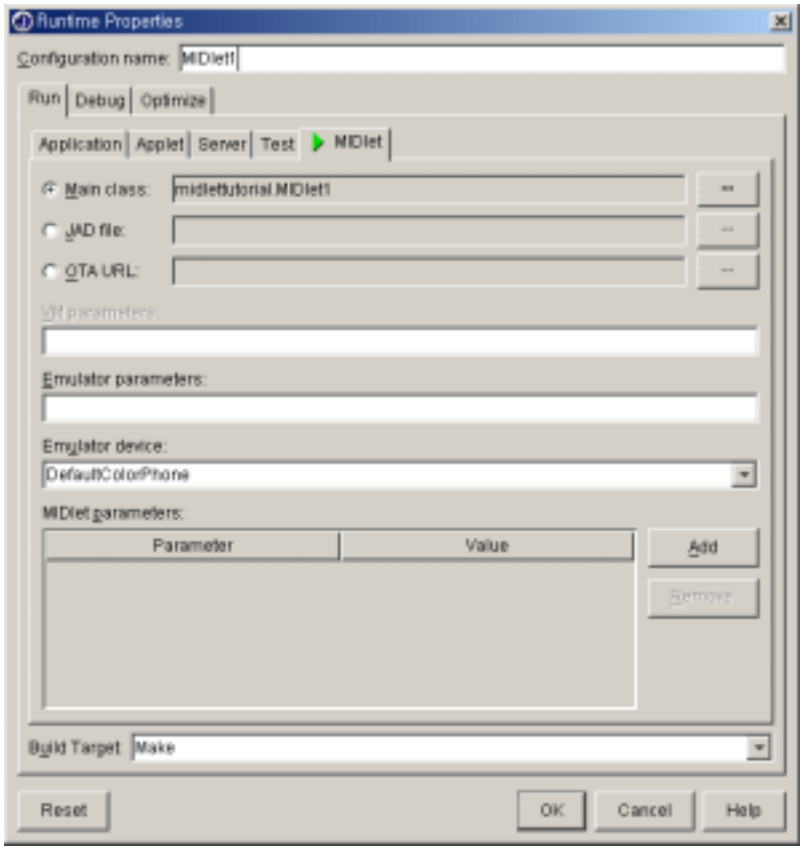
1. Archive 노드를 펼친다.
2. JAD파일에 오른쪽 클릭하여 Micro-Run을 선택합니다.



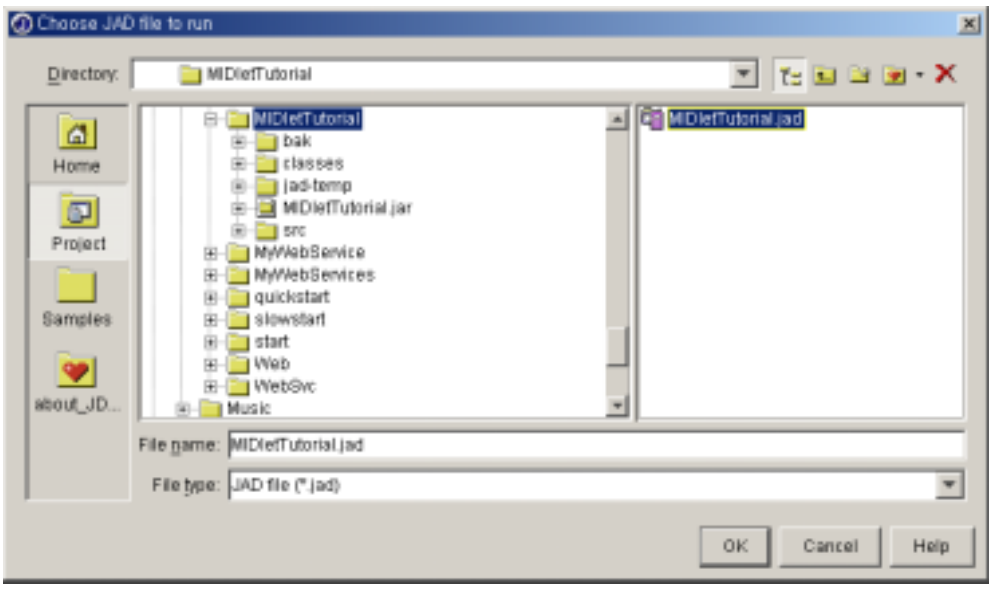
Specifying the default file to run

1. Project | Project Properties를 클릭하고 Run 탭을 선택합니다.
2. MIDlet1 런타임 컨피규레이션을 선택하고 Edit 버튼 클릭합니다.

3. JAD파일에 Radio 버튼을 클릭하고 JAD 파일 필드의 오른쪽에 있는  를 클릭합니다.



4.프로젝트 디렉토리를 검색하여 여러분이 방금 생성한 Archive된 JAD파일을 선택합니다.



5. OK를 눌러 파일 브라우저를 닫고, OK를 눌러 Project Properties 다이얼로그 박스를 닫습니다.

6. Run 버튼을 클릭합니다.

만약 여러분이 JBuilder7 SE 또는 JBuilder7 Enterprise Edition을 사용하고 있다면 JBuilder7 메뉴의 Run | Configuration 을 선택하여 좀더 세세한 부가적인 런타임 설정을 하실 수 있습니다. 이에 대한 자세한 내용은 Building Application with JBuilder의 “Setting Runtime configuration” 항목을 참고하시기 바랍니다.

JBuilder™ Mobileset 개발자 가이드

문서번호 : IMOBILE-JBM001

Appendix B

Tutorial: StopWatch MIDlet

개요

StopWatch 튜토리얼은 J2ME MIDP 어플리케이션 개발에 있어서 몇가지 기본적인 개념들을 소개합니다. 이는 Canvas 위에 그리기, 색 지정, 색으로 사각형을 채우기, 폰트 지정, 문자를 쓰는 것 등에 관한 방법을 알려줍니다. 또한 다수의 Displayable를 사용한 MIDlet의 생성과 하나에서 다른 Displayable로 어떻게 이동하는 지에 관한 내용을 보여줍니다. 마지막 파트에서는 RMS 데이터베이스로부터 정보를 저장하고 불러오는 것을 배우게 됩니다.

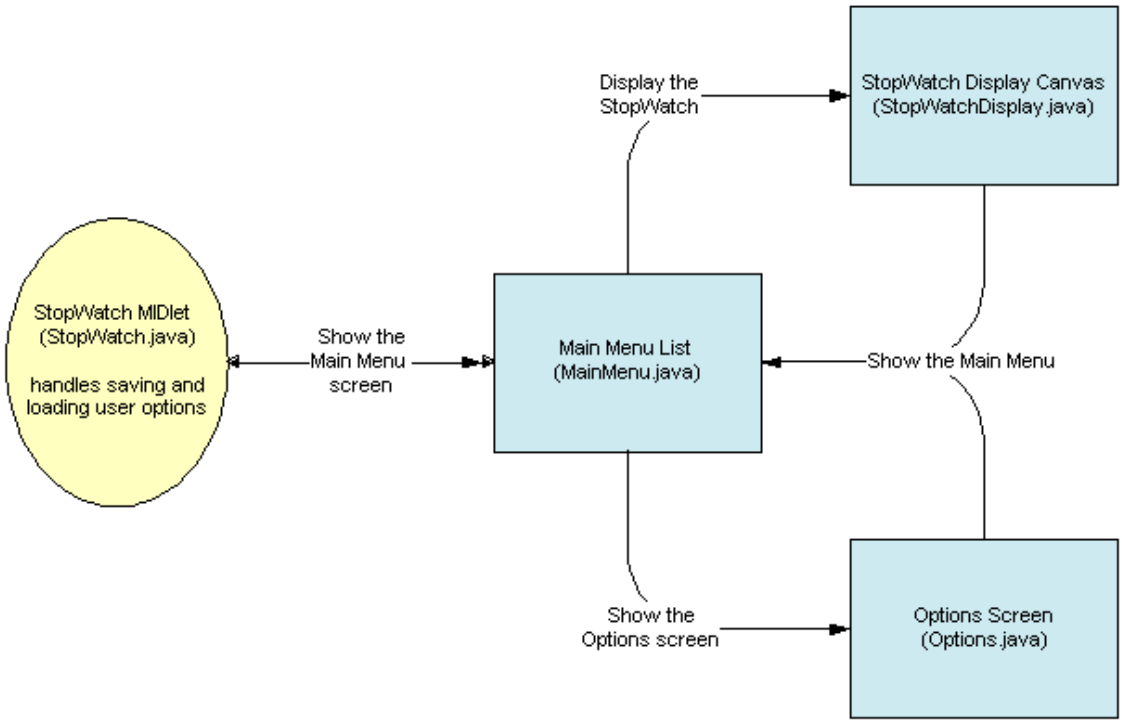
이 튜토리얼은 8개의 파트로 나누어 집니다.

- 파트 1은 StopWatch MIDlet 프로젝트와 MainMenu Displayable을 포함한 핵심 MIDlet 파일들을 만들기 위해 Project 와 MIDlet 마법사를 사용합니다.
- 파트 2는 기기에서 StopWatch를 나타내기 위한 MainMenu Displayble 내의 코드를 만들고, 메뉴 선택에 따른 결과를 다루게 됩니다.
- 파트 3은 MIDlet에 StopWatchDisply라 불리는 두번째 Displayble을 추가하고, 그 안에 기본적인 Stopwatch 기능들을 작성할 것입니다.
- 파트 4는 Canvas에 stopwatch 시각을 그리는 코드를 작성할 것입니다.
- 파트 5는 화면에 나타나는 시간의 형식을 지정하는 메소드를 생성합니다.
- 파트 6은 Options라 불리는 세번째 Displayable을 생성하고, stopwatch 디스플레이 옵션을 선택하고 Displayable과 Main Menu 간의 화면 전환에 필요한 코드를 작성하게 될 것입니다.
- 파트 7은 사용자가 Options 메뉴에서 선택한 시간 형식에 근거한 디스플레이 변경을 위한 코드를 작성합니다.
- 파트 8은 RMS 레코드 저장소에 stopwatch 옵션들을 저장하고, 레코드 저장소로부터 정보를 얻어내는 코드를 작성합니다.

StopWatch MIDlet는 다음 4개의 클래스들을 포함합니다.

1 StopWatch.java는 사용자 옵션의 저장과 로딩을 제어하는 MIDlet 주 클래스입니다.
2 MainMenu.java는 화면에 메뉴를 나타내기 위한 Displayable 클래스입니다.
3 StopWatchDisplay.java는 stopwatch를 위한 기본적인 기능들을 포함하는 Displayable 클래스입니다.
4 Options.java는 사용자에게 시간의 포맷 옵션 지정을 제공하는 Displayable 클래스입니다.

아래는 이 네가지 클래스 파일들의 기능과 상호작용을 보여주는 다이어그램입니다.



StopWatch MIDlet 완전한 작업 샘플은 JBuilder7\samples\Mobile\Stopwatch\ 디렉토리에서 볼 수 있습니다. MobileSet 샘플들을 실행하기 위한 내용을 보고 싶다면, 10-1 페이지의 "Running MIDP samples in JBuilder"을 참고하시기 바랍니다.

주의 각 에뮬레이터는 다르게 작동하기 때문에 MIDlet이 다양한 에뮬레이터에서는 다르게 보이거나 같게 작동하지 않을 수도 있습니다. 이 튜토리얼을 만들 때는 Sun의 J2ME Wireless Toolket을 사용하였고, 테스트를 위해서 DefaultColorPhone을 선택하였습니다. 사용하는 에뮬레이터의 작동 방법에 관한 내용을 알고 싶다면 관련 문서를 참조하시기 바랍니다.

이 튜토리얼에서 제공되는 모든 코드들은 어플리케이션 안에 선택적으로 포함할 수 있는 설명적인 주석에 의한 서두를 달고 있습니다. StopWatch 예제 안의 JavaDoc 주석을 보려면, JBuilder7 내의 StopWatch 프로젝트를 열고, 컨텐츠 창에서 열기 위해 .java 파일 중 하나를 더블 클릭합니다. 그 파일의 주석을 보려면, 파일의 하단에 있는 Doc 탭을 클릭합니다.

팁 튜토리얼의 각 파트가 끝날 때에는 모든 파일들을 저장하는 것이 좋습니다.

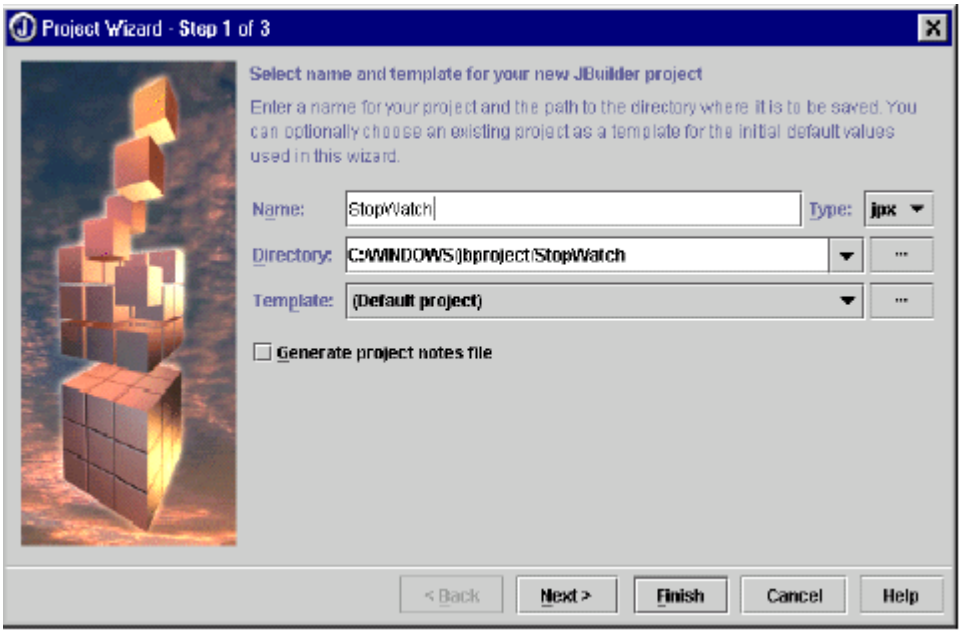
파트 1 : 프로젝트와 핵심 MIDlet 파일 생성

1단계 : MIDlet 프로젝트 생성

JBuilder에서 MIDlet을 생성하기 위한 첫번째 단계는 프로젝트를 생성해 주는 마법사를 사용하는 것입니다. JBuilder 프로젝트는 MIDlet에 필요한 모든 파일들을 포함합니다.

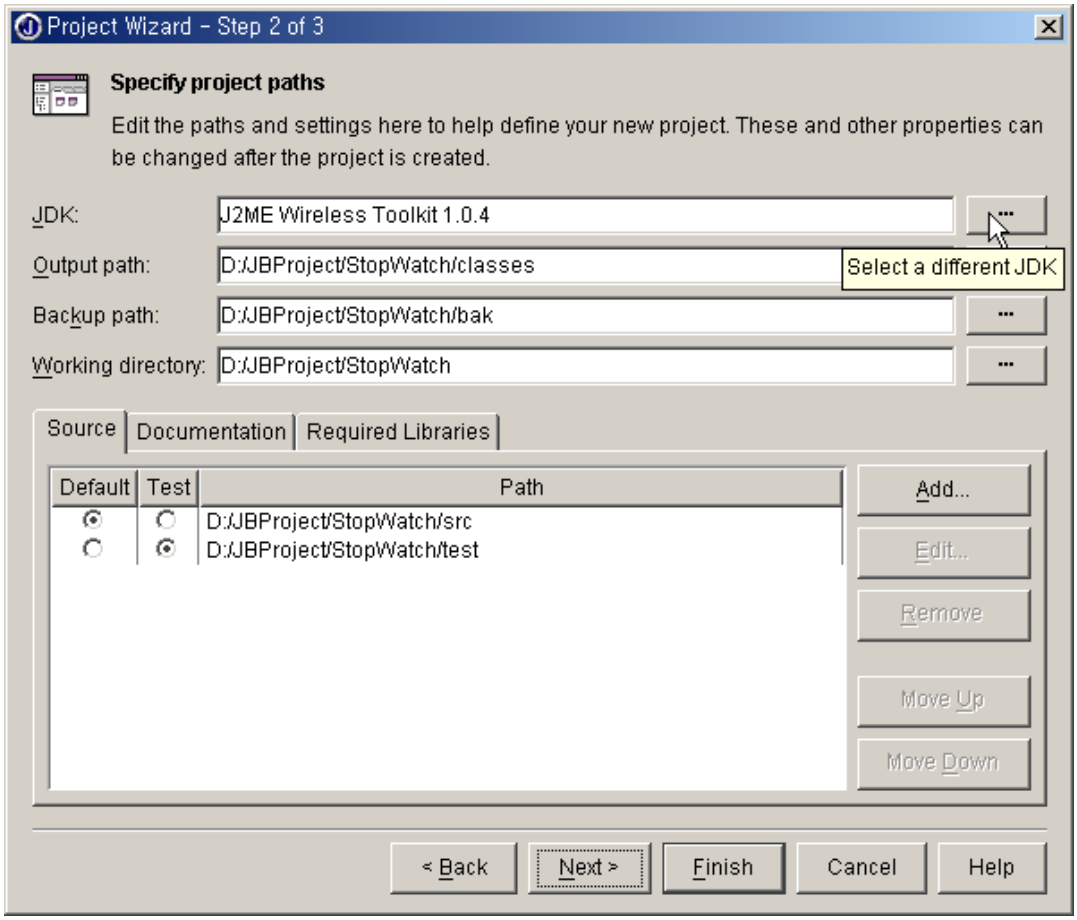
1 프로젝트 마법사를 열기 위해 File|New Project 를 선택

2 프로젝트 이름으로 StopWatch 입력



나머지는 기본 설정으로 남겨두고 Next를 클릭합니다.

3 JDK 필드 옆에 위치한 ellipse를 클릭하고, 이 MIDlet을 사용하기 위한 MIDP/CLDC JDK를 선택합니다.



나머지는 기본 설정으로 남겨두고 Next를 클릭합니다..

4 마지막 단계는 기본 설정으로 남겨두고 Finish를 클릭합니다.

프로젝트 마법사는 모든 기본적인 설정을 포함한 프로젝트 파일인 StopWatch.jpx 를 만들어줍니다. 이 파일은 AppBrowser의 좌측 상단의 Project 창 내부에 나타날 것입니다.

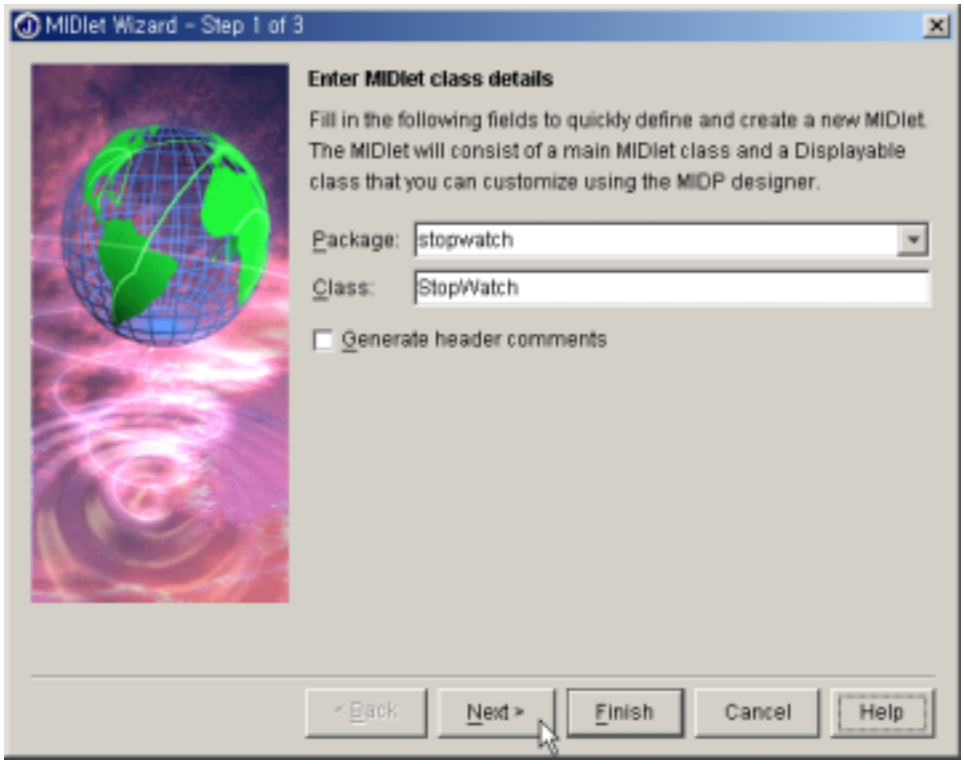
2단계: MIDlet 파일 생성

이제 프로젝트를 만들었으니, 프로젝트에 개략의 MIDlet 파일을 생성하기 위한 MIDlet 마법사를 사용할 수 있습니다.

1 File|New 를 선택하고 오브젝트 갤러리에서 Micro 탭을 선택합니다.

2 MIDlet 마법사 아이콘을 더블클릭합니다.

3 메인MIDlet 클래스(MIDlet1)의 기본 이름을 StopWatch로 하고 다음으로 넘어갑니다.

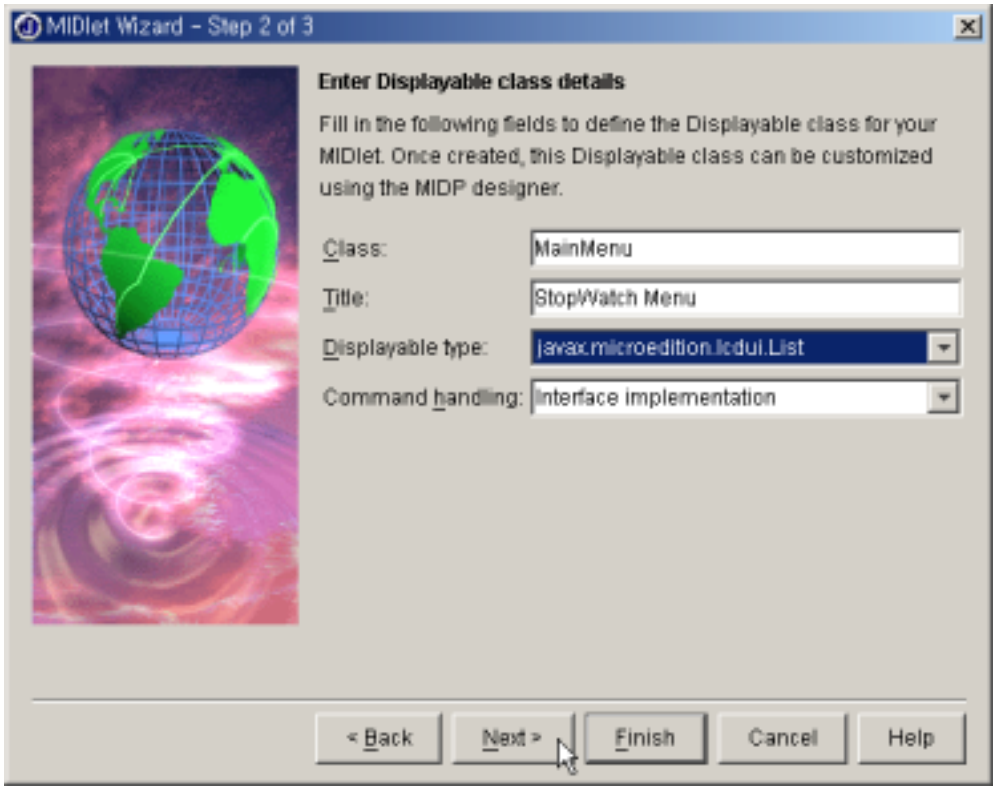


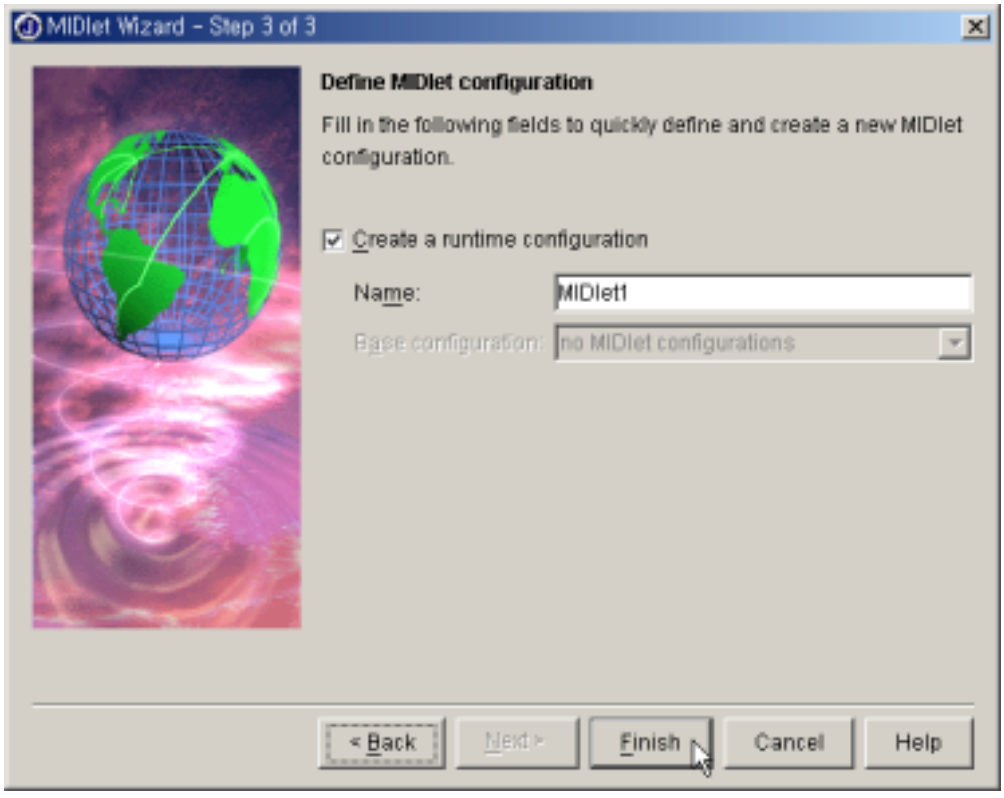
4 Displayable 클래스 이름을 MainMenu로, 제목을 StopWatch Menu로 대체합니다.

5 Displayable 타입 필드 옆의 화살표를 다운 클릭하여 javax.microedition.lcdui.List를 선택합니다.

6 기본 Command Handling 필드를 Interface Implementation으로 남겨둔다. 이 옵션은 클래스 내에 commandListener 인터페이스를 구현하는 코드를 생성할 것입니다. 이러한 방식으로 생성된 클래스의 크기는 adapter로 이루어진 것들보다 작을 것입니다.

7 Next 버튼을 누르고 위저드의 3단계에서는 Create a Runtime Configuration을 선택합니다. Finish 클릭.


MIDlet Wizard - Step 2 of 3
Enter Displayable class details
Fill in the following fields to define the Displayable class for your MIDlet. Once created, this Displayable class can be customized using the MIDP designer.
Class:
MainMenu
Title:
StopWatch Menu
Displayable type:
javax.microedition.lcdui.List
Command handling:
Interface implementation
< Back
Next >
Finish
Cancel
Help


MIDlet Wizard - Step 3 of 3
Define MIDlet configuration
Fill in the following fields to quickly define and create a new MIDlet configuration.
Create a runtime configuration
Name:
MIDlet1
Base configuration:
no MIDlet configurations
< Back
Next >
Finish
Cancel
Help

MIDlet 위저드는 StopWatch.java 라는 이름을 가진 메인 MIDlet 클래스와 MainMenu.java라 불리는 Displayable 클래스를 만듭니다. 두 파일 모두 자동적으로 프로젝트에 추가되며 아래 그림과 같이 프로젝트 창에 나타납니다.

MainMenu는 컨텐츠 창의 Source View로 열려있습니다.

파트2: MainMenu 기능 생성

MainMenu.java 파일은 javax.microedition.lcdui.List 를 확장하여 StopWatch MIDlet 을 위한 MainMenu를 생성합니다.

메뉴에는 두 가지 선택 항목이 생기게 됩니다.

- Run StopWatch : 메인 StopWatchDisplay 스크린에 로딩시킵니다.
- Options : 사용자가 시간 디스플레이 형식을 선택할 수 있도록 지정하는 Option 화면을 로딩시킵니다.

아래의 단계에서는 이러한 메뉴 선택 항목을 생성하고, 메뉴 선택 이벤트를 처리하는 내용을 다루게 됩니다.

1단계: MainMent 클래스 상수 추가

먼저 MIDlet 를 위한 선택 메뉴들을 정의합니다.

1 커서는 생성자(public class MainMenu extends List implements CommandListener {}의 첫번째 줄 끝의 왼쪽 괄호 다음에 위치시키고, 빈 몇 줄을 추가하기 위해 엔터 키를 여러 번 칩니다.

2 이 빈 공간에 다음의 클래스 상수들을 삽입합니다.

```
// These are our menu choices that will be in our List.
private static final String[] MENU_CHOICES = new String[2];
static {
   MENU_CHOICES[0] = "Run StopWatch";
   MENU_CHOICES[1] = "Options";
};
```

2단계: MainMenu 인스턴스 변수 추가

다음으로 StopWatchDisplay 스크린에 앞으로 튜토리얼에서 생성될 다음의 인스턴스 변수를 추가할 필요가 있습니다. 방금 생성한 클래스 상수 다음 줄에 커서를 위치시키고, 아래와 같은 인스턴스 변수를 입력합니다.

```
// StopWatch screen. (the screen that displays the actual stopwatch time).
private StopWatchDisplay stopWatchDisplay = new StopWatchDisplay(this);
```

주의 앞으로 튜토리얼에 다시 되돌아와 Options 스크린에 인스턴스 변수들을 추가할 것입니다.

3단계: 새로운 리스트 아이템을 사용하기 위한 super() 호출의 변경

이제, 클래스 상수내에서 정의한 리스트 아이템을 호출하기 위한 super() 메소드를 변경시켜보자. MainMenu 클래스 생성자 내에 다음의 super() 메소드를 위치시킵니다.

```
super("StopWatchMenu", List.IMPLICIT);
```

메뉴 선택을 추가하기 위해 내용을 변경시킵니다.

```
// Create a new List containing our defined menu choices. We will title
// our list "StopWatch Menu".
super("StopWatch Menu", // List title.
        List.IMPLICIT, // List type.
        MENU_CHOICES, // Our defined array of menu choices.
        null); // Images. (we're not including any images).
```

4단계: commandAction() 메소드에 이벤트 핸들링 추가

다음으로 리스트에서 사용자가 메뉴를 선택하여 사용할 때 생기는 이벤트 핸들링을 추가 시킬 필요가 있습니다.

MIDlet 마법사에 의해 생성된 jbInit() 메소드는 EXIT 명령이 포함되어 있고, commandAction() 메소드는 EXIT가 선택되었을 때 MIDlet을 종료시키는 이벤트 핸들링 코드를 포함합니다. 자 메뉴 이벤트 핸들링을 위해 commandAction() 메소드에 다음의 Exit 명령 이벤트 핸들러 코드를 추가합니다.

현재, actionCommand() 메소드는 다음과 같습니다.

```
/**Handle command events*/
public void commandAction(Command command, Displayable displayable) {
    /** @todo Add command handling code */
```

```
        if (command.getCommandType() == Command.EXIT) {
            // stop the MIDlet
            StopWatch.quitApp();
        }
    }
```

그것을 다음과 같이 바꿉니다. (각 코드의 기능을 설명한 주석이 포함되어 있음에 주의합니다. 만일 원합니다면 코드내의 그것들을 포함할 수도 있습니다.)

```
    /**
    * <p>Handle command events.</p>
    * @param command a Command object identifying the command
    * @param displayable the Displayable on which this event has occurred
    */
    public void commandAction(Command command, Displayable displayable) {
        // First, get the type of command that was just received.
        int commandType = command.getCommandType();
        // Now perform an action based on the type of command received.
        if (commandType == Command.EXIT) {
            // We just received the EXIT command (user just pressed EXIT), so quit
            // the MIDlet.
            StopWatch.quitApp();
        }
        else {
            // User must have selected one of the menu items. Find out what is
            // selected and display the appropriate screen.
            String selectedItem = getString(getSelectedIndex());
            if (selectedItem.equals(MENU_CHOICES[0])) {
                // Show the StopWatch screen.
                Display.getDisplay(StopWatch.instance).setCurrent(stopWatchDisplay);
            }
        }
    }
```

5단계: destroy() 메소드 생성

최종으로 MIDlet을 종료할 때 변수들을 정리하는 메소드를 추가해 봅니다. 다음의 메소드는 변수 레퍼런스를 null로 지정합니다. MainMenu 클래스의 끝(닫는 괄호 바로 위)에 커서를 위치하고, 다음의 destroy() 메소드를 추가합니다.

```
/**
* <p>Releases references. Used when quitting the MIDlet to make sure that
* all variable references are null.</p>
*/
void destroy() {
    stopWatchDisplay.destroy();
    stopWatchDisplay = null;
}
```

파트3: 기본 stopwatch 기능 생성하기

튜토리얼의 이번 파트에서는 StopWatchDisplay라 불리는 새로운 Displayable를 생성할 것인데, 이 클래스는 사용자가 Run StopWatch를 선택할 때 MENU_CHOICES[0]로서 MainMenu 내에서 연관, 지정되어 있습니다.

1단계: 새로운 Displayable 생성

StopWatchDisplay를 생성하기 위해 MIDP Displayable 마법사를 사용할 것이고, MIDlet 프로젝트에 그것을 추가할 것입니다.

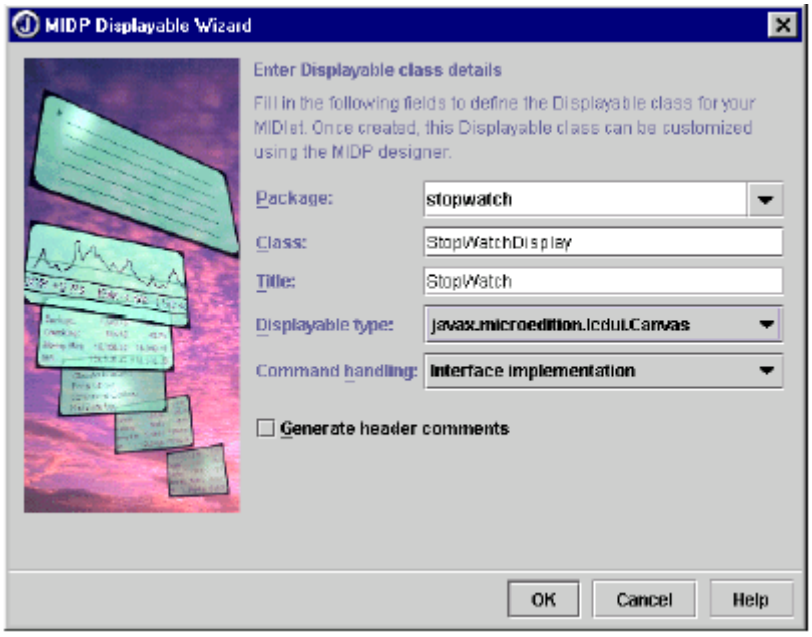
1 File|New를 선택하고 마법사를 실행시키기 위해 오브젝트 갤러리의 Micro 탭의 MIDP Displayable 아이콘을 더블 클릭합니다.

2 Displayable 클래스 이름을 StopWatchDisplay로 대체합니다.

3 Displayable 제목을 StopWatch로 대체합니다.

4 Displayable 타입 필드 옆의 화살표를 다운 클릭하여 javax.microedition.lcdui.Canvas를 선택합니다.

5 기본 Command Handling 필드를 Interface Implementation 으로 남겨둡니다.



6 끝났으면 OK 를 클릭합니다.

마법사는 StopWatchDisplay.java 라는 새로운 클래스를 생성하고, 이는 프로젝트에 추가되며 프로젝트 창에 나타납니다. StopWatchDisplay는 컨텐츠 창에 편집을 위해 선택되어 열려있습니다.

2단계 : 추가 클래스 임포트하기

기본 stopwatch 기능을 위해 java.util.* 패키지로부터 두 개의 추가되는 클래스들을 임포트할 필요가 있습니다. StopWtatchDisplayable 클래스 맨 위의 import 부분에 다음 줄들을 추가시킵니다.

```
import java.util.Timer;
import java.util.TimerTask;
```

3단계: 인스턴스 변수 생성

기본적인 stopwatch 기능을 코딩하기 전에, stopwatch를 시작하고, 실행하고, 중지시킬 몇 개의 클래스 레벨의 인스턴스를 생성할 필요가 있습니다.

1 StopWatchDisplay 정의 맨 윗 부분에 몇 개의 빈줄을 만들고 아래 보이는 것과 같은 인스턴스 변수들을 삽입합니다.(모든 코드가 시작되기 전에는 그 코드의 기능을 설명하는 주석을 둘 수 있다는 것을 알아두시기 바랍니다.)

```
public class StopWatchDisplay extends Canvas implements CommandListener {
// Timer for running the stopwatch task
```

```
private Timer timer;
// Stopwatch running task
private RunStopWatch runStopWatchTask;
// Stopwatch start time
private long startTime = 0;
// Canvas width
private int width = getWidth();
// Canvas height
private int height = getHeight();
// Font for drawing the stopwatch time
private Font font = Font.getDefaultFont();
// Command for starting the stopwatch
private Command startCommand = new Command("Start", Command.SCREEN, 1);
// Command for stopping the stopwatch
private Command stopCommand = new Command("Stop", Command.SCREEN, 1);
// Main menu.
private MainMenu mainMenu;
```

4단계: 생성자 변경하기

StopWatchDisplay() 생성자에 만들어 질 필요가 있는 두 개의 변경 사항이 있습니다.

- 인자로서 MainMenu 를 등록시킨다.
- 생성자 내의 인자를 저장합니다..

이러한 변화는 아래 보이는 바와 같습니다.

```
/**
* Constructs a new StopWatchDisplay canvas for drawing the stopwatch,
* starting and stopping the stopwatch, and provides an option to go back
* to the given MainMenu instance.
*
* @param mainMenu parent MainMenu screen instance
*/
public StopWatchDisplay(MainMenu mainMenu) {
    // Save the MainMenu instance so that we can access it later when we
    // want to switch back to the MainMenu screen.
    this.mainMenu = mainMenu;
```

```
        // Initialize the canvas.
        try {
            jbInit();
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
```

5단계: stopwatch 명령의 초기화

이제 startCommand를 이미 생성하였으므로, jbInit() 메소드 내에 이를 위한 초기화 코드를 생성할 필요가 있습니다. 또한 캔버스에 그리기 위한 폰트도 초기화할 필요가 있습니다.

1 jbInit() 메소드 내의 왼쪽 괄호 다음에 빈줄을 만든다.

```
private void jbInit() throws Exception {
```

2 jbInit() 메소드 안에 자동으로 생성된 코드를 다음의 addCommand(startCommand)와 initializeFont() 메소드로 대체시킨다. 모두 수행하였을 때, jbInit() 메소드를 다음과 같을 것입니다.

```
/**
* Component initialization. Registers the stopwatch commands and initializes
* fonts to use for drawing the stopwatch.
*/
private void jbInit() throws Exception {
    // Add the "Start" command for starting the stopwatch. Later, after
    // the user hits "start", we will swap this command for the "Stop"
    // command.
    addCommand(startCommand);
    // Initialize the font that we will use to draw the stopwatch time
    // onto the canvas.
    initializeFont();
    // Set up this Displayable to listen to command events.
    setCommandListener(this);
    // Add the command for exiting the MIDlet.
    addCommand(new Command("Back", Command.BACK, 1));
}
```

이 튜토리얼의 후반부에서 initalizeFont() 메소드를 생성할 것입니다.

6단계: stopwatch 명령 제어하기

자 stopwatch 명령을 제어하기 위한 코드를 작성할 준비가 되었습니다. 이 과정은 다음의 작업에 따라 처리됩니다.

1 발생한 명령을 얻습니다.

2 명령이 SCREEN 명령이며, 그것이 stopwatch를 시작하는 지 아니면 종료시키는 지를 파악합니다.

1 만일 stopwatch를 시작시키는 명령이며, 새로운 RunStopWatch 작업을 생성하고 실행시킴으로써 stopwatch를 시작합니다.

2 만일 stopwatch 명령을 중지하는 것이라면, RunStopWatch 작업을 중지시킴으로써 stopwatch를 끝냅니다.

3 만일 명령이 BACK 명령이며, 사용자는 MainMenu displayable 로 돌아가도록 요청됩니다.

주의 위 리스트의 작업을 클릭하면, 작업을 수행하는 아래의 코드가 작성되어야 할 것입니다. Public void commandAction(Command c, Displayable d)를 찾습니다. 아래와 같이 왼쪽 괄호와 오른쪽 괄호 사이의 내용들을 삭제합니다.

```
/**
* Handle command events.
*
* @param command a Command object identifying the command
* @param displayable the Displayable on which this event has occurred
*/
public void commandAction(Command c, Displayable d) {
   |
}
```

이 commandAction() 메소드의 몸체에 연속적으로 다음의 코드를 추가합니다.

1 발생된 명령을 얻는다.

```
// Get the command type
int commandType = command.getCommandType();
```

2 명령이 SCREEN 명령이며, stopwatch를 시작하는 명령인지 끝내는 명령인지를 파악합니다.

```
// Check if the command received was one that we defined (as opposed
// to an EXIT command or a MIDlet STOP command.
if (commandType == Command.SCREEN) {
// Get the string label associated with the command.
```

```
String commandName = command.getLabel();
```

1 stopwatch를 시작하는 명령이며, 새로운 RunStopWatch 작업을 생성하고 실행함으로써 stopwatch를 시작합니다.

```
// Now check if the string label matches either our Start or Stop
// commands.
if (commandName.equals("Start")) {
    // Someone has pressed the Start button, so...
    // Get our current time.
    startTime = System.currentTimeMillis();
    // Create a new Timer to run the stopwatch.
    timer = new Timer();
    // Create a new RunStopWatch task to give to the timer.
    runStopWatchTask = new RunStopWatch();
    // Ask the Timer to run the stopwatch.
    // The task to run.
    timer.scheduleAtFixedRate(runStopWatchTask,
    0, // How many milliseconds of delay BEFORE running.
    10); // Time in milliseconds between successive task executions,
    // (i.e. run it every 10 milliseconds).
    // Now, swap the Start command for the Stop command so that the
    // user can stop the stopwatch!
    removeCommand(startCommand);
    addCommand(stopCommand);
}
```

2 만일 중지시키는 명령이면, RunStopWatch 작업을 중단함으로써 stopwatch를 중지시캅니다.

```
else if (commandName.equals("Stop")) {
    // Someone has pressed the Stop button.
    timer.cancel(); // Stop the timer task (which is currently running
    // the RunStopWatch task).
    timer = null; // Set it explicitly to null to make sure that the
    // timer has been reset.
    // Swap the Stop command for the Start command so that the user can
    // start the stopwatch again.
    removeCommand(stopCommand);
    addCommand(startCommand);
```

```
        }
    }
    3 만일 BACK 명령이면, 사용자는 MainMenu displayable을 보도록 요구됩니다.
    else if (commandType == Command.BACK) {
        // Go back to the Main Menu screen.
        Display.getDisplay(StopWatch.instance).setCurrent(mainMenu);
    }
```

7단계: destroy() 메소드 생성

MainMenu 클래스에서 했던 바와 같이, MIDlet 를 종료할 때 참조되었던 변수들을 정리하기 위한 destroy() 메소드를 추가합니다.

StopWatchDisplay 클래스의 끝에 커서를 위치시키고(닫는 괄호 바로 위에), 다음의 destroy() 메소드를 추가합니다.

```
/**
* <p>Releases references. Used when quitting the MIDlet.</p>
*/
void destroy() {
    timer = null;
    runStopWatchTask = null;
    startCommand = null;
    stopCommand = null;
}
```

8단계: TimerTask 생성

이 단계에서는 시간을 갱신하기 위한 내부 클래스를 생성합니다. Stopwatch가 시작되었을 때 타이머에 작업을 정의합니다. 이 작업은 stopwatch를 실행하고 지정된 간격으로 화면에 시간을 다시 그려줍니다.

괄호를 닫기 전에 StopWatchDisplay 클래스의 끝에 커서를 위치시키고, 다음의 내부 클래스를 만든다.

```
/**
* This inner class is the task to give the timer when the user starts the
* stopwatch. This task will run at the interval specified by the timer.
* (in our case, every 10 milliseconds) This task will stop only when the
* user presses Stop and therefore cancels the timer.
*/
class RunStopWatch extends TimerTask {
```

```
        public void run() {
            // Repaint() automatically calls paint().
            StopWatchDisplay.this.repaint();
        }
    }
```

이것은 stopwatch를 위한 기본 기능을 완성시킨다. 다음 파트에 가서는 캔버스에 시간을 어떻게 그리는 지에 관한 내용에 관해 알아볼 것입니다.

파트4: 캔버스에 시간 그리기

튜토리얼의 이번 파트에서는 캔버스 위에 그리는 방법을 배울 것입니다. 이것은 다음의 작업들과 연관되어 있습니다.

1 캔버스를 지운다.

2 펜 색깔과 폰트를 설정합니다.

3 stopwatch위에 현재 시간을 그린다.

Paint() 메소드는 Canvas를 위해 필요하며, Displayable 타입으로서 Canvas를 선택하였다면 MIDlet 또는 Displayable 마법사에 의해 자동으로 생성될 것입니다.

```
    /** Required paint implementation */
    protected void paint(Graphics g) {
    /** @todo Add paint codes */
    }
```

화면에 시간을 그릴 paint() 메소드 내에 기능을 추가할 필요가 있습니다.

주의 다음 단계의 모든 코드 부분들은 paint() 메소드에 연속적으로 추가됩니다.

1단계: 캔버스 지우기

첫번째 단계는 디스플레이 화면(Canvas)을 지우는 것입니다. 이것은 현재의 펜 색깔을 흰색으로 설정하고, 현재의 펜 색깔로 캔버스 전체를 채움으로써 이루어집니다.

```
// "Clear" the Canvas by painting the background white (you may choose
// another color if you like.
// Set the current pen color to white (red=255, green=255, blue=255).
g.setColor(255, 255, 255);
// Fill the entire canvas area with the current pen color.
g.fillRect(0, 0, width, height);
```

2단계: 현재 시간 알아내기

다음으로 그릴 현재의 시간을 알아낼 필요가 있습니다. 이를 위해 다음 코드를 추가하자.

```
// Find out what is the current stopwatch time. The stopwatch time is
// the current time MINUS the startTime that was recorded when the
// user pressed the Start button. If the startTime is 0, then the
// current stopwatch time is 0 as well.
long elapsed = (startTime == 0)
? 0
: System.currentTimeMillis() - startTime;
```

3단계: 펜 색깔 변경

흰색에서 검정색까지 팬 색깔을 변경시키는 다음의 코드를 추가합니다.

```
// Set the pen to black (it's currently white).
g.setColor(0);
```

4단계: 폰트 지정

시간을 그리는 폰트를 설정하기 위한 다음의 코드를 추가합니다.

```
// Set a font to use for drawing the time. (see the method initializeFont())
g.setFont(font);
```

5단계: 현재 시간 그리기

이제 캔버스의 xy 좌표를 사용하여 현재의 stopwatch 시간을 그릴 다음의 코드를 추가하자.

```
// Dynamically compute the best position to draw the stopwatch string given
// the width and height of your canvas.
// For instance, to center the text on ANY canvas:
// x position = (canvaswidth / 2) - (stopwatchstringlength / 2)
// y position = (canvasheight / 2) - (stopwatchstringheight / 2)
// Formats the current time into MM:SS:T format.
String formattedTime = formatTime(elapsed);
// Compute the start point of our text such that it will be centered
```

```
// on our canvas.
int x = (width / 2) - (font.stringWidth(formattedTime) / 2);
int y = (height / 2) - (font.getHeight() / 2);
// Now draw the string at the (x, y) anchor point that we just computed.
// Specify that we want the anchor point to be the top left corner of our
// text.
g.drawString(formattedTime, x, y, Graphics.TOP | Graphics.LEFT);
```

파트 5: 시간의 포맷 지정

formatTime() 메소드를 구성하는 코드의 마지막 부분에 주의하자. 자바에서의 시간은 밀리초의 증가로 인식되기 때문에, 이 밀리초를 stopwatch에 나타내기 위한 하나 이상의 관습에 따른 mm:ss:t(분:초:tenths of seconds)로 전환할 필요가 있습니다.

1단계: formatTime() 메소드 생성하기

paint() 메소드 뒤에 다음의 메소드를 생성하고, 분, 초, tenths of seconds를 추가합니다.

```
/**
* <p>Formats the given number of milliseconds into minute/seconds/tenths
* display. For instance, 2346 milliseconds will translate to 00:02:4.</p>
*
* @param milliseconds number of milliseconds
* @return string in the form of "MM:SS:T"
*/
private String formatTime(long milliseconds) {
    long minutes = 0; // Start with 0 minutes.
    long seconds = milliseconds / 1000; // Get the number of seconds.
    long tenths = (milliseconds / 100) % 10; // Number of tenths of seconds.
}
```

2단계: 얼마나 많은 초가 있는지 체크

다음으로 얼마나 많은 초가 있는지를 체크하기 위해 formatTime() 메소드 내에 코드를 추가합니다. 만일 60이상이라면, 분으로 나누어질 것입니다.

```
// Check how many seconds we have.
if (seconds >= 60) {
    // If we have more than 60 seconds, we can break it down into minutes.
    minutes = seconds / 60; // Get the number of minutes.
```

```
        seconds = seconds % 60; // Get the number of remainder seconds.
    }
```

3단계: 분, 초에 해당하는 문자 생성

다음과 같이 minutes String 과 seconds String을 추가합니다.

```
// Create our "minutes" string.
String minuteString = String.valueOf(minutes);
if (minutes > 10) {
    // We have less than 10 minutes, so prepend a "0" to make sure.
    // Our minutes string has 2 digits.
    minuteString = "0" + minuteString;
}
// Create our "seconds" string
String secondsString = String.valueOf(seconds);
if (seconds > 10) {
    // We have less than 10 seconds, so prepend a "0" to make.
    // Sure our seconds string has 2 digits.
    secondsString = "0" + secondsString;
}
```

만일 분이나 초가 10 미만이라면, 자리 앞에 0을 추가하여 항상 두자리 수를 만듭니다.

4단계: 최종 시간 생성

분, 초, tenths of seconds를 함께 놓을 코드를 추가하고 다음처럼 반환 String을 생성합니다.

```
//Put together and return a String of minutes, seconds, and tenths of
//seconds, each separated by a ":".
return minuteString + ":" + secondsString + ":" + String.valueOf(tenths);
```

5단계: 최대 가능한 크기로 폰트를 초기화하기

최종적으로, 코드내에서 이미 호출하였던 initializeFont() 메소드를 생성할 필요가 있습니다. 다양한 기기는 각기 다른 크기의 화면으로 구성되어 있기 때문에, 폰트를 초기화 하기에 앞서 어떠한 크기의 폰트를 사용할지를 결정해야 합니다. 화면에 맞도록 시간을 나타내기 위해 최대의 크기를 사용하기 원합니다면, 그것이 어떤 크기인지를 알도록 테스트할 필요가 있습니다. InitializeFont() 메소드는 테스트한 바와 같이 수행될 것입니다. 이는 다음의 작업을 포함합니다.

• 디스플레이 영역에 맞는 폰트를 테스트하기 위해 사용될 String.을 생성합니다.
• 기본 폰트로 화면에 얼마나 많은 테스트 문자가 적당한 지를 알아본다.
• 테스트 결과를 제어하기 위해 if()/else if() 문장을 생성합니다: 만일 기본 폰트 크기의 두 개의 테스트 문자가 화면에 적당하다면, 한 사이즈 크게 설정될 것입니다. 만일 기본 크기의 한 테스트 문자가 적당하지 않다면, 폰트를 한 사이즈 줄어들게 설정됩니다.

커서를 formatTime() 메소드 뒤에 위치시키고 initializeFont() 메소드를 위해 다음의 코드를 입력합니다.

```
/**
 * Initialize the font to the largest possible that can fit the display area.
 */
private void initializeFont() {
    // Test the font with this string to see if the string
    // will fit on the canvas.
    String test = "00:00:0";
    // See how many of the test strings we can fit on the screen with
    // the default font.
    int numStringsForThisWidth = width / font.stringWidth(test);
    if (width / numStringsForThisWidth > 2) {
        // More than 2 strings can fit across our canvas using the current font.
        // Set the font to one size bigger.
        font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
        Font.SIZE_LARGE);
    }
    else if (numStringsForThisWidth == 0) {
        // Our test string does NOT fit on the canvas with the current font. Set
        // the font to one size smaller.
        font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
        Font.SIZE_SMALL);
    }
}
```

6단계: MIDlet 컴파일과 실행

이 시점에서는 StopWatchMIDlet 을 컴파일하고 실행시킬 수 있습니다.

1 프로젝트를 저장합니다. (File|Save All를 선택, Save All 버튼을 클릭합니다.) 

2 Project|Make Project "StopWatch.jpx 를 선택하거나 Make  버튼을 클릭합니다.

만일 컴파일러가 어떤 에러를 잡아낸다면 에디터의 하단에 있는 메시지 판이 열리고 에러들을 보여줄 것입니다. 에러를 수정하고 성공할 때 까지 재컴파일합니다.

3 Run|Run project를 선택하거나 run  버튼을 클릭합니다.

JBUILDER는 StopWatch라 불리는 런타임 메시지 탭을 열고 MIDlet을 실행하기 위한 에뮬레이터를 구동합니다.

주의 MIDlet은 각 에뮬레이터마다 다르게 보이며 동작할 것입니다. 이 튜토리얼을 만들 때는 Sun J2ME Wireless Toolket의 DefaultColorPhone 에뮬레이터를 사용하였다. 만일 사용하고 잇는 에뮬레이터 작동에 관한 내용을 알고 싶다면 관련 문서를 참조하라.

StopWatch MIDlet의 수행이 끝나면, MIDlet을 종료하고 에뮬레이터를 닫습니다. JBUILDER로 돌아가서 StopWatch 탭의 메시지 판을 마우스 오른쪽 클릭하고 "StopWatch" 탭을 제거합니다.

이제 StopWatch MIDlet에 옵션을 추가할 준비가 되었다.

파트 6: StopWatch 옵션 생성

이번 파트에서는 시간이 증가하면서 stopwatch 시간을 어떠한 방법으로 나타낼 지를 사용자가 선택하는 Option 화면을 추가할 것입니다. 또한 선택한 옵션을 저장하고 로딩을 지원하는 RMS를 추가할 것입니다. 옵션의 선택에 따라 시간의 포맷이 변화합니다.

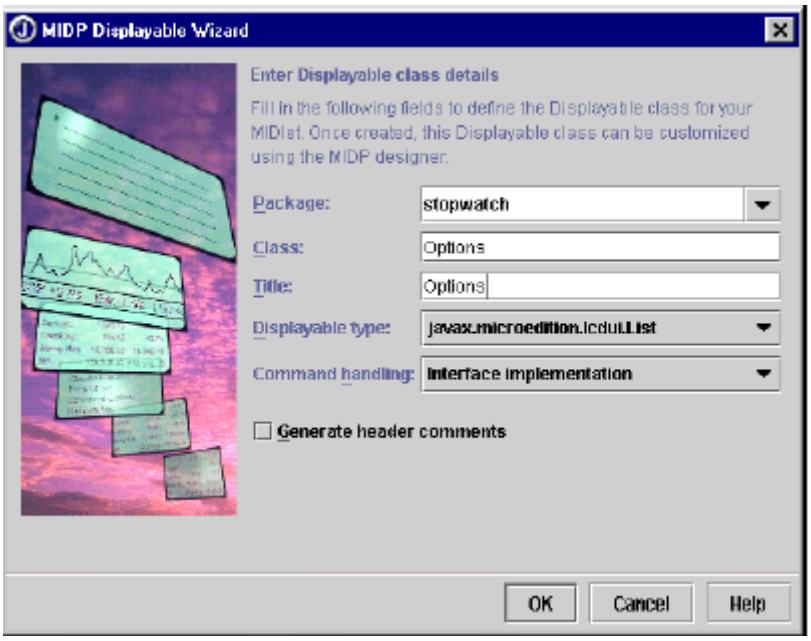
1단계: Options Displayable 생성

다시 한 번, 프로젝트에 MIDP Displayable 마법사를 사용하여 새로운 Displayable 을 추가할 것입니다.

1 File|New 를 선택하고 다시 한 번 MIDP Displayable 마법사를 실행시킨다.

2 마법사에서 다음 사항을 변경시키고 종료하기 위해 OK를 누른다.

- 클래스와 제목 이름을 "Options"로 합니다.
- javax.microedition.lcdui.List 를 Displayable 타입으로 선택합니다.



2단계: 클래스 상수 추가

디스플레이서 형식과 문자 표현을 위한 클래스 정의 맨 윗 부분에 다음의 클래스 상수들을 생성합니다.

```
// Constants for defining our time display format - either display
// in seconds or display in tenths of seconds.
public static final int FORMAT_TENTHS = 0;
public static final int FORMAT_SECONDS = 1;
// An array that contains string descriptions of the display formats.
private static final String[] FORMAT_DESCRIPTIONS = new String[2];
static {
    FORMAT_DESCRIPTIONS[FORMAT_TENTHS] = "Tenths of seconds (00:00:1)";
    FORMAT_DESCRIPTIONS[FORMAT_SECONDS] = "Seconds (00:01)";
};
```

3단계: 인스턴스 변수 생성

클래스 상수 뒤에 다음의 인스턴스 변수들을 추가합니다.

```
private MainMenu parent; // Main StopWatch menu screen.
private int displayFormat = FORMAT_TENTHS; // Initialized to default format.
```

4단계: 생성자 변경

생성자에 변경을 가할 필요가 있습니다.

• Options 생성자의 인자로서 MainMenu를 추가합니다.

• 클래스 상수에 새로운 리스트 아이템을 사용하기 위한 super() 호출을 변경합니다.

• 사용자가 Option 화면에서의 작업을 마쳤을 때, 부모(parent) 화면으로 되돌아갈 수 있도록 저장합니다.

• 명령 이벤트를 잡아낼 수 있도록 Options Displayable 을 설정합니다. 아래는 변경된 후의 생성자이다. 변경하고자 합니다면 이 코드 샘플을 사용하라. 코드내의 주석은 각각의 변경 사항을 정의하고 있습니다.

```
/**
* <p>Construct the displayable</p>
*/
public Options(MainMenu parent) {
// Construct a new exclusive List titled "Options" and containing the
// items in the FORMAT_DESCRIPTIONS array.
super("Options", // List title.
        List.EXCLUSIVE, // List type. User can only select 1 item.
        FORMAT_DESCRIPTIONS, // Our defined array of format descriptions.
        null); // Images. (we're not including any images).
// Save the parent so that we can go back to that screen when the user is
// finished with the Options screen.
this.parent = parent;
try {
    jbInit();
} catch(Exception e) {
    e.printStackTrace();
}
}
```

5단계: jbInit()에 OK와 Cancel 추가

Ok와 취소 명령을 초기화하기 위한 jbInit() 메소드의 몸체 코드에 더합니다. 다음과 같은 부분을

```
// add the Exit command
addCommand(new Command)"Exit", Command.EXIT, 1));
```

이와 같이 대체합니다.

```
// Add commands for OK (when user selects an option) and Cancel.
addCommand(new Command("OK", Command.OK, 1));
addCommand(new Command("Cancel", Command.CANCEL, 1));
```

수행하고 난 다음의 jbInit() 메소드는 다음과 같습니다.

```
private void jbInit() throws Exception {
    // set up this Displayable to listen to command events
    setCommandListener(this);
    // Add commands for OK (when user selects an option) and Cancel.
    addCommand(new Command("OK", Command.OK, 1));
    addCommand(new Command("Cancel", Command.CANCEL, 1));
}
```

6단계: OK와 Cancel 명령 제어

비어있는 CommandAction() 메소드 내에 커서를 위치시키고, OK 와 Cancel을 위한 이벤트 제어 코드를 추가합니다.

```
/**
* <p>Handle command events.</p>
* @param command a Command object identifying the command
* @param displayable the Displayable on which this event has occurred
*/
public void commandAction(Command command, Displayable displayable) {
    if (command.getCommandType() == Command.OK) {
        // User pressed ok, so save the user's selection.
        displayFormat = getSelectedIndex();
    }
    else if (command.getCommandType() == Command.CANCEL) {
        // Reset the display to the original selected value.
        setSelectedIndex(displayFormat, true);
    }
    // Now, set the display back to the main menu.
    Display.getDisplay(StopWatch.instance).setCurrent(parent);
}
```

7단계: 시간 포맷을 위한 getter 와 setter 메소드 생성

이제 시간의 형식을 지정하는 방법에 관한 내용으로, 선택된 시간 표현 방법을 얻어오고 지정하며 MIDlet 내의 다른 클래스들이 사용자가 행한 선택에 접근할 수 있도록 하는 두 개의 메소드가 필요하다. 이 두개의 메소드는 아래와 같고, 괄호를 닫기 바로 전에 클래스의 끝에 추가됩니다.

```
/**
* <p>Retrieves the selected time display format.</p>
```

```
* @return Either FORMAT_TENTHS or FORMAT_SECONDS
*/
public int getDisplayFormat() {
    return displayFormat;
}
/**
* <p>Set the selected display format.</p>
* @param format display format - either FORMAT_TENTHS or FORMAT_SECONDS
*/
public void setDisplayFormat(int format) {
    displayFormat = format; // Save the new display format.
    setSelectedIndex(format, true); // Set it as the currently selected format.
}
```

프로젝트를 저장합니다.

8단계: MainMenu 에서 Options을 지원

이 모든 것을 함께 묶기 위해서, MainMenu 클래스 안에 Options 클래스를 지원하는 코드를 추가할 필요가 있습니다.

1 에디터에서 열기 위해 MainMenu를 더블 클릭합니다.

2 MainMenu 클래스 StopWatchDisplay 변수 바로 다음에 아래의 인스턴스 변수를 추가합니다.

```
// Options screen to allow user to set preferences.
private Options optionsScreen = new Options(this);
```

3 MainMenu 클래스 comandAction() 메소드 바로 다음에 geiDisplayFormat()와 setDisplayFormat() 메소드를 생성합니다. 이 메소드들은 디스플레이 형식을 얻고 지정하기 위한 Option 화면으로의 연결을 담당합니다.

```
/**
* <p>Retrieves the selected display format from the options screen.</p>
* @return time display format
*/
public int getDisplayFormat() {
    return optionsScreen.getDisplayFormat();
}
/**
* <p>Sets the selected display format in the options screen.</p>
```

```
* @param format time display format
*/
public void setDisplayFormat(int format) {
    optionsScreen.setDisplayFormat(format);
}
```

4 아래 보이는 것처럼 Options 화면을 null로 만드는 destroy() 메소드를 MainMenu 클래스에 추가합니다.

```
void destroy() {
    optionsScreen = null;
    stopWatchDisplay.destroy();
    stopWatchDisplay = null;
}
```

5 commandAction 메소드의 끝에 Options 화면을 다룰 수 있는 다음의 코드를 추가합니다.

```
else if (selectedItem.equals(MENU_CHOICES[1])) {
    // Show the Options screen.
    Display.getDisplay(StopWatch.instance).setCurrent(optionsScreen);
}
```

6 프로젝트를 저장합니다. 이제 실행하여 보고, Options 화면이 동작하는 것을 살펴본다.

파트 7: option에 기반한 시간 표현 방법의 변경

사용자가 시간 형식 화면 옵션을 선택하게 되면, 그 옵션을 사용하기 위해 canvas에서 디스플레이를 변경할 필요가 있습니다.

1 StopWatchDisplay 소스코드로 돌아가서, formatTime() 메소드안에서 다음의 코드를 찾아라.

```
//Put together and return a String of minutes, seconds, and tenths of
//seconds, each separated by a ":".
return minuteString + ":" secondsString + ":" String.valueOf(tenths);
```

2 다음과 같은 새로운 코드로 대체합니다.

```
// Put together and return a string of minutes, seconds, and tenths of
// seconds, each separated by a ":".
String resultString = minuteString + ":" + secondsString;
```

```
if (mainMenu.getDisplayFormat() == Options.FORMAT_TENTHS) {
    // User has indicated that the tenths of seconds should be displayed.
    // Add it to our result string.
    resultString += ":" + String.valueOf(tenths);
}
return resultString;
```

파트 8 : RMS 레코드 저장소로부터 옵션을 읽고/쓰기

이 튜토리얼의 마지막 부분은 Record Management System(RMS) 레코드 저장소로 정보를 저장하도록 하고, 그로부터 다시 정보를 읽어와서 MIDlet에 사용하도록 할 것입니다. 저장되고 불러올 정보는 사용자에 의하여 선택된 시간 표시 옵션입니다.

저장소를 관리하고 그로부터 정보를 불러오기 위한 코드는 메인 StopWatch.java 클래스에 추가되어야 합니다.

1단계 : RMS 패키지 임포트하기

1 에디터에서 열기 위해 StopWatch.java를 더블클릭합니다.

2 클래스의 맨 위의 임포트 부분에 다음의 임포트 구문을 위치시킨다.

```
import javax.microedition.rms.*;
```

2단계 : 데이터베이스 이름 정의하기

데이터베이스로 어떤 사용자 옵션을 저장하기 전에, 먼저 데이터베이스 이름을 지정해 주어야 합니다. 이를 위해서는 먼저 다음과 같이 PREFERENCES_DATABASE 라 불리우는 클래스 상수를 생성합니다.

StopWatch 클래스 정의 맨 윗 부분에 다음의 클래스 상수를 삽입합니다.

```
// preferences database name
private static final String PREFERENCES_DATABASE = "Preferences";
```

3단계 : 디스플레이 변수 이름 변경하기

“mainMenu”와 같이 “displayable”보다 좀 더 의미있는 이름으로 MainMenu 클래스의 변수 이름들을 변경해 보자. StopWatch.java 내의 다음 인스턴스 변수를 찾도록 합니다.

```
private MainMenu displayable = new MainMenu();
```

을 다음과 같이 변경합니다.

```
private MainMenu mainMenu = new MainMenu();
```

또한 startApp() 메소드 내에서, 새로운 이름을 사용하기 위해 Display.getDisplay(this).setCurrent(displayable); 줄에서 “displayable” 변수 이름을 다음처럼 변경할 필요가 있습니다.

```
/** Main method */
public void startApp() {
    Display.getDisplay(this).setCurrent(mainMenu);
}
```

4단계 : loadOptions() 메소드 생성하기

이제는 미리 저장된 사용자 옵션을 로딩할 loadOptions()라 불리우는 새로운 메소드를 생성하자. StopWatch 클래스의 끝에 커서를 위치시키고, 닫는 괄호 앞에 다음의 코드를 작성합니다. 각 라인의 주석은 각 코드를 설명합니다.

```
  /**
  * <p>Retrieves the user preferences stored off in the local
  * database file.</p>
  */
  private void loadOptions() {
      RecordStore database = null;
      try {
          // open the database of stored stock symbols
          database = RecordStore.openRecordStore(PREFERENCES_DATABASE, true);
          // we store the display format option value as the only record in the
          // database.
          String displayFormatString = null;
          RecordEnumeration enum = database.enumerateRecords(null, null, false);
          // This loop will just extract the first element (our display format
          // option) and then quit the loop.
          while (enum.hasNextElement()) {
              displayFormatString = new String(enum.nextRecord());
              break;
          }
          // set the display format
          if (displayFormatString != null) {
              // Convert the stored string into an integer that represents the
              // display time format.
              int displayFormat = Integer.parseInt(displayFormatString);
```

```
                // Now, set it as the currently selected display time format.
                mainMenu.setDisplayFormat(displayFormat);
            }
        }
        catch (Exception e) {
        // If anything goes wrong, we will see a stack trace.
            e.printStackTrace();
        }
        finally {
            // we have (hopefully) retrieved our data. Tidy up and close the
            // database.
            if (database != null) try {
                database.closeRecordStore();
            }
                catch (Exception e) {
            }
        }
    }
```

5단계 : saveOptions() 메소드 생성하기

또한 레코드 저장소에 사용자 옵션을 저장하기 위한 메소드를 생성할 필요가 있습니다. loadOptions() 뒤에 커서를 위치시키고, 다음의 saveOptions() 메소드를 작성합니다.

```
/**
* <p>Save user preferences to database.</p>
*/
private void saveOptions() {
    RecordStore preferencesDb = null;
    try {
        // Remove the old preferences database to be sure that we save a clean
        // copy. (overwriting it may not overwrite properly so we delete it
        // instead).
        RecordStore.deleteRecordStore(PREFERENCES_DATABASE);
        // Now create a fresh preferences database and store off our
        // user preferences.
        // (The openRecordStore() method creates a new database if it can't find
```

```
            // one with the given name. Since we just deleted the old one, it will
            // just create a new database.)
            preferencesDb = RecordStore.openRecordStore(PREFERENCES_DATABASE,
            true);
            // Retrieve the currently selected display time format.
            int displayFormat = mainMenu.getDisplayFormat();
            // Store the display format as a string.
            // First, convert the display format to a string.
            String displayFormatString = String.valueOf(displayFormat);
            // Now, add it to the database.
            preferencesDb.addRecord(
            displayFormatString.getBytes(), // The string bytes.
            0, // Record offset.
            displayFormatString.getBytes().length); // Number of bytes to store.
        } catch (Exception e) {
            // If anything went wrong, we will get a stack trace.
            e.printStackTrace();
        } finally {
            // Close the database.
            if (preferencesDb != null)
            try {
                    preferencesDb.closeRecordStore();
             } catch (Exception e1) {
          }
        }
    }
```

6단계 : 시작할 때 저장된 사용자 옵션 로딩하기

startApp() 메소드안의 Display.getDisplay(this).setCurrent(mainMenu) 라인 전에 새로운 loadOptions() 메소드를 호출하는 부분을 추가합니다.

:

작업이 끝나면 startApp() 메소드는 다음과 같을 것입니다.

```
public void startApp() {
    // Load any previously-saved user options.
    loadOptions();
```

```
        // Display the main stop watch screen.
        Display.getDisplay(this).setCurrent(mainMenu);
    }
```

7단계 : 종료 전에 사용자 옵션 저장하기

최종적으로 MIDlet을 종료시키기 전 destroyApp() 메소드내에, 레코드 저장소로 사용자 옵션을 저장할 새로운 saveOptions() 메소드를 호출하기 위한 라인을 추가할 필요가 있습니다. 또한 지정된 레퍼런스를 풀고 mainMenu를 null로 설정합니다.

destroyApp() 메소드 안에 커서를 위치시키고, 다음과 같은 코드를 삽입합니다.

```
public void destroyApp(boolean unconditional) {
    // Save all options.
    saveOptions();
    // Release references.
    mainMenu.destroy();
    mainMenu = null;
}
```

8단계: MIDlet 저장과 테스트

프로젝트를 저장하고 새로 컴파일하여 에뮬레이터에서 실행시키고 테스트 해 본다. 디스플레이 옵션을 선택하여 선택한 형식이 나타나는 지 stop watch를 실행시킨다. 컴파일 시에 에러가 나타나고, 어느 부분에서 발생하였는지 알 수 없다면 <JBuilder>\samples\Mobile\Stopwatch\ 디렉토리 안의 StopWatch 샘플을 열어서 코드를 비교해 보도록 합니다. 이것으로 StopWatch 튜토리얼은 끝내도록 합니다. 좋은 경험이 되었기를 바랍니다.

Acronym

MIDP : Mobile Information Device Profile

IDE : Integrated Development Environment

CLDC : Connected Limited Device Configuration

J2ME : JavaTM2 Platform, Micro Edition

OTA : Over The Air

JBuilder™ Mobileset 개발자 가이드를 이용해 주셔서 감사합니다.

볼랜드 및 자바 개발자 여러분께 높은 수준의 번역 문서를 제공해 드릴 수 있도록 ㈜아이모바일 테크놀러지는 최선을 다하겠습니다.

감 사 합 니 다.

(www.imobiletek.com, www.borlandexpert.com)

볼랜드 제품구입 문의는 (TEL:02-873-2249 FAX:02-873-2257)로 해 주시기 바랍니다.

문서번호 : IMOBILE-JBM001