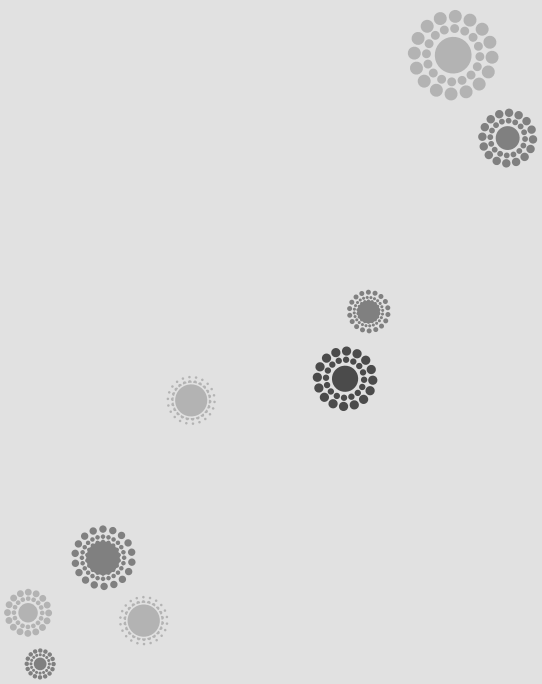
APPENDIX

안드로이드 2.2



1 Froyo

2010년 5월 21일 안드로이드 SDK 2.2 업데이트가 발표되었다. 본문은 2.1 버전을 기준으로 작성되어 있으므로 부록을 통해 간단하게나마 새로 발표된 SDK의 개발환경 설정 방법과 추가 기능에 대해 소개한다. 아직 상세한 문서를 구하기 어렵고 DevGuide와 공식 레퍼런스만을 참고하여 분석한 것이지만 새 SDK의 기능을 둘러 보기엔 부족하지 않을 것이다.

안드로이드 2.2의 코드명은 Frozen Yoghurt의 약자인 Froyo로 붙여졌다. 번역하자면 얼린 요구르트라는 뜻인데 안드로이드는 빵이나 디저트류로 코드명을 붙이며 Donut, Cupcake, Eclair 식으로 첫자 알파벳이 1씩 증가한다. Froyo의 다음 버전은 마늘빵(Ginger Bread)로 이미 명명되어 있으며 2010년 하반기에 발표될 예정이다.

Froyo는 이전 버전인 Eclair에 비해 버전이 0.1만큼 증가한 마이너 업그레이드 버전이지만 이전의 업데이트에 비해서는 굵직한 기능 개선이 많이 이루어졌다. 그동안 아쉽다고 여겨졌던 문제들이 대부분 해결되어 뭔가 중무장을 하고 나타난 묵직한 느낌이 든다. 2.2에서 추가된 주요 기능들의 목록은 다음과 같으며 개별 기능들에 대해서는 잠시 후 천천히 연구해 보기로 하자.

- 전반적인 성능 향상
- 외부 저장 장치에 앱 설치 지원
- 데이터 백업 지원
- 익스체인지 지원 강화
- 플래시 10.1 지원
- OpenGL ES 2.0 지원
- 최대 8대까지 연결할 수 있는 포터블 핫스팟 기능 제공
- 음성 인식 기능 제공

이 외에 홈 스크린의 UI가 대폭적으로 변경되었으며 카메라, 캠코더의 기능들도 확장되었다. 새로운 기능을 지원하기 위한 클래스들이 추가되었고 기존 클래스의 메서드들도 추가 또는 변경되었다. 코드와 관련된 부분은 잠시 후 예제를 작성해 보면서 상세하게 연구해 볼 것이다.

2.2의 개선 사항중 가장 먼저 눈에 띄는 부분은 성능 향상이다. 모바일 장비들은 아직까지 PC

에 비해서는 CPU의 속도가 느리므로 운영체제의 효율성이 중요한 관건인데 Froyo에서 이 부분이 많이 개선되었다. 구체적인 향상 정도는 다음과 같다.

① 달빅 JIT 컴파일러를 최적화하여 CPU를 과중하게 사용하는 코드의 실행 속도가 2~5배 정도 빨라졌다.
② 개선된 자바 스크립트 엔진인 V8 엔진을 채용하여 자바 스크립트를 많이 사용하는 웹 페이지의 로딩 및 실행 속도가 4배 정도 향상되었다.
③ 메모리 관리 능력이 20배 정도 개선되어 작업 전환이 부드러워졌다. 제한된 메모리를 가진 장비에서 메모리 회수 속도가 개선되면 전반적인 성능이 향상된다.

물론 이는 어디까지나 제작사의 주장이므로 액면 그대로 다 믿을 필요는 없다. 특정한 부분에 대한 속도가 개선되었다는 것이지 전체적인 속도를 얘기하는 것은 아니므로 최종 사용자가 느끼는 체감 속도 향상은 이보다 덜할 것이다. 아직 실장비가 없어 확인할 수 없지만 해외 사이트의 테스트 결과를 보면 이전 버전에 비해 눈에 띄게 성능이 향상되었다고 한다.

기능 개선은 많이 이루어졌지만 개발 문서에 대한 지원은 별다른 개선이 없다는 점이 못내 아쉽다. 레퍼런스에는 아직도 비어 있는 항목들이 많으며 오타도 제대로 수정되지 않았는데 아직 그럴 여유는 없는 모양이다. 운영체제의 기능만큼이나 개발자들에게 제대로 된 정보를 제공하는 것도 중요하다. 이후 구글에서 공식 문서에 좀 더 투자해 주기를 바란다.

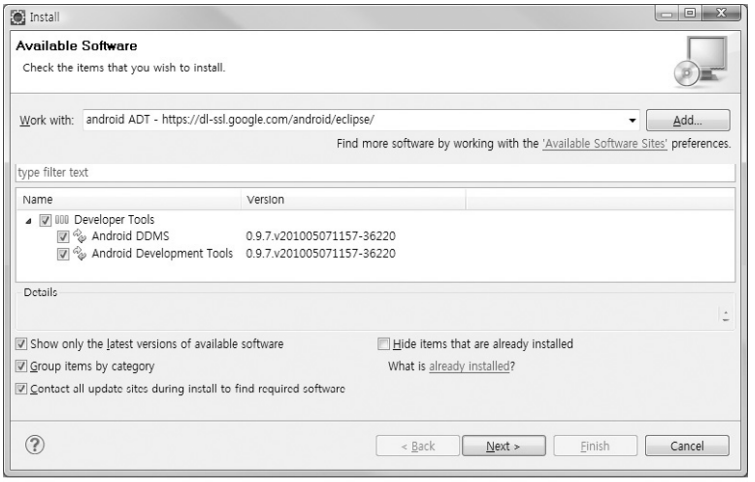
2 개발툴 업데이트

새 버전에 추가된 기능들을 구경해 보려면 개발 환경부터 업데이드해야 한다. 부록은 2.1이 설치되어 있다는 가정하에 작성된 것이므로 업그레이드 위주로 설치 과정을 설명한다. 새로 설치하는 경우는 당연히 업데이트와는 다르므로 본문의 1장을 따라 하되 다운로드하는 파일과 설치 과정에서 선택하는 항목만 달라질 뿐 설치 방법은 사실상 동일하다.

주 개발툴인 이클립스는 SR2로 소폭 업그레이드되었지만 버전은 여전히 3.5이므로 굳이 업그레이드하지 않아도 상관없다. JDK 6도 Update18에서 20으로 개정되었지만 안드로이드는 JDK의 일부 기능만 사용하므로 역시 업데이트할 필요는 없다. 물론 둘 다 최신 버전으로 업데이트해도 무방하며 새 컴퓨터에 설치할 때는 당연히 최신 버전을 설치하는 것이 좋다. SDK 업데이트 과정은 자동화되어 있어 별도로 파일을 다운로드받거나 할 필요없이 이클립스 내에서 모든 작업이 가능하다. 아마 이후 발표되는 SDK도 동일한 방법으로 업그레이드가 가능할 것

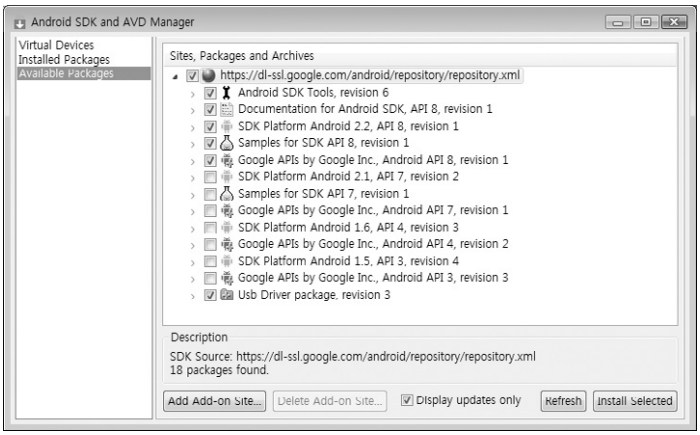
이다. 여러 단계를 거쳐야 하고 순서를 잘 지켜야 하므로 다소 번거롭기는 하지만 통합 환경 안에서 모든 업그레이드를 수행할 수 있으므로 많이 편리해졌다. 여러 번 테스트해 본 결과 다음 순서대로 업데이트하는 것이 가장 간편하다.

SDK를 업데이트하기 전에 ADT를 업데이트해야 한다. 기존의 2.1 버전에서는 0.9.5가 사용되었는데 2.2와 함께 0.9.7로 업그레이드되었으며 일부 기능이 개선되었다. 이클립스 메뉴에서 Help/Install New software 명령을 선택한 후 Work with 목록에서 https://dl-ssl.google.com/android/eclipse/ 를 선택한다. 이전에 설치를 한 적이 있다면 이 주소가 목록에 기억되어 있으므로 콤보 박스에서 선택만 하면 된다. ADT와 DDMS의 0.9.7 새 버전이 목록에 나타나는데 둘 다 선택한 후 설치한다.

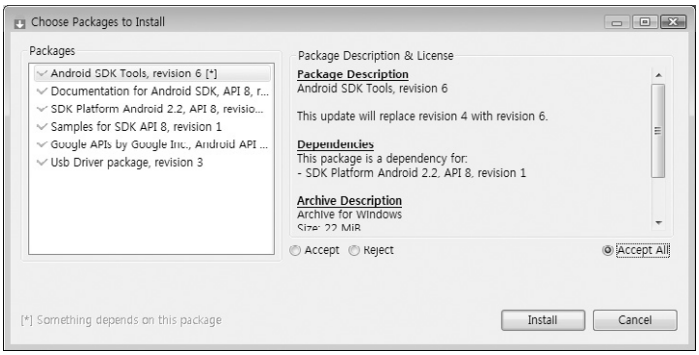


[Next] 버튼을 누르면 설치가 시작된다. 설치중에 서명이 없다는 경고가 나타날 수도 있는데 믿을 수 있는 툴이므로 경고를 무시해도 상관없다. 네트워크를 통해 필요한 파일을 다운로드받는데 만약 방화벽이 다운로드를 막으면 풀어 주어야 한다. 잠시 기다리면 ADT 설치가 완료되며 새로운 ADT로 교체하기 위해 이클립스 재시작해야 한다.

다음은 SDK를 업그레이드한다. 이클립스의 Windows/Android SDK and AVD Manager 메뉴를 선택하고 왼쪽 목록에서 Available Packages를 선택한다. 설치 가능한 소프트웨어 목록이 나타나는데 시스템에 어떤 버전이 설치되어 있는지에 따라 실제 목록은 달라진다. 2.1까지만 설치된 경우의 목록은 다음과 같다.



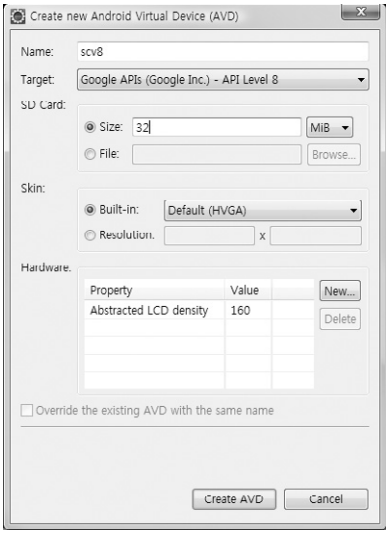
이 목록에서 설치하고자 하는 항목을 선택한다. SDK Tools 버전 6과 SDK API 8, 문서, 샘플 예제, Google API 8 등은 필수적으로 선택해야 한다. 제일 아래쪽의 Usb Driver package도 선택하도록 하자. 이전 버전 개정판은 프로젝트에 꼭 필요하지 않는 한 굳이 설치할 필요없다. 설치할 항목을 선택한 후 [Install Selected] 버튼을 눌러 다음 단계로 넘어간다.



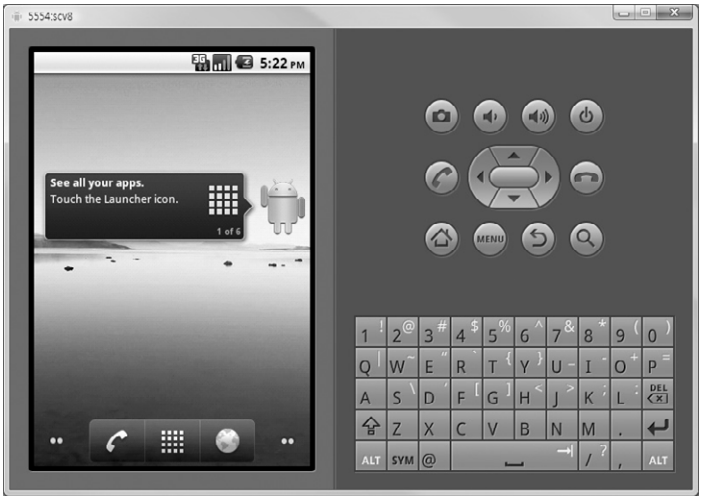
설치 대상 항목과 라이센스 동의문이 나타나는데 Accept All을 선택하고 [Install] 버튼을 누르면 설치가 시작된다. 대용량의 자료를 다운로드 받아야 하므로 다소 시간이 걸린다. 중간에 ADB를 재시작하겠다는 메시지가 나타나면 확인 버튼을 눌러 ADB를 재시작하되 이클립스를 다시 시작할 필요는 없다.

여기까지 진행하면 개발을 위한 SDK는 모두 설치한 것이다. 다음은 새 SDK로 작성한 예제를 실행할 AVD를 생성한다. 아직 실장비가 없으므로 에뮬레이터를 통해서만 새 버전의 앱을

실행해 볼 수 있다. Virtual Devices 페이지에서 [New] 버튼을 눌러 에뮬레이터를 새로 생성한다. scv8이라는 이름을 주고 타겟은 Level 8의 Android 2.2 또는 Google APIs를 선택한다. SD 카드는 32메가 정도면 충분하다. 아래쪽의 [Create AVD] 버튼을 누르면 AVD가 생성된다.



제대로 생성되었는지 목록에서 [Start] 버튼을 눌러 시험 기동시켜 보자. 잠시 기다리면 다음과 같은 에뮬레이터가 실행될 것이다. 이제 2.2 버전의 SDK로 예제를 작성해서 이 에뮬레이터로 보내면 실행해 볼 수 있다.



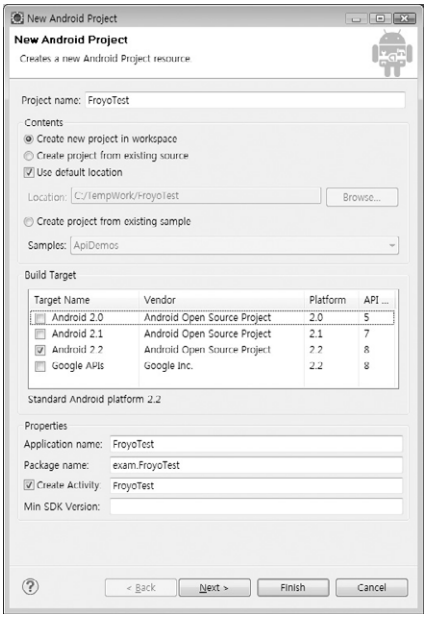
에뮬레이터 스킨은 바뀌지 않았지만 버전이 올라간만큼 홈 스크린이 많이 바뀌었다. 제일 먼저 눈에 띄는 것은 중앙에 있는 팁 위젯인데 초록색 로봇이 초보 사용자에게 간단한 사용법을 알려주는 역할을 한다. 최초 6개의 도움말이 나타나며 터치하면 다음 도움말을 볼 수 있다. 보기 싫으면 로봇을 드래그해서 쓰레기통에 던져 버린다.

하단에는 전화, 론처, 브라우저가 고정된 쇼트컷에 배치되어 있다. 홈 스크린은 모두 5개의 페이지로 구성되어 있고 드래그해서 좌우로 이동 가능하지만 고정된 쇼트컷은 스크롤되지 않고 어느 페이지에서나 보이므로 빠르게 선택할 수 있다는 점에서 편리하다. 중앙의 론처 버튼을 누르면 기본 제공되는 프로그램의 목록이 나타난다.

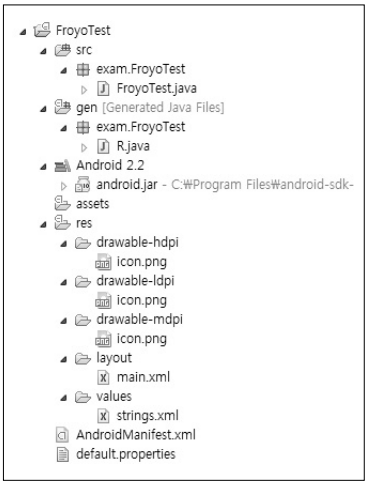
아쉽게도 사용자 ID에 한글이 포함된 경우 에뮬레이터가 실행되지 않는 버그는 아직 수정되지 않았다. 만약 에뮬레이터가 기동되지 않는다면 본문 52페이지 내용대로 AVD의 경로를 한글이 없는 경로로 옮겨야 한다.

3 테스트 예제

SDK가 제대로 설치되었는지, 새 SDK의 프로젝트는 어떤 모습인지 테스트 프로젝트를 만들어 보자. 프로젝트를 생성하는 방법은 이전 버전과 동일하다. 메뉴에서 File/New/Android Project 항목을 선택하고 생성할 프로젝트의 정보를 입력한다.



프로젝트의 이름은 FroyoTest로 주고 타겟은 Android 2.2로 설정한다. 제일 아래쪽의 Min SDK Version은 8로 설정하거나 아니면 비워 두어도 상관없다. Properties란에 앱 이름, 패키지 명, 액티비티명을 적당히 입력한 후 [Finish] 버튼을 누르면 프로젝트가 생성되며 패키지 탐색기에 프로젝트가 나타날 것이다.



프로젝트의 폴더 구조나 생성되는 파일의 목록은 이전 버전과 완전히 동일하다. 레이아웃 파일은 다음과 같다. 부모의 크기를 다 사용하라는 의미의 fill_parent가 2.2 버전에서 match_parent로 변경되었지만 마법사는 여전히 fill_parent를 사용한다. 두 플래그 모두 당분간은 계속 사용할 수 있을 것으로 보인다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

리니어안에 텍스트 뷰가 배치되어 있으며 문자열 리소스가 지정되어 있다. 소스 코드는 이전

버전에 비해 전혀 바뀌지 않았다. onCreate에서 레이아웃을 전개하여 액티비티에 가득 채우기만 한다.

ProyoTest.java

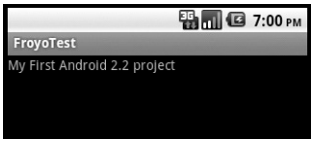
```
package exam.FroyoTest;

import android.app.Activity;
import android.os.Bundle;

public class FroyoTest extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

예제를 실행해 보자. Run/Run 항목을 선택하거나 Ctrl + F11 을 누르면 다음 대화상자가 나타난다. 아무 항목도 선택되어 있지 않으므로 아래, 위 키를 누른 후 Enter 를 눌러야 하는데 이후부터는 Ctrl + F11 을 누르면 바로 실행된다.

에뮬레이터가 실행되며 빈 화면에 문자열만 나타날 것이다. 수정도 잘 되는지 코드를 약간 바꿔 보자. main.xml의 텍스트 뷰를 다음과 같이 수정한다. 새로 추가된 match_parent 레이아웃 값을 적용해 보고 문자열을 변경해 보았다.

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="My First Android 2.2 project"
    />
```

수정한 파일을 저장하고 다시 실행하면 수정한 문자열이 화면에 나타날 것이다. 수정 후 재실행할 때 xml 파일에서 Ctrl + F11을 누르거나 툴바의 [Run] 버튼을 누르면 xml 자체를 실행하라는 것으로 해석하여 main.out.xml 파일을 만들고 에러를 내므로 반드시 java 파일로 포커스를 옮긴 후 실행해야 한다.



다행스럽게도 프로젝트를 생성하고 관리하는 방법상의 변화는 거의 없다. Ctrl + F11을 누를 때 별도의 질문없이 바로 실행한다는 점에서 약간의 기능 개선이 있는 정도이다. 기존 방법대로 프로젝트를 만들고 실습을 진행하면 된다.

다음은 기존 예제가 새 버전의 에뮬레이터에서 잘 실행되는지 호환성 테스트를 해 보자. 이 책의 통합 예제인 AndroidExamAll.zip을 Import하면 아무 문제없이 컴파일되고 실행된다. 레벨 8의 에뮬레이터에서 레벨 7의 예제가 정상 실행되는 것은 당연하다고 할 수 있다. 어쨌거나 하위 호환성이 잘 유지된다는 면에서 무척 다행스러운 일이다.

이전 버전의 프로젝트를 계속 관리해야 한다면 새 SDK를 설치해 놓은 상태에서도 이전 버전으로 계속 개발할 수 있다. 새 기능이 꼭 필요치 않다거나 타겟 장비의 버전이 낮다면 프로젝트의 버전을 굳이 8로 올리지 않아도 무방하다.

4 외부 메모리에 앱 설치

모바일 장비의 메모리는 폰에 내장되어 있는 내부 메모리와 SD 카드처럼 별도로 확장 가능한 외부 메모리로 구분된다. 내부 메모리는 항상 장착되어 있어 언제나 사용 가능하다는 이점이 있지만 가격이 비싸므로 용량이 비교적 작다. 외부 메모리는 용량이 크고 가격이 저렴한데 현재 16G 정도가 겨우 4만원 수준이다. 그러나 탈착 가능하므로 항상 사용 가능한 상태가 아니라는 단점이 있다.

안드로이드의 앱은 내부 메모리에만 설치할 수 있으며 외부 메모리에는 설치할 수 없도록 되어 있는데 이런 제한이 합당한 여러 가지 이유가 있다. 설치된 프로그램이 언제나 사용 가능해야 하지만 교체 가능한 외부 메모리에 설치할 경우 외부 메모리 분리시 시스템의 설정이 깨진다는 위험이 있다. 또 교체 가능한 메모리에 설치를 허락하면 마켓에서 내려받은 유료 프로그램의 불법 복제가 난무하는 문제도 간과할 수 없다.

합당하기는 하지만 이 제한에 의해 장비에 애초에 내장된 메모리 용량을 초과하는 앱은 설치할 수 없다는 심각한 문제가 있으며 실제로 안드로이드 폰 출시 후 상당수의 사용자들이 이 문제로 인해 불편을 호소했다. 이런 불편함이 2.2 버전에서 드디어 해결되었다. 2.2 버전부터는 외부 메모리에도 앱을 설치할 수 있다. 이를 위해 안드로이드는 매니페스트에 설치 가능한 위치를 지정하는 installLocation 속성을 추가했다.

속성	설명
internalOnly	내부 메모리에만 설치할 수 있으며 외부 메모리에는 설치할 수 없다. 내부 메모리가 부족하면 설치는 실패한다. 이 값이 디폴트다. 별 지정이 없거나 2.2 이전의 앱은 이 속성을 지정한 것으로 간주된다.
preferExternal	가급적이면 외부 메모리에 설치한다. 만약 외부 메모리가 부족하거나 존재하지 않으면 내부 메모리에 설치된다. 사용자는 설치 후에도 위치를 변경할 수 있다.

auto	시스템이 몇 가지 조건에 따라 설치 위치를 결정한다. 디폴트로 내부 메모리에 설치하지만 내부 메모리가 부족하면 외부 메모리에 설치한다. 사용자는 설치 후에도 위치를 변경할 수 있다.

외부 메모리에 설치하더라도 속도상의 불이익은 거의 없으며 사용자 데이터나 DB 파일 등은 여전히 내부 메모리에 저장된다. 외부 메모리에 설치되는 프로그램은 암호화되어 저장되므로 최초 설치한 장비에서만 실행된다. 외부 메모리를 다른 장비로 옮기거나 파일을 복사해서는 실행되지 않으므로 불법 복사는 할 수 없다. 만약 실행중에 외부 메모리가 제거되면 외부 메모리에 설치된 앱은 강제로 종료된다.

외부 메모리 지원에 의한 부작용을 최소화하기 위한 여러 가지 안전 장치가 같이 마련되었음을 알 수 있다. 외부 메모리에 설치 가능한 테스트 프로그램을 작성해 보자. 마법사로 다음 프로젝트를 생성한다.

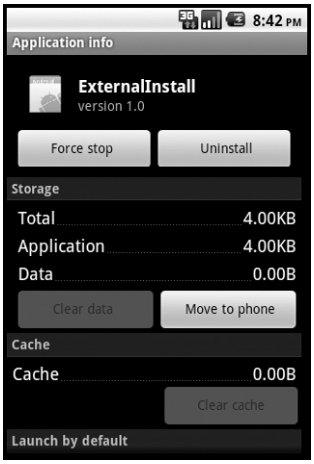
ExternalInstall.java

```
public class ExternalInstall extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

     public boolean onTouchEvent(MotionEvent event) {
         super.onTouchEvent(event);
         if (event.getAction() == MotionEvent.ACTION_DOWN) {
             Toast.makeText(this,"Touch Event Received",
                     Toast.LENGTH_SHORT).show();
             return true;
         }
         return false;
     }
}
```

아무 동작이 없으면 심심하므로 터치 입력을 받았을 때 토스트만 살짝 띄워 보았다. 외부 설치를 허가하기 위해 매니페스트에 다음 속성을 설정한다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      android:installLocation="preferExternal"
      package="exam.ExternalInstall"
      android:versionCode="1"
      android:versionName="1.0">
```

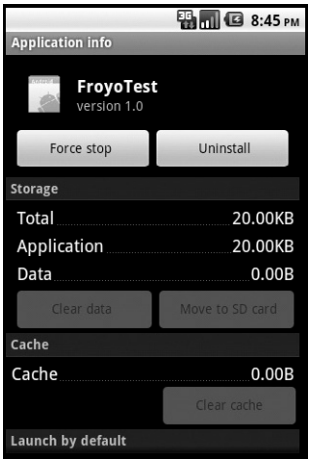
preferExternal 속성으로 지정하여 가급적 외부 메모리에 설치하도록 했다. 실행하면 에뮬레이터에 설치되고 빈 화면이 나타나며 터치 입력도 잘 받는다. 어디에 설치되어 있는지 Settings/Applications/Manage applications 메뉴를 찾아 들어가 ExternalInstall 예제를 선택해 보자. 다음 관리창이 뜰 것이다.



오른쪽 가운데에 Move to phone 버튼이 배치되어 있다. 이 버튼은 프로그램 설치 위치를 내부 메모리로 이동하라는 명령이며 현재는 외부 메모리에 설치되어 있음을 알 수 있다. 과연 그런지 DDMS의 파일 탐색기로 확인해 볼 수 있다.

이 버튼을 누르면 내부 메모리로 옮겨지며 버튼의 캡션은 Moving으로 바뀌었다가 잠시 후 Move to SD card로 변경된다. 즉 이 버튼은 현재 설치된 위치를 토글하는 기능을 제공하는

데 필요에 따라 위치를 자유롭게 옮길 수 있다. 별다른 위치를 지정하지 않은 FroyoTest 예제의 경우는 어떤지 보자.



Move to SD card 버튼이 사용 금지되어 있다. 즉, 현재는 내부 메모리에 설치되어 있는데 외부 메모리로 옮길 수는 없다는 뜻이다. 2.1 버전 이전의 모든 예제들은 installLocation 속성이 지원되지 않으므로 무조건 내부 메모리에만 설치할 수 있다. 이전 버전의 프로그램도 외부 메모리에 설치 가능하게 하려면 매니페스트를 수정한 후 다시 컴파일해야 한다. 최소 SDK 설정(minSdkVersion)은 그대로 두고 타겟 레벨만 8로 변경한 후 이 속성만 지정하는 것도 가능하다. 타겟을 8로 변경하더라도 최소 SDK 버전은 그대로 유지되므로 이전 버전의 장비에서도 문제없이 설치 및 실행된다.

외부 설치가 가능하려면 타겟 장비는 반드시 2.2 버전이어야 한다. 이전 버전에는 외부 설치 기능이 없으므로 속성을 지정해도 인식하지 않으며 에러는 나지 않지만 무시당한다. 응용 프로그램은 2.2 버전으로 컴파일했더라도 설치되는 장비가 2.1이면 외부 설치는 불가능하다. 사실 2.2 앱을 2.1 장비에 설치하는 것 자체가 불가능하다.

외부 설치 기능은 값비싼 메모리를 절약한다는 면에서 유용하지만 반드시 내부에만 설치해야 하는 앱도 있다. 서비스나 앱 위젯, 알람 등의 앱은 설치 후 항상 사용 가능해야 하므로 반드시 내부에 설치해야 하며 SD 카드를 제거하더라도 시스템의 설정이 깨지지 말아야 한다. 예를 들어 MP3 플레이어는 재생을 위해 서비스를 등록하는데 이 서비스는 언제든지 사용 가능해야

한다. 외부에 설치된 서비스는 SD 카드 제거시 강제 종료되며 SD 카드를 다시 마운트해도 자동으로 재시작되지 않는다.

앱 위젯의 경우는 더 심각해서 SD 카드 제거시 부팅을 다시 해야만 재실행된다. SD 카드가 있을 때만 재생 가능해서는 안되므로 이런 앱은 반드시 내부에 설치해야 한다. 서비스 류는 응용 프로그램이라기보다는 운영체제의 확장이므로 외부 메모리 상황과 상관없이 항상 사용 가능해야 한다. 이점은 사실 PC도 마찬가지인데 USB 외장 하드에 오피스나 개발툴 따위를 설치하는 사람은 없다. D 드라이브에 프로그램을 설치해 놓았을 때 D 드라이브를 떼 버리면 컴퓨터는 완전 바보가 된다.

반면 외부에 설치해도 별 문제가 없는 프로그램도 많다. 다른 프로그램과 엮이지 않는 독립적인 프로그램은 외부에 설치해도 무방하다. 그래픽 이미지를 많이 사용하는 게임이나 대용량의 데이터를 필요로 하는 사전류들은 가급적이면 외부에 설치하는 것이 유리할 것이다. 외부 설치된 앱은 SD 카드를 제거하면 당연히 실행할 수 없으며 SD카드가 마운트되어 있을 때만 실행 가능하다. 게임은 강제 종료되어도 언제든지 재실행할 수 있으므로 문제될 것이 없다. 두 개의 SD 카드에 각각 다른 프로그램들을 설치해 놓고 번갈아 가며 쓸 수도 있다.

이 기능은 사실 꼭 필요해서 만들었다기보다는 비난을 받기 싫어서 넣은 기능이다. 구글도 이 문제를 몰랐을 리가 없으며 사실 앱은 내부 메모리에만 설치하는 것이 논리적으로 옳다. 그러나 두 메모리의 용량, 가격차가 너무 현격하기 때문에 초기 실장비를 제작하는 업체가 내부 메모리를 충분히 제공하지 않았고 그러다 보니 안드로이드의 큰 약점으로 부각된 것이다. 다른 운영체제와 비교를 당하는 것이 구글 입장에서 결코 유쾌하지 않았을 것이며 또 저가의 장비도 지원해야 한다는 명분으로 인해 이 기능이 늦게나마 포함된 것이다.

메모리 값이 더 저렴해질 것이라는 것은 누구나 다 아는 사실이고 앞으로 발표되는 장비는 충분한 내부 메모리를 제공할 것이다. 최근 발표된 Galaxy S는 2G의 설치 메모리와 16G의 내장 메모리를 제공해 앱 설치에 거의 문제가 없다. SD 카드는 대용량의 동영상이나 사진, 음악 파일을 저장하는 것이 본래 용도이지 앱 설치용은 아니다. 다만 백과 사전류 같은 특수한 예외들이 있기 때문에 SD 카드 설치도 허락하게 된 것이다.

5 백업 및 복구

모바일 장비도 PC와 마찬가지로 프로그램을 이것 저것 깔다 보면 점점 느려지므로 가끔은 깔끔하게 포맷을 다시 해야 한다. 리셋하는 방법은 뻔하다. 이전 데이터 백업해 놓고 프로그램 재설치 후 원래 데이터를 가져 오는 것이다. 장비를 교체할 때도 동일한 과정을 반복해야 하는데 이전 데이터와 세팅을 새 장비에 수작업으로 복원하는 것은 굉장히 번거로운 일이다. 장비를 분실한 경우는 원본 데이터가 없으므로 아예 복구가 불가능하다.

Froyo는 이런 경우를 위해 데이터와 세팅을 백업하고 복원하는 솔루션을 제공한다. 응용 프로그램이 주요 데이터와 세팅을 입출력하는 에이전트를 작성해 두면 백업 관리자가 필요할 때 에이전트를 호출하여 백업과 복구를 수행하는 식이다. 이 과정은 사용자에게는 보이지 않으며 백그라운드에서 완전히 자동으로 수행된다. 저장된 데이터는 복구시에만 읽을 수 있으며 사용자가 임의로 액세스할 수 없으므로 싱크와는 다른 개념이다.

데이터가 백업되는 장소는 클라우드 저장소(cloud storage)인데 구체적인 위치는 장비와 사업자에 따라 달라진다. 대개의 경우는 원격지의 네트워크이겠지만 실제 위치는 굳이 몰라도 상관없다. 사업자는 백업 지원을 위해 별도의 저장소를 준비해야 하는 부담이 있지만 사용자들이 이 기능에 맛을 들이면 도저히 벗어날 수 없는 충성스런 고객이 되므로 사업자 입장에서 아주 매력적인 기능이라고 할 수 있다. 저장된 정보를 다른 앱이 사용할 수는 없지만 보안이 완벽한 것은 아니므로 패스워드같은 민감한 데이터는 저장하지 말아야 한다.

백업/복원 기능을 사용할 응용 프로그램은 내부에 에이전트를 구현해 놓고 백업 관리자의 요청이 있을 때 백업할 데이터를 전달한다. 에이전트는 백업 관리자와 통신하는 응용 프로그램 내부의 객체이며 미리 약속된 방식으로 백업 관리자와 협조적으로 데이터를 백업 및 복원한다. 응용 프로그램은 다음 두 가지 방법으로 에이전트를 구현한다.

- BackupAgent 클래스 확장 : 백업 관리자에 의해 호출되는 onBackup, onRestore 메서드를 직접 구현하는 방식이다. 섬세한 버전 관리를 할 수 있으며 파일의 일부만 선택적으로 백업할 수 있고 데이터베이스에 저장할 수도 있다. 자유도는 높지만 스트림을 직접 제어하므로 예외 처리가 완벽해야 하며 파일 액세스시 동기화에도 유의해야 하므로 다량의 코드가 필요하고 난이도도 높은 편이다. 고급 백업 기능이 필요치 않으면 가급적 이 방법은 사용하지 않는 것이 좋다.
- BackupAgentHelper 클래스 확장 : 이 클래스에는 BackupAgent의 기능 일부가 미리 구현되어 있으며 onBackup, onRestore를 내부에서 알아서 처리한다. 응용 프로그램은 특정 타입의 데이터를

입출력하는 도우미 객체를 등록해 놓기만 하면 된다. FileBackupHelper 클래스는 파일을 입출력하고 SharedPreferencesBackupHelper 클래스는 프레프런스의 세팅을 저장 및 복원한다. 도우미에 백업 대상 파일이나 프레프런스 키의 목록을 전달하고 addHelper 메서드로 도우미를 등록해 놓기만 하면 나머지는 자동으로 수행된다.

백업 관리자가 통신할 대상을 찾을 수 있어야 하므로 에이전트의 이름을 매니페스트에 기록해 놓아야 한다. application 엘리먼트의 backupAgent 속성에 에이전트 클래스 이름만 기록해 놓으면 백업 관리자가 이 객체를 생성하여 백업과 복구에 필요한 메서드를 호출할 것이다. restoreAnyVersion 속성은 버전이 달라도 복원할 것인가를 지정하는데 디폴트는 false이다. 버전간의 데이터 구조가 동일하거나 별도의 수동 변환 코드가 있으면 이 속성을 true로 지정한다.

```
<application android:label="MyApplication"
    android:backupAgent="MyBackupAgent">
```

응용 프로그램은 백업 대상 데이터의 일부가 변경될 때 백업 관리자의 dataChanged 메서드를 호출하여 백업을 갱신할 것을 요청한다. 이때 백업 관리자는 에이전트와 함께 데이터를 다시 백업할 것이다. 매 요청이 들어올 때마다 백업을 수행하지는 않으며 요청들을 모아 두었다가 한꺼번에 백업함으로써 백업의 효율성을 높인다. bmgr 쉘 명령을 사용하면 백업 관리자에게 즉시 백업을 명령할 수도 있는데 이 기능은 개발중의 테스트를 위해 아주 요긴하다.

복원은 시스템이 알아서 수행하므로 직접 요청할 필요는 없지만 꼭 수동 복원하려면 requestRestore 메서드를 호출할 수는 있다. 시스템은 응용 프로그램이 설치될 때 클라우드 저장소를 점검해 보고 이전에 백업해 놓은 정보가 있으면 가져와 복원한다. 사용자에게는 이 과정이 보이지 않으므로 새로 설치한 프로그램이 이전에 사용하던 모습 그대로 실행되는 마법 같은 일이 벌어진다.

백업 에이전트를 구현하는 방법에 대해서는 DevGuide에 상세하게 설명되어 있으며 Backup and Restore 샘플 예제도 제공된다. 예제의 길이도 짧고 주석도 풍부하게 기술되어 있어 이 예제를 보면 구현 절차를 어렵지 않게 터득할 수 있다. 그러나 이 예제를 에뮬레이터에서 테스트해 본 결과 백업 및 복구는 되지 않았는데 에뮬레이터는 클라우드 저장소가 없기 때문이다. 이 기능이 동작하려면 개통된 실제 장비가 필요하며 사업자가 저장소를 제공해야 한다.

모바일 장비는 분실 및 교체가 잦으므로 운영체제 차원에서 백업 기능을 제공하는 것은 멋진 일이다. 그러나 실용성에 대해서는 다소 의아스러운데 응용 프로그램마다 별도의 복잡한 코드가 필요하고 백업시마다 네트워크 접속 비용이 발생하는 것도 문제다. 어디서나 무료 WiFi에 접속할 수 있고 배터리가 항상 넉넉하다면 이 기능이 꽤 쓸만할 것이며 특히 기업 사용자에게 아주 유용할 것이다.

6 멀티미디어 기능 개선

멀티미디어 관련 클래스들이 일부 확장되었다. 먼저 사운드를 재생하는 SoundPool 클래스부터 연구해 보자. 이 클래스는 MediaPlayer의 서비스를 사용하므로 압축 포맷을 재생해도 CPU 점유율이 높지 않다. 또한 동시에 여러 개의 사운드를 재생할 수 있어 게임 제작에 유용하며 재생 속도나 볼륨도 개별적으로 지정할 수 있어 활용성이 높다. 다만 대용량의 사운드를 처리할 때 로드 시간이 오래 걸리는 것이 단점인데 이를 보완하기 위해 다음 메서드가 추가되었다.

```
void setOnLoadCompleteListener (SoundPool.OnLoadCompleteListener listener)
```

이 메서드는 음원 로드가 완료될 때 호출되는 리스너를 지정함으로서 대용량의 음원이 로드되는 즉시 재생 가능하다. 리스너를 지정해 놓으면 음원 로드가 완료될 때 다음 메서드가 호출된다.

```
void onLoadComplete (SoundPool soundPool, int sampleId, int status)
```

인수로 전달되는 SoundPool 객체와 로드한 음원의 ID, 상태를 참조하여 음원을 즉시 재생할 수 있다. SoundPool은 동시에 여러 개의 효과음을 재생할 수 있는데 이 기능을 지원하기 위해 재생중인 모든 사운드를 한꺼번에 정지 및 재개할 수 있는 메서드가 추가되었다.

void autoPause ()
void autoResume ()

이전에도 pause, resume 메서드가 제공되었지만 개별적인 스트림 단위로만 정지/재개를 수행했었다. 게임처럼 여러 사운드가 동시에 출력되는 프로그램은 단 하나의 메서드 호출로 전체 사운드를 통제할 수 있으므로 아주 실용적이다. 다음 예제는 상기의 메서드들을 테스트한다.

LoadComplete.java

```
public class LoadComplete extends Activity {
     SoundPool pool;
     int stream;

     public void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.main);

         pool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);

         findViewById(R.id.load1).setOnClickListener(mClickListener);
         findViewById(R.id.load2).setOnClickListener(mClickListener);
         findViewById(R.id.pause).setOnClickListener(mClickListener);
         findViewById(R.id.resume).setOnClickListener(mClickListener);
     }

     SoundPool.OnLoadCompleteListener mListener =
               new SoundPool.OnLoadCompleteListener() {
         public void onLoadComplete(SoundPool soundPool, int sampleId, int status) {
             if (status == 0) {
                 stream = soundPool.play(sampleId, 1, 1, 0, 0, 1);
             }
         }
     };

     Button.OnClickListener mClickListener = new Button.OnClickListener() {
         public void onClick(View v) {
             MediaPlayer player;
             switch (v.getId()) {
             case R.id.load1:
                  int song = pool.load(LoadComplete.this, R.raw.goodtime, 1);
                 pool.play(song, 1, 1, 0, 0, 1);
```

```
                break;
            case R.id.load2:
                pool.setOnLoadCompleteListener(mListener);
                pool.load(LoadComplete.this, R.raw.goodtime, 1);
                break;
            case R.id.pause:
                //pool.pause(stream);
                pool.autoPause();
                break;
            case R.id.resume:
                //pool.resume(stream);
                pool.autoResume();
                break;
            }
        }
    };
}
```

레이아웃에는 버튼 4개가 배치되어 있으며 raw 폴더에는 "즐거운 시간 되십시오" 라는 음성 리소스가 저장되어 있다. 각 버튼들을 순서대로 눌러 보자.

첫 번째 버튼은 사운드를 로드한 후 곧바로 재생하는데 로드중에 재생하므로 제대로 출력되지 않는다. 음원 크기가 아무리 작아도 로드에는 다소간의 시간이 걸리므로 play 메서드가 호출될 때는 아직 준비가 안된 상태이기 때문이다.

두 번째 버튼은 로드 완료 리스너를 등록해 놓고 로드 명령을 내리며 리스너가 호출될 때 사운드를 재생한다. 이렇게 하면 사운드가 준비된 상태에서 바로 재생 가능하며 로드되는 동안에도 다른 작업을 할 수 있다는 이점이 있다. 물론 이상적인 방법은 메서드 내부에서 사운드를 로드하지 않고 생성자나 onCreate에서 미리 로드해 놓는 것이다. 이 기능은 임의의 사운드를 실행중에 로드해서 재생할 때 유용하다.

나머지 두 버튼은 재생을 잠시 멈추거나 재개하는데 이 예제는 재생하는 사운드가 하나밖에 없으므로 pause, resume으로도 동일한 기능을 구현할 수 있다. 그러나 여러 개의 사운드를 재생 중이었다면 개별 사운드마다 정지를 명령해야 하므로 번거로워진다.

녹음 및 녹화 기능을 담당하는 MediaRecorder 클래스도 채널수, 샘플링 비율 등을 지정하는 메서드가 추가되었다. 고정된 품질로만 녹화를 하는 것이 아니라 품질과 용량을 적절한 수준에서 선택할 수 있다. YouTube 등의 동영상 공유 사이트에 올릴만한 적절한 품질을 직접 선택할 수 있다. 프로파일은 하드웨어의 능력치를 조사하는 읽기 전용의 클래스인데 응용 프로그램이 하드웨어의 모든 능력을 십분 활용할 수 있다.

```
void setAudioEncodingBitRate (int bitRate)
void setAudioSamplingRate (int samplingRate)
void setVideoEncodingBitRate (int bitRate)
void setProfile (CamcorderProfile profile)
```

에뮬레이터는 녹화를 지원하지 않으므로 이 메서드를 테스트하려면 2.2 버전의 실장비가 필요하다. 3D 그래픽 라이브러리인 OpenGL ES도 2.0으로 업그레이드되었다. 그외 ImageForma 클래스가 추가되었으며 YUV 포맷을 관리할 수 있는 API가 추가되었다.

7 기타 개선 사항

굵직한 성능 개선 외에 자잘한 개선 및 변경 사항들도 많은데 간략하게 요약만 하고 추후에 상세하게 연구해 보도록 하자.

fill_parent 이름 변경

레이아웃 속성값인 fill_parent 플래그의 명칭이 match_parent로 변경되었다. fill_parent는 부모의 폭이나 높이를 모두 사용하라는 뜻인데 실제로는 부모의 크기에서 안쪽 여백은 제외된다. 즉, 위젯이 안쪽 여백을 가질 경우는 여백 때문에 부모를 가득 채우지 못하는 상황이 발생하며 따라서 fill이라는 명칭이 직관적이지 못한 경우가 있다. 이런 의미상의 불일치를 해소하기 위해 fill_parent에서 fill이라는 단어를 match로 변경하였다.

당연한 얘기겠지만 fill_parent라는 명칭도 후방 호환성을 위해 계속 사용할 수 있다. 실제 상수 정의를 보면 fill_parent와 match_parent 모두 -1로 정의되어 있으므로 이름만 다를 뿐이지 의미는 같다. 앞으로는 가급적 새로운 속성값을 쓰는 것이 좋겠지만 이전 버전과의 호환성을 유지해야 한다면 당분간은 fill_parent를 고수하는 것이 더 현실적이다. 실제로 2.2의 마법사조차도 match_parent 속성값을 사용하지 않는다.

속성의 이름이 좀 더 분명해진 것은 좋지만 기존 개발자들은 명칭 변경으로 인해 다소 혼란스러워할 것이며 새 명칭을 사용하면 2.2 버전 이상에서만 컴파일된다는 부작용도 생겨 버렸다. 역시 어떤 제품이나 버전이 올라가면 찌꺼기가 생길 수밖에 없는 모양이다. 의미가 조금 틀리더라도 기존 개발자의 지식은 최대한 존중해주는 것이 오히려 더 나은 선택이 아닐까 생각된다.

최적화 금지 플래그 vmSafeMode

매니페스트의 <application> 엘리먼트에 vmSafeMode라는 속성이 추가되었다. ApplicationInfo 클래스에는 이 속성값을 나타내는 FLAG_VM_SAFE_MODE 플래그도 추가되었다. 이 속성이 true이면 JIT 컴파일러의 최적화가 금지된다. JIT 컴파일러는 극한의 효율 향상을 위해 컴파일중에 코드의 최적화를 수행하는데 일반적으로 이 최적화에 의한 부작용은 거의 없다.

그러나 미처 예상하지 못한 민감한 문제로 인해 최적화 전과 후의 동작이 약간씩 달라지는 경우가 드물게 발생하는데 이런 경우 최적화를 금지시켜 개발자의 의도대로 컴파일하도록 강제해야 한다. 응용 프로그램 수준에서는 이 플래그를 사용할 경우가 극히 드물 것이다.

카메라/캠코더의 기능 개선

모바일 기기의 핵심 액세서리인 카메라의 기능이 일부 개선되었다. 기존에는 가로 모드만 지원했으나 세로 모드도 지원하기 시작했으며 줌 지원 API도 추가되었고 노출, 각도, 초점 거리 등의 파라미터가 생겼다. 하드웨어의 능력치를 최대한 조사하여 활용할 수 있는 프로파일 클래스가 추가되었으며 GPS에서 Exif 정보를 좀 더 상세하게 조사할 수 있다.

추가된 기능들은 물론 카메라 제작에 바로 사용할 수 있으며 에뮬레이터에 기본 설치된 카메라에도 이 기능들이 적용되어 있다. 그러나 카메라는 하드웨어에 강하게 종속적이어서 장비 제작사들이 이미 필요한 API를 확장해서 사용하고 있으므로 추가된 API가 당장 성능 개선으로 이어지지는 못할 듯하다. 커스텀 카메라 제작에는 추가된 API 들이 유용하게 사용될 것이다.

익스체인지 지원

마이크로소프트의 메일 솔루션 익스체인지에 대한 지원이 강화되었다. 알파벳과 숫자키를 같이 사용하는 비밀번호를 채택하여 보안성을 높였고 달력과 자동 싱크를 지원한다. 분실된 장비로 인해 민감한 정보가 유출되는 것을 방지하기 위한 원격 장비 리셋 기능도 지원하는데 이는 기업용 장비에 필수적인 기능이다. 전역 디렉토리 목록에서 수신자의 주소를 자동 완성할 수 있는 편의 기능도 지원된다.

핫 스팟 기능 지원

모바일 장비를 휴대용 WiFi 핫 스팟으로 활용할 수 있다. 이 기능을 사용하면 모바일 장비에 최대 8대까지의 장비를 물려 동시에 인터넷을 사용할 수 있다. 이런 기능을 테더링(Ththering)이라고 하는데 폰의 인터넷 접속 기능을 빌려 다른 기기에 인터넷 연결을 제공하는 것이다. 통신 사업자는 이런 기능을 좋아하지 않지만 현실적으로 이를 막을 방법은 없다. 물론 속도는 기대에 미치지 못할 것으로 예상된다.

PC에서 마켓 앱 설치

이전에는 폰에서 마켓을 검색하여 필요한 앱을 설치했으나 이제는 PC에서 마켓을 검색하고 PC에서 앱을 설치하면 폰으로 다운로드된다. PC의 빠른 네트워크와 넓은 화면에서 앱을 신속하게 검색할 수 있고 설치도 편리하다. 마켓은 장비의 구성에 맞는 앱만 필터링하여 보여 주므로 실행 가능한 앱만 설치할 수 있다. 또한 마켓은 새 버전의 앱이 올라 왔을 때 폰에 설치된 프로그램을 자동으로 업데이트하는 기능도 지원한다.

이상으로 안드로이드 2.2에서 확장된 여러 가지 기능들을 소개했고 일부는 예제도 작성해 보았다. 이 외에도 UI Mode 관리자나 음성 인식 등의 흥미로운 기능들이 있다. 부지런히 새 기술을 연구해 봐야겠지만 아직 문서대로 동작하지 않는 기능들도 있고 일부는 실제 장비가 있어야만 테스트해 볼 수 있다.