No.1 Android Book
www.kandroid.org

안드로이드 베이직&클래식

안드로이드 입문서

3rd Edition

구글이 직접 안내한 **The Developer's Guide** 최초 번역

www.kandroid.org 운영자 들풀 양정수
운영진 김은영 · 김남국 공동번역

안드로이드 베이직 & 클래식

안드로이드 입문서

안드로이드 베이직 & 클래식

안드로이드 입문서

삼판 1쇄 발행 2010년 2월 22일

옮 긴 이 | www.kandroid.org 운영자 들풀 양정수, 운영진 김은영 · 김남국 공동번역
펴 낸 이 | 이윤선

주 소 | 서울특별시 서초구 서초동 1338-20 현대렉시온 1519호
전 화 | 02-2182-5531
팩 스 | 02-2182-5532
홈페이지 | www.kandroid.org
펴 낸 곳 | ㈜오버나비

출판신고 | 2009년 02월 26일 제322-2009-000044호

ISBN | 978-89-962353-0-9 (93000)
값 30,000원

이 책은 저작권법에 따라 보호받는 저작물이므로 무단 전재와 무단 복제를 금지하며,
이 책 내용의 전부 또는 일부를 이용하려면 반드시 저작권자와 오버나비의 서면동의를 받아야 합니다.
잘못된 책은 구입하신 사이트 http://www.kandroid.org 에서 바꾸어 드립니다.

안드로이드 베이직 & 클래식

안드로이드 입문서

www.kandroid.org
운영자 들풀 양정수
운영진 김은영·김남국 공동번역

서 문

이 책은, www.android.com의 'The Developer's Guide'에 대한 번역이다.

왜 이 글을 번역했는가?

일반적으로 새로운 API에 대한 애플리케이션 개발 방법을 배우는 데에는 두 가지 방법이 있다. 첫 번째 방법은, 자신이 만들고자 하는 것에 대한 API 사용법을 읽혀 나가면서 플랫폼의 뉘앙스를 배워나가는 것이다. 두 번째 방법은, 애플리케이션 제작에 대한 올바른 방법을 배운 후, 원하는 애플리케이션을 제작하는 방법이다.

두 번째 방법은, 시간을 요하며 또한 그렇게 효율적인 방법이 아니다. 그래서 우리는 일반적으로 첫 번째 방법을 많이 사용한다.

하지만, 이 책은 두 번째 방법으로 안드로이드를 공부하기 위한 책이다.

안드로이드는 오픈 소스 기반이다. 오픈 소스의 장점은 누구나 그 기술을 깊게 살펴볼 수 있고, 많은 기술적 정보들이 공유되고 있다는 것이다. 그 만큼 접근이 용이하고 활용도 용이하다. 하지만, 접근과 활용이 용이한 만큼 기술적 차별화 역시 쉽지 않다.

기술적 차별화를 위해서는, 총체적이고 정확한 지식 기반을 확보해야 한다. 우리가 어떤 과정을 통해 안드로이드에 대한 총체적이고 정확한 지식 기반에 근접하게 된다면, 현재 우리가 모르고 있는 것이 무엇이며, 그것을 어떻게 공부해야 하고, 궁극적으로는 경쟁력 있는 기술에 한 걸음 앞서갈 수 있으리라 믿는다.

이 책은, 이런 것을 제공해 줄 것이다. 이 책의 제목은 "안드로이드 베이직 & 클래식, 안드로이드 입문서"이다. 하지만 결코 초보자를 위한 책은 아니다. 이 책은 기본에 충실한 책이다. 힘들더라도 꼼꼼히 읽어 나간다면, 모바일에 다시 온 아침, 안드로이드와 함께 푸르른 새아침을 맞이할 것으로 믿는다.

2009년 8월, 첫번째 번역서를 출판하고 나서 부족한 부분과 변경 사항에 대한 추가 반영을 위해 2009년 10월 15일 2nd Edition을 출판하였으며, 그 연장선에서 3rd Edition을 출판한다. 지금까지의 작업에 도움을 주신 모든 분들께 진심으로 감사합니다.

2010년 2월 22일

www.kandroid.org 운영자
양정수 (닉네임 들풀)

목 차

1. 안드로이드란 무엇인가? 14

1.1. 주요 특징 14
1.2. 안드로이드 아키텍처 15
1.3. 애플리케이션 15
1.4. 애플리케이션 프레임워크 15
1.5. 라이브러리 16
1.6. 안드로이드 런타임 17
1.7. 리눅스 커널 18

2. 애플리케이션 기초 20

2.1. 애플리케이션 컴포넌트 21
2.2. 액티비티와 태스크 30
2.3. 프로세스와 쓰레드 38
2.4. 컴포넌트 생명주기 43

3. 사용자 인터페이스 58

3.1. 개요 58
3.2. 레이아웃 선언하기 65
3.3. 메뉴 생성 73
3.4. 다이얼로그 생성 87
3.5. UI 이벤트 처리 102
3.6. 사용자에게 통보하기 109
3.7. 스타일과 테마 적용하기 126
3.8. 커스텀 컴포넌트 제작 131
3.9. 어댑터뷰를 사용한 데이터 바인딩 142
3.10. 일반적인 레이아웃 오브젝트 145
3.11. 안드로이드가 뷰를 그리는 방법 151

4. 리소스와 에셋 156
4.1. 리소스와 국제화 157
4.2. 사용가능한 리소스 타입 175

5. 인텐트와 인텐트 필터 200
5.1. 인텐트 오브젝트 201
5.2. 인텐트 확정 205

6. 데이터 저장공간 216
6.1. 프레퍼런스 217
6.2. 파일 218
6.3. 데이터베이스 218
6.4. 네트워크 220

7. 컨텐트 프로바이더 222
7.1. 컨텐트 프로바이더 기초 222
7.2. 컨텐트 프로바이더 쿼리하기 225
7.3. 프로바이더 데이터 수정하기 230
7.4. 컨텐트 프로바이더 생성 234
7.5. 컨텐트 URI 요약 239

8. 보안 및 퍼미션(Permission) 242
8.1. 보안 아키텍처 242
8.2. 애플리케이션 사인하기 243
8.3. 사용자 ID와 파일 접근 243
8.4. 퍼미션 사용하기 244
8.5. 퍼미션 선언 및 강제 245
8.6. URI 퍼미션 250

9. AndroidManifest.xml 파일 252

9.1. 매니페스트 파일 구조 253
9.2. 파일 규약 254
9.3. 파일 특성 257
9.4. 〈action〉 261
9.5. 〈activity〉 262
9.6. 〈activity-alias〉 274
9.7. 〈application〉 277
9.8. 〈category〉 282
9.9. 〈data〉 282
9.10. 〈grant-uri-permission〉 285
9.11. 〈instrumentation〉 287
9.12. 〈intent-filter〉 288
9.13. 〈manifest〉 291
9.14. 〈meta-data〉 293
9.15. 〈path-permission〉 295
9.16. 〈permission〉 296
9.17. 〈permission-group〉 299
9.18. 〈permission-tree〉 300
9.19. 〈provider〉 301
9.20. 〈receiver〉 307
9.21. 〈service〉 310
9.22. 〈supports-screens〉 313
9.23. 〈uses-configuration〉 315
9.24. 〈uses-feature〉 318
9.26. 〈uses-library〉 320
9.26. 〈uses-permission〉 321
9.27. 〈uses-sdk〉 322

10. 그래픽 326
10.1. 개요 326
10.2. 2차원 그래픽 331
10.3. OpenGL을 사용하는 3차원 그래픽 345

11. 오디오 및 비디오 348
11.1. 오디오 및 비디오 재생 348
11.2. 오디오 캡쳐 351
11.3. 안드로이드 지원 미디어 포맷 353

12. 로케이션과 맵 358
12.1. 로케이션 서비스 358
12.2. 구글 맵 외부 라이브러리 360

13. App 위젯 364
13.1. 기초 364
13.2. 매니페스트에 App 위젯 선언하기 365
13.3. AppWidgetProviderInfo 메타 데이터 추가하기 366
13.4. App 위젯 레이아웃 생성하기 368
13.5. AppWidgetProvider 클래스 사용하기 369
13.6. App 위젯 설정 액티비티 생성하기 372

14. 블루투스 394
14.1. 기초 394
14.2. 블루투스 퍼미션 395
14.3. 블루투스 설정(Setting Up) 396
14.4. 블루투스 검색(Finding) 398
14.5. 디바이스 연결 403
14.6. 커넥션 관리 410

15. AIDL을 사용하여 원격 인터페이스 디자인하기 416
15.1. AIDL을 사용하여 IPC 구현하기 416
15.2. IPC 메쏘드 호출하기 424

16. 애플리케이션 사인하기(Signing) 432
16.1. 개요 432
16.2. 사인 전략 434
16.3. 사인을 위한 기초 셋업 435
16.4. 디버그 모드에서 사인하기 436
16.5. 공식 릴리즈를 위해 사인하기 438
16.6. 개인 키 안전하게 하기 445

17. 애플리케이션 버전부여 448
17.1. 애플리케이션 버전 설정 449
17.2. 최소 시스템 API 버전 지정하기 450

18. 배포를 위한 준비 : 체크리스트 454
18.1. 애플리케이션 릴리즈 준비 이전 단계 455
18.2. 애플리케이션 최종 컴파일 이전 단계 457
18.3. 애플리케이션 컴파일 이후 단계 460

19. 애플리케이션 배포하기 462
19.1. 안드로이드 마켓에 배포하기 462

20. 성능을 위한 설계 468
20.1. 소개 469
20.2. 오브젝트 생성을 피하라 470
20.3. 네이티브(Native) 메쏘드를 사용하라 471
20.4. 인터페이스(Interface) 보다 버추얼(Virtual)을 선호하라 471
20.5. 버추얼(Virtual)보다 정적(Static)을 선호하라 472

20.6. 내부적인 Getters/Setters를 피하라 472
20.7. 캐쉬 필드 룩업(Lookup) 473
20.8. 상수를 Final로 선언하라 474
20.9. Enhanced For Loop 문법을 주의하여 사용하라 475
20.10. Enums을 피하라 476
20.11. 내부 클래스를 가지는 패지지 영역(Scope)을 사용하라 478
20.12. 부동소수점을 피하라 479
20.13. 약간의 샘플 성능 숫자들 480
20.14. 마무리 노트 481

21. 응답성을 위한 설계 484
21.1. 무엇이 ANR을 유발하는가? 485
21.2. 어떻게 ANR을 피할 것인가? 486
21.3. 응답성 강화하기 487

22. 무결절성을 위한 설계 490
22.1. 데이터를 누락하지 마라 492
22.2. 원시 데이터를 드러내지 마라 492
22.3. 사용자를 방해하지 마라 493
22.4. 해야 할 것이 많다면, 쓰레드에서 하라 493
22.5. 단일 액티비티 스크린에 오버로드를 주지 마라 494
22.6. 시스템 테마를 확장하라 495
22.7. UI가 다양한 스크린 해상도에서 작동하도록 디자인하라 438
22.8. 네트워크가 느리다는 것을 가정하라 495
22.9. 터치 스크린 또는 키보드를 가정하지 마라 496
22.10. 디바이스 배터리를 절약하라 496

22. 용어집 500

Section | 1

안드로이드란 무엇인가?

1.1. 주요 특징
1.2. 안드로이드 아키텍처
1.3. 애플리케이션
1.4. 애플리케이션 프레임워크
1.5. 라이브러리
1.6. 안드로이드 런타임
1.7. 리눅스 커널

1. 안드로이드란 무엇인가?

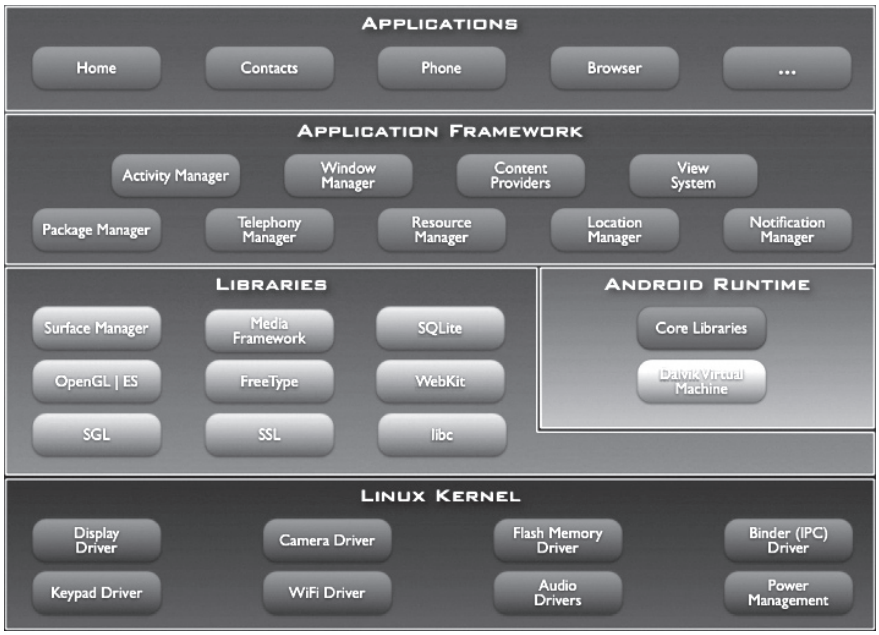
안드로이드는 운영체제와 미들웨어 그리고 핵심 애플리케이션을 포함하고 있는 모바일 디바이스를 위한 소프트웨어 스택Stack이다. 안드로이드 SDK는 Java 프로그래밍 언어를 사용하여 안드로이드 플랫폼 상의 애플리케이션을 개발하기 위해 필요한 도구들과 API를 제공한다.

주요 특징

- **애플리케이션 프레임워크** 컴포넌트에 대한 재사용 및 치환replacement를 가능하게 함.
- **달빅(Dalvik) 가상머신** 모바일 디바이스를 위해 최적화됨.
- **통합 브라우저** 오픈 소스 웹킷WebKit 엔진 기반.
- **최적화된 그래픽** 자체 제작된 2D 그래픽 라이브러리와 OpenGL ES 1.0 스펙(하드웨어 가속은 선택사항임)에 기반의 3D 그래픽에 의해 강화됨
- **SQLite** 정형화된 데이터 저장공간을 위한 것.
- **미디어 지원** 일반적인 오디오, 비디오 그리고 정지still 이미지 포맷들을 지원(MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM 테크놀로지** (하드웨어 의존적)
- **블루투스, EDGE, 3G와 WiFi** (하드웨어 의존적)
- **카메라, GPS, 나침반과 가속도계** (하드웨어 의존적)
- **풍부한 개발 환경** 디바이스 에뮬레이터, 디버깅 툴, 메모리 및 성능 프로파일링 그리고 이클립스 통합개발환경IDE을 위한 플러그인 포함.

안드로이드 아키텍처

아래의 다이어그램은 안드로이드 운영체제의 주요 구성 요소를 보여준다. 각 영역은 아래에 더 자세히 설명된다.



애플리케이션

안드로이드는 이메일 클라이언트, SMS 문자메시지 프로그램, 캘린더, 맵, 브라우저, 컨택contact 등과 같은 핵심 애플리케이션을 탑재하고 있다. 모든 애플리케이션은 Java 프로그래밍 언어를 사용하여 작성되었다.

애플리케이션 프레임워크

개발자는 안드로이드에 이미 탑재된 핵심 애플리케이션에 사용된 것과 동일한 프레임워크 API에 완벽하게 접근할 수 있다(역주: 그럼에도 불구하고 안드로이드 SDK와 함께 배포되고 있는 android.jar와 안드로이드 런타임에 존재하는 core library는 정확히 일치하지 않은다). 애플리케이션의 아키텍처는 컴포넌트의 재사용을 쉽게 하도록 디자인되었다. 애플리케

이션은 자신의 기능을 외부 애플리케이션에게 제공할 수 있으며, 다른 애플리케이션이 제공하는 기능을 사용할 수도 있다(단, 프레임워크에 의해 관리되는 보안 제약에 기초함). 이 같은 메커니즘은 사용자에 의한 컴포넌트의 치환replacement을 가능하게 한다. 모든 애플리케이션 하단에는 다음과 같은 서비스와 시스템의 집합이 존재한다.

- 애플리케이션을 만드는 데 사용될 수 있는 풍부하고 확장가능한 뷰 집합, 이것은 리스트list, 그리드grid, 텍스트 박스text box, 버튼button, 심지어 임베딩 가능한 웹 브라우저까지도 포함하고 있다.
- (Contacts과 같이) 애플리케이션이 다른 애플리케이션의 데이터에 접근하는 것을 가능하게 하거나, 또는 자신의 데이터를 공유하는 것을 가능하게 하는 컨텐트 프로바이더Content Provider.
- 리소스Resource 매니저; 로컬라이즈된localized 문자열, 그래픽, 그리고 레이아웃 파일 같은 비 코드non-code 리소스에 대한 접근을 제공하는 리소스 매니저Resource Manager.
- 모든 애플리케이션이 상태 바status bar에 커스텀 알림 메시지를 표시하는 것을 가능하게 하는 노티피케이션 매니저Notification Manager.
- 애플리케이션의 생명주기lifecycle를 관리하며, 사용자의 일반적인 애플리케이션 네비게이션 히스토리를 관리(단, back stack, 즉 뒤로 돌아가기 기능만 지원함)하는 액티비티 매니저Activity Manager.

애플리케이션에 대한 보다 자세한 설명은 SDK와 함께 제공되는 노트패드Notepad 예제를 참고하기 바란다.

라이브러리

안드로이드는 다양한 안드로이드 시스템 컴포넌트에서 사용되는 C/C++ 라이브러리들을 포함하고 있다. 이런 기능은 안드로이드 애플리케이션 프레임워크를 통해 개발자에게 제공된다. 아래에 몇몇 핵심 라이브러리가 나열되어 있다.

- **시스템 C 라이브러리** – 임베디드 리눅스 기반의 디바이스에 적합하도록 만들어진 표준 C 시스템 라이브러리로써 BSD libc로부터 파생되어 구현되었음.

- **미디어 라이브러리** – PacketVideo의 OpenCORE 기반임; MPEG4, H.264, MP3, AAC, AMR, JPG, 그리고 PNG를 포함하는 정지still 이미지 파일과 널리 사용되는 오디오와 비디오 포맷의 재생과 녹화를 지원한다.
- **서피스(Surface) 매니저** – 디스플레이 서브시스템subsystem에 대한 접근 관리 및 다양한 애플리케이션에 의해 사용되는 2D 및 3D 그래픽 레이어들을 단일하게 관리함.
- **LibWebCore** – 안드로이드 브라우저와 임베딩 가능한 웹 뷰, 두 가지 모두를 제공하는 최신 웹 브라우저 엔진
- **SGL** – 2D 그래픽스 엔진의 하단을 구성함.
- **3D 라이브러리** – OpenGL ES 1.0 API 기반으로 구현되었음. 이 라이브러리는 하드웨어 3D 가속기(단, 그것이 사용가능하다면)를 사용하거나, 또는 보유하고 있는 최적화된 3D 소프트웨어 래스터라이저를 사용함.
- **FreeType** – 비트 맵 및 벡터 폰트 렌더링 엔진
- **SQLite** – 모든 애플리케이션에서 이용할 수 있는 강력하며 경량화lightweight된 관계형 데이터 베이스 엔진

안드로이드 런타임

안드로이드의 핵심 라이브러리core library는 Java 프로그래밍 언어의 핵심 라이브러리에서 사용가능한 대부분의 기능을 포함하고 있다.

모든 안드로이드 애플리케이션은 달빅Dalvik 가상 머신에 대한 자기 자신의 인스턴스를 가지고, 자신의 프로세스 내에서 실행된다. 달빅은 하나의 디바이스에서 다양한 가상머신이 효율적으로 실행될 수 있도록 만들어졌다. 달빅 가상머신은 최소 메모리를 사용footprint하도록 최적화된 달빅 실행executable 포맷(.dex)의 파일을 실행한다. 이 가상머신은 레지스터 기반이며, Java 언어 컴파일러에 의해 컴파일된 후, SDK에 포함되어 있는 "dx"라는 툴tool을 통해 .dex 포맷으로 변환된 클래스를 실행한다.

달빅 가상머신은 쓰레딩threading과 저 수준low-level 메모리 관리와 같이 그것의

기반이 되는 기능에 대해서는 리눅스 커널에 의존한다.

리눅스 커널(Kernel)

안드로이드는 보안, 메모리 관리, 프로세스 관리, 네트워크 스택, 그리고 드라이버 모델과 같은 핵심 시스템 서비스에 대해서는 리눅스 버전 2.6에 의존한다. 또한 리눅스 커널은 하드웨어와 안드로이드 플랫폼 스택의 나머지 계층 간의 추상화된 계층 역할을 담당한다.

Section | 2

애플리케이션 기초

2.1. 애플리케이션 컴포넌트

2.2. 액티비티와 태스크

2.3. 프로세스와 쓰레드

2.4. 컴포넌트 생명주기

2. 애플리케이션 기초

안드로이드 애플리케이션은 Java 프로그래밍 언어로 작성된다. 컴파일된 Java 코드 – 애플리케이션에 필요한 임의의 데이터와 리소스 파일과 함께 – 는 SDK에 존재하는 aapt 유틸리티에 의해 안드로이드 패키지로 묶여지며, 이렇게 하나로 압축된 아카이브archive 파일에는 .apk 접미사가 부쳐진다. 이 파일은 모바일 디바이스에 안드로이드 애플리케이션을 배포하고 설치하기 위한 수단이다(사용자들이 자신의 모바일 디바이스에 다운로드하게 되는 파일이 바로 이 파일이다). 하나의 .apk 파일 안에 있는 모든 코드는 하나의 애플리케이션으로 간주된다.
다양한 방법으로 각각의 안드로이드 애플리케이션은 그것 자신의 세계 내에서in its own world 존속한다.

- 디폴트로 모든 애플리케이션은 자신의 리눅스 프로세스 내에서 실행된다. 안드로이드는 임의의 애플리케이션 코드가 실행될 필요가 있을 때 그 프로세스를 시작시키며, 그것이 더 이상 필요치 않고 다른 애플리케이션이 시스템 자원을 요구할 때 그 프로세스를 종료시킨다shuts down.
- 각각의 프로세스는 자기 자신의 Java 가상머신을 가진다. 그러므로 애플리케이션 코드는 다른 모든 애플리케이션 코드와는 격리되어 실행된다.
- 디폴트로 각각의 애플리케이션에는 고유한 리눅스 유저user ID가 부여된다. 애플리케이션을 구성하는 파일들은 해당 사용자와 해당 애플리케이션 그 자신에게만 보여지도록 퍼미션permission이 설정된다 – 비록, 다른 애플리케이션에게도 그 파일을 보여줄 방법이 있기는 하지만.

두 개의 애플리케이션에 대해 동일한 유저user ID를 공유할 수 있도록 배치하는 것이 가능하다. 그런 경우에 두 개의 애플리케이션은 각자 다른 애플리케이션의 파일을 볼 수 있을 것이다. 시스템 자원을 절약하기 위해 동일한 ID를 가지는 애플리케이션은 또한 동일한 리눅스 프로세스 안에서 실행되도록 배치되며, 동일한 가상머신을 공유한다.

애플리케이션 컴포넌트

안드로이드의 중심 특징은 하나의 애플리케이션이 다른 애플리케이션의 엘리먼트를(단, 그 엘리먼트의 애플리케이션이 사용을 허가하는 조건하에서) 사용할 수 있다는 것이다. 예를 들어 만약 여러분의 애플리케이션이 이미지 스크롤 리스트를 표시할 필요가 있고, 다른 애플리케이션이 그에 알맞은 스크롤러scroller를 개발했고 또한 그것을 다른 애플리케이션이 사용할 수 있도록 만들어 놓았다면, 여러분은 자신의 스크롤러를 별도로 개발하지 않고 이미 있는 스크롤러가 그 작업을 수행할 수 있도록 호출할 수 있다. 이를 위해, 여러분의 애플리케이션은 다른 애플리케이션의 코드를 통합하거나 링크하지는 않는다. 오히려 그것이 필요할 때 다른 애플리케이션 내에 있는 해당 영역piece을 간단하게 구동start-up시킬 수 있다.

이런 작업을 위해서는, 어떤 애플리케이션의 특정 영역이 필요해 질 때 시스템은 애플리케이션 프로세스를 시작할 수 있어야 하며, 그리고 그 영역에 대한 Java 오브젝트를 인스턴스화 할 수 있어야 한다. 그러므로 다른 대부분 시스템에서의 애플리케이션과는 달리, 안드로이드는 애플리케이션 안의 모든 것들에 대한 하나의 시작점entry point을 가지지 않는다(예를 들어 main() 함수가 없음). 대신에, 안드로이드 애플리케이션에는 시스템이 그것을 필요로 할 때 인스턴스화하고 실행할 수 있는 기본적인 컴포넌트들이 존재한다. 안드로이드 애플리케이션에는 네 가지 유형의 컴포넌트가 존재한다.

액티비티(Activity)

액티비티는 사용자가 착수할 수 있는 하나의 포커스된 시도에 대한 비쥬얼visual 사용자 인터페이스를 나타낸다. 예를 들어 액티비티는 사용자들이 선택할 수 있는 메뉴 아이템 리스트를 나타내거나, 사진과 함께 그것의 캡션caption을 나타낼 수도 있다. 텍스트 메시징 애플리케이션은, 메시지를 보내기 위한 컨택contact 리스트를 보여주는 하나의 액티비티와 선택된 컨택contact에 대해 메시지를 작성하는 두 번째 액티비티, 그리고 오래된 메시지들을 다시 보거나 설정을 바꾸기 위한 나머지 액티비티를 가질 수 있다. 이들 액티비티들은 하나의 결합된 사용자 인터페이스를 형성하기 위해 함께 동작함에도 불구하고, 각각의 액티비티는 다른 액티비티와 독립되어 있다. 각각의 액티비티는

Activity 베이스 클래스의 서브클래스로 구현된다.

하나의 애플리케이션은 단지 한 개의 액티비티로 구성될 수도 있으며, 또한 앞서 간단히 언급했던 텍스트 메시징 애플리케이션처럼 몇 개의 액티비티를 포함하고 있을 수도 있다. 당연히 애플리케이션을 구성하는 액티비티가 무엇인지 그리고 얼마나 많은 액티비티가 그곳에 있는 지는, 그 애플리케이션과 그것의 디자인에 의존적이다. 일반적으로 액티비티들 중 하나에는, 그 애플리케이션이 시작됐을 때 사용자에게 보여질 수 있도록 첫 번째 액티비티라는 표시가 붙게 된다. 하나의 액티비티에서 또 다른 액티비티로의 이동은, 현재 액티비티가 다음번 액티비티를 시작함으로써 이루어진다.

각각의 액티비티에는 그리기draw가 가능한 하나의 디폴트 윈도우가 부여된다. 일반적으로 윈도우는 스크린 전체를 채우지만 스크린보다 작을 수도 있고, 다른 윈도우 그 위에 떠있을 수도 있다. 하나의 액티비티는 또한 추가적인 윈도우를 사용할 수 있다 – 예를 들어 액티비티의 한복판에서 사용자 응답을 요청하는 팝업 다이얼로그dialog라든지, 또는 스크린 상에서 특정 아이템을 선택할 때 사용자에게 필수적인 정보를 표시하는 윈도우가 그것이다.

윈도우 내의 비쥬얼visual 컨텐트는 – 베이스 뷰 클래스로부터 파생된 오브젝트들인 – 뷰View들의 계층구조hierarchy에 의해 만들어진다. 각각의 뷰는 윈도우 내의 하나의 특정 직사각형 영역을 제어한다. 부모Parent 뷰은 그것들의 자식들children의 레이아웃을 포함하고 구조화한다. 계층구조의 최하단의 뷰는 그것들이 제어하는 직사각형 영역을 그리며, 그 영역에 국한된 사용자 액션에 대한 책임을 진다. 이와 같이 뷰는 액티비티와 사용자 간의 상호작용이 이루어지는 영역이다. 예를 들어 하나의 뷰는 작은 이미지를 보여줄 수도 있고, 사용자가 그 이미지를 만질 때 특정 액션을 일으킬 수도 있다. 안드로이드는 여러분이 사용할 수 있는 – 버튼, 텍스트 필드, 스크롤 바, 메뉴 아이템, 체크 박스, 그리고 추가적인 것들을 포함해서 – 이미 만들어진 많은 뷰를 가지고 있다.

뷰 계층구조는 Activity.setContentView() 메쏘드에 의해 액티비티의 윈도우 내에 위치하게 된다. 그 컨텐트 뷰는 그 계층구조의 최상위에 있는 뷰view 오브젝트이다(뷰와 계층구조에 대한 더 많은 정보에 대해서는, 3장. "사용자 인터페이스"를 보라).

서비스(Service)

서비스는 비쥬얼한 사용자 인터페이스를 가지지 않는다. 하지만 오히려 정해지지 않은 시간동안 백그라운드background에서 실행된다. 예를 들어 서비스는 사용자가 다른 일에 주의를 기울이는 동안에 백그라운드 음악을 재생할 수 있다. 또한 그것은 네트워크 상에서 데이터를 가져오거나, 또는 어떤 것을 계산해서 그 결과를 필요로 하는 액티비티에게 제공할 수도 있다. 각각의 서비스는 Service 베이스 클래스를 확장extend한다.

이것의 기본적인 예제는 재생 목록에서 음악을 재생하는 미디어 플레이어이다. 아마도 그 플레이어 애플리케이션은 사용자에게 음악을 선택할 수 있게 하고 그것을 재생하는, 하나 또는 그 이상의 액티비티를 가지고 있을 수 있다. 하지만 음악 재생 그 자체는 액티비티에 의해 제어되어서는 안된다. 왜냐하면 사용자가 플레이어 화면을 벗어난 후, 다른 어떤 것을 실행한 이후에도 그 음악이 계속 재생되길 기대할 것이기 때문이다. 음악이 계속 재생되게 하기 위해, 미디어 플레이어 액티비티는 백그라운드에서 실행되는 하나의 서비스를 시작할 수 있다. 시스템은 심지어 그 서비스를 시작했던 액티비티가 스크린에서 사라진 후 조차도 음악 재생 서비스를 계속 유지할 것이다.

실행되고 있는 서비스에 접속connect하는 것(바인드하는 것)이 (그리고 만약 아직 실행되지 않았다면, 그 서비스를 실행하는 것이) 가능하다. 그것에 접속되어 있는 동안, 여러분은 그 서비스가 제공하는 인터페이스를 통해 서비스와 커뮤니케이션할 수 있다. 음악 서비스의 경우, 이러한 인터페이스는 사용자에게 음악을 일시 멈추고, 되감고, 정지시키고, 다시 재생하는 것을 가능하게 한다.

액티비티와 다른 컴포넌트 처럼, 서비스도 애플리케이션 프로세스의 메인 쓰레드 내에서 실행된다. 그러므로 서비스가 다른 컴포넌트 또는 사용자 인터페이스에 의해 방해받지 않도록 하기 위해서 음악 재생같이 시간을 요하는 작업들을 위해서는 종종 또 다른 쓰레드를 만들어서 작동시킨다. 이 부분에 대해서는 추후 언급되는 “프로세스와 쓰레드”를 보라.

브로드캐스트 리시버(Broadcast receiver)

브로드캐스트 리시버는 아무 것도 하지 않는 컴포넌트지만, 브로드캐스트 공지announcement를 수신하고 응답한다. 많은 브로드캐스트는 시스템 코드에서 발생된다 – 예를 들어 시간대가 바뀌었다, 배터리가 부족하다, 사진이 찍혔다, 사용자가 언어 설정을 바꾸었다 등이 그것이다. 애플리케이션 또한 브로드캐스트를 발생시킬 수 있다 – 예를 들어 다른 애플리케이션에게 어떤 데이터가 디바이스에 다운로드되었고, 그것들이 그것을 사용하는 것이 가능하다는 것을 알게 할 수 있다.

애플리케이션은 중요하다고 여겨지는 임의의 공지에 응답할 수 있는 임의의 수의 브로드캐스트 리시버를 가질 수 있다. 모든 리시버는 Broadcast Receiver 베이스 클래스를 확장extend한다.

브로드캐스트 리시버는 사용자 인터페이스를 보여주지 않는다. 하지만 그것이 수신한 정보에 응답하는 액티비티를 시작하거나, 또는 사용자에게 알려주기 위해 노티피케이션 매니저notification manager를 사용할 수 있다. 노티피케이션은 여러가지 방법으로 사용자의 주의를 끌 수 있다 – 배경조명 발광, 디바이스 진동, 사운드 재생 등이 있다. 그것은 일반적으로 상태바status bar 내에 아이콘을 상주시켜 놓으며, 사용자는 그 메시지를 얻기 위해 그것을 오픈할 수 있다.

컨텐트 프로바이더(Content provider)

컨텐트 프로바이더는 다른 애플리케이션에게 유용한 특정 애플리케이션의 데이터 집합set을 만든다. 그 데이터는 파일 시스템 내, SQLite 데이터베이스 내, 또는 의미있는 다른 어떤 방식으로 저장될 수도 있다. 컨텐트 프로바이더는 자신이 제어하는 타입의 데이터를 다른 애플리케이션이 얻고 저장할 수 있도록 하게 하는 표준 메쏘드 집합set을 구현하기 위해, ContentProvider 베이스 클래스를 확장extend한다. 하지만 애플리케이션은 이러한 메쏘드를 직접적으로 호출하지 않는다. 대신 애플리케이션은 컨텐트 리졸버ContentResolver 오브젝트를 사용하여 그것의 메쏘드를 호출한다. 컨텐트 리졸버는 어떤 컨텐트 프로바이더와도 대화할 수 있으며, 그것과 관련된 임의의 프로세스 간 통신(IPC, interprocess communication)을 관리하기 위해 프로바이더와 협력한다. 컨텐

트 프로바이더 사용법에 관한 더 많은 정보는 7장. 컨텐트 프로바이더를 보라.

특정 컴포넌트에 의해 처리되어야 하는 요청이 있을 때마다, 안드로이드는 그 컴포넌트의 애플리케이션 프로세스가 실행running되도록 하며, 필요하다면 그것을 시작start하기도 한다. 그리고 그 컴포넌트에 맞는 인스턴스가 사용가능해지도록 하며, 필요하다면 그 인스턴스를 생성도 한다.

컴포넌트 활성화하기(Activating) : 인텐트(Intent)

컨텐트 프로바이더content provider는 컨텐트 리졸버content resolver 요청의 대상target이 될 때 활성화된다. 하지만 다른 세 가지 컴포넌트, 즉 액티비티, 서비스, 그리고 브로드캐스트 리시버는 인텐트intent라 불리는 비동기적인 메시지에 의해 활성화된다. 인텐트는 컨텐트 메시지를 보유하고 있는 인텐트Intent 오브젝트다. 인텐트는 액티비티와 서비스에 대해서, 그것에게 요청되는requested 액션을 가리키고 그 액션이 처리해야 하는 데이터의 URI를 나머지 다른 것들과 함께 명시한다. 예를 들어 인텐트는 어떤 액티비티가 사용자에게 이미지를 보여주도록 하거나, 또는 사용자가 어떤 텍스트를 편집할 수 있도록 하는 요청을 전달할 수 있다. 인텐트 오브젝트는 브로드캐스트 리시버에 대해서는, 공지되는announced 액션을 가리킨다. 예를 들어 그것은 카메라 버튼이 눌려졌다는 사실에 관심을 갖는 관계자party에게 공지될 수 있다.

각각의 컴포넌트 타입을 활성화하기 위한 별도의 메쏘드가 있다.

- 액티비티는 인텐트 오브젝트를 Context.startActivity() 또는 Activity.startActivityForResult()에 파라미터로 전달함으로써 런치된다(또는 새로 처리할 어떤 것이 주어진다). 이것에 응답하는 액티비티는 getIntent() 메쏘드를 호출함으로써, 그것을 런치하도록 한 초기 인텐트를 볼 수 있다. 안드로이드는 그 액티비티에게 그 이후 이어지는 임의의 인텐트를 전달하기 위해서는, 그 액티비티의 onNewIntent() 메쏘드를 호출한다.

  하나의 액티비티는 흔히 그 다음번 액티비티를 시작시킨다. 만약 그것이 자신이 시작시킨 액티비티로부터 결과를 리턴받고자 한다면, 그것은 startActivity() 대신에 startActivityForResult()를 호출해야 한다. 예를 들어 어떤 액티비티가 사용자에게 사진을 선택하게 하는 액티비티를 시작시킨다면,

그 액티비티는 사용자가 선택한 사진을 리턴받기를 원할 수 있다. 그 결과는, 액티비티의 onActivityResult() 메쏘드가 호출되는 곳에 전달되는 인텐트 오브젝트 내에서 리턴된다.

- 서비스는 Context.startService()에 인텐트 오브젝트를 파라미터로 전달함으로써 시작된다(또는 이미 실행중인 서비스에게 새로운 명령이 주어진다). 안드로이드는 서비스의 onStart() 메쏘드를 호출하고 그곳에 인텐트 오브젝트를 전달한다.

  유사한 방식으로, 인텐트는 호출하는 컴포넌트와 타겟target 서비스 간의 지속적인 연결을 확립하기 위해 Context.bindService()에 파라미터로 전달될 수도 있다. 그리고 해당 서비스는 onBind()가 호출되는 곳에서 그 인텐트 오브젝트를 전달받는다(만약 서비스가 아직 실행중이 아니라면, bindService()는 선택적으로 서비스를 시작시킬 수 있다). 예를 들어 액티비티는 앞서 언급했듯이 음악 재생을 제어하는 수단(즉, 사용자 인터페이스)을 사용자에게 제공하기 위해 음악 재생 서비스와의 연결을 확립할 수 있다. 액티비티는 그러한 커넥션을 셋업하기 위해 bindService()를 호출할 것이며, 그런 다음 음악 재생에 관여하기 위해 서비스에 정의된 메쏘드를 호출하게 된다.

  나중에 나오는 원격 프로시저 호출 섹션은 서비스에 바인딩하는 것에 대해 좀 더 상세히 설명한다.

- 애플리케이션은 Context.sendBroadcast(), Context.sendOrderedBroadcast(), 그리고 Context.sendStickyBroadcast() 같이 다양하게 변형된 메쏘드에 인텐트 오브젝트를 전달함으로써 브로드캐스트broadcast를 만들어낼 수 있다. 안드로이드는 이것에 관심을 갖는 모든 브로드캐스트 리시버에게 그들의 onReceive() 메쏘드를 호출해 줌으로써 그 인텐트를 전달한다.

인텐트 메시지에 대한 보다 많은 정보는 5장. “인텐트와 인텐트 필터”를 보라.

컴포넌트 종료하기(Shutting down)

컨텐트 프로바이더는 컨텐트 리졸버의 요청에 응답하는 동안에만 활성화되어 있다. 그리고 브로드캐스트 리시버도 브로드캐스트 메시지에 응답하는 동안에만 활성화되어 있다. 그러므로 이러한 컴포넌트는 명시적으로 종료시켜야 할 필요가 없다.

반면, 액티비티는 사용자 인터페이스를 제공한다. 이것은 사용자와 오랜시간 동

안 대화를 하며 활성화 상태로 남아 있다. 심지어는 대화가 지속될 수 있는 한, 그것이 사용되지 않는idle 상태에도 활성화 상태로 남는다. 유사하게, 서비스 또한 오랜 시간동안 실행 상태로 남아 있다. 그래서 안드로이드는 어떤 규칙적인 방식으로 액티비티와 서비스를 종료시키기 위한 메쏘드를 가지고 있다.

- 액티비티는 finish() 메쏘드의 호출을 통해 스스로 종료될 수 있다. 그리고 finishActivity()를 호출함으로써, 다른 액티비티(startActivityForResult()를 사용해서 시작된 액티비티)를 종료시킬 수도 있다.
- 서비스는 stopSelf() 메쏘드의 호출이나 Context.stopService()를 호출함으로써 종료될 수 있다.

또한, 컴포넌트는 그것이 더 이상 사용되지 않을 때, 또는 안드로이드가 더 많은 활성화된 컴포넌트를 위해 메모리를 회수해야 할 때 시스템에 의해 종료될 수 있다. 이후에 나오는 "컴포넌트 생명주기" 섹션에서 그것이 일어나는 경우를 좀 더 상세히 논의할 것이다.

매니페스트(manifest) 파일

안드로이드는 애플리케이션의 컴포넌트를 시작하기 전에, 컴포넌트의 존재를 인식해야 한다. 그러므로 애플리케이션은 매니페스트 파일에 그것들의 컴포넌트를 선언한다. 그리고 그 매니페스트 파일은 안드로이드 패키지, .apk 파일 내에 포함된다. 이 패키지는 애플리케이션의 코드, 파일, 리소스 또한 보유하고 있다.

이 매니페스트는 구조화된 XML 파일이며, 모든 애플리케이션에서 AndroidManifest.xml이란 파일명으로 존재한다. 매니페스트는 애플리케이션 컴포넌트를 선언하는 것과 더불어, (디폴트 안드로이드 라이브러리와 별도로) 애플리케이션에서 링크되어야 할 필요가 있는 특정 라이브러리를 지정하거나 애플리케이션에게 부여되길 원하는 임의의 퍼미션permission을 식별하는 것과 같은 몇 가지 것들을 한다.

그러나 매니페스트의 주요한 작업은 안드로이드에게 애플리케이션의 컴포넌트를 알려주는 것이다. 예를 들어 액티비티는 다음과 같이 선언될 수 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

〈activity〉 엘리먼트의 name 애트리뷰트는 액티비티를 구현하는 Activity 서브클래스를 지칭한다. icon과 label 애트리뷰트는 액티비티를 나타내기 위해 사용자에게 보여주는 아이콘과 라벨을 담고있는 리소스 파일을 가리킨다.

나머지 다른 컴포넌트도 유사한 방법으로 선언된다 – 서비스는 〈service〉 엘리먼트로, 브로드캐스트 리시버는 〈receiver〉 엘리먼트로, 컨텐트 프로바이더는 〈provider〉 엘리먼트로 선언된다. 매니페스트에 선언되지 않은 액티비티, 서비스, 컨텐트 프로바이더는 시스템에게 보여지지 않으며, 결과적으로 결코 실행되지 않는다. 하지만 브로드캐스트 리시버는 매니페스트 내에서 선언되거나, 또는 (BroadcastReceiver 오브젝트 형태로) 코드 내에서 동적으로 만들어져서 Context.registerReceiver()를 호출함으로써 시스템에 등록될 수 있다.

애플리케이션의 매니페스트 파일을 구조화하는 방법에 대한 더 많은 정보는 9장. "AndroidManifest.xml"을 보라.

인텐트 필터(Intent filter)

인텐트 오브젝트는 그것의 대상이 되는 컴포넌트를 명시적으로 지정할 수 있다. 만약 그렇다면, 안드로이드는 (매니페스트 파일 내의 선언들에 기초해서) 그 컴포넌트를 찾아내고 그것을 활성화한다. 하지만 타겟target이 명시적으로 지정되지 않으면, 안드로이드는 인텐트를 처리할 수 있는 최선의 컴포넌트를 찾아내야 한다. 이것은 해당 인텐트 오브젝트와 그 인텐트의 잠재적 대상이 될 수 있는 컴포넌트들의 인텐트 필터를 비교함으로써 이루어진다. 컴포넌트의 인텐트 필터는 안드로이드에게 해당

컴포넌트가 처리할 수 있는 인텐트의 종류를 알려준다. 컴포넌트에 대한 다른 필수적인 정보들처럼, 인텐트 필터도 매니페스트 파일 내에 선언된다. 여기에 액티비티에 두 개의 인텐트 필터를 추가한, 앞선 예제의 확장된 형태가 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:type="image/jpeg" />
                <category
                    android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

예제의 첫 번째 필터, 즉 "android.intent.action.MAIN" 액션과 "android.intent.category.LAUNCHER" 카테고리로 구성된 필터는 일반적인 것이다. 이것은 디바이스에서 사용자가 실행할 수 있는 애플리케이션 리스트를 스크린에 보여주는 애플리케이션 런처launcher에 그 액티비티가 표시되게 한다. 달리 말하면, 그 액티비티가 애플리케이션의 시작점이고, 사용자가 런처에서 애플리케이션을 선택했을 때 그들이 처음으로 보게 될 액티비티라는 것을 의미한다.

두 번째 필터는 특정 타입의 데이터에 대해 액티비티가 수행할 수 있는 액션을 선언한다.

컴포넌트는 인텐트 필터를 얼마든지 가질 수 있다. 그리고 각각의 필터는 각기 다

른 능력capability을 선언할 수 있다. 만약 컴포넌트가 어떤 필터도 가지지 않는다면, 그것은 인텐트의 대상으로 컴포넌트를 명시적으로 지정하는 인텐트에 의해서만 활성화될 수 있다.

코드 내에서 생성되어 등록된 브로드캐스트 리시버에 있어서, 인텐트 필터는 IntentFilter 오브젝트에 의해 직접적으로 인스턴스화 된다. 나머지 모든 필터들은 매니페스트 내에서 설정된다.

인텐트 필터에 대한 보다 많은 것은 5장. "인텐트와 인텐트 필터"를 보라.

액티비티와 태스크

앞에서 언급했던 것처럼 하나의 액티비티는 또 다른 액티비티를 시작시킬 수 있다. 그렇게 시작시킬 수 있는 액티비티에는 다른 애플리케이션이 정의한 액티비티도 포함된다. 예를 들어 여러분이 사용자에게 특정 위치 상의 시가지에 대한 지도를 보여주고 싶다고 가정해 보자. 이미 안드로이드에는 그 위치를 보여줄 수 있는 액티비티가 존재한다. 그러므로 여러분의 액티비티에 필요한 모든 것은, 인텐트 오브젝트에 필요한 정보들을 추가한 후 그것을 startActivity()의 파라미터로 전달하는 것이다. 그러면 맵 뷰어는 그 지도를 보여줄 것이다. 그리고 사용자가 BACK 키를 누를 때 여러분의 액티비티는 화면에 다시 나타날 것이다.

맵 뷰어는 다른 애플리케이션 내에 정의되어 있고 다른 애플리케이션 프로세스 상에서 실행되고 있음에도 불구하고, 사용자에게 맵 뷰어는 여러분의 액티비티처럼 동일한 애플리케이션의 일부분처럼 비춰질 것이다. 안드로이드는 두 개의 액티비티를 같은 태스크내에 유지함으로써 이런 사용자 경험user experience을 유지시킨다. 간단히 말하면, 태스크는 하나의 "애플리케이션"처럼 사용자들이 경험하는 그 무엇인 것이다. 그것은 하나의 스택 내에 정렬된 관련 액티비티들의 그룹이다. 스택의 루트Root 액티비티는 그 태스크를 시작했던 액티비티이다 – 일반적으로, 그것은 사용자가 애플리케이션 런처에서 선택했던 액티비티이다. 스택의 최상위 액티비티는 현재 실행중인 액티비티이다 – 그것은 사용자 액션을 위해 포커스 된 액티비티이다. 하나의 액티비티가 다른 것을 시작할 때 새로운 액티비티는 스택에 푸쉬(push)되며, 실행중인 상태의 액티비티가 된다. 이전 액티비티는 여전히 스택

에 남아있다. 사용자가 BACK 키를 누를 때 현재의 액티비티는 스택으로부터 팝(pop)되며, 이전의 액티비티는 실행중인 액티비티로 재개resume된다.

스택은 오브젝트들을 포함하고 있다. 그러므로 태스크가 – 예를 들어 여러 개의 맵 뷰어와 같이 – 동일 액티비티 서브클래스의 인스턴스를 한 개 이상 가지고 있다면, 스택은 각각의 인스턴스에 대한 별도의 시작점을 가진다. 스택에서 액티비티는 결코 재정렬되지 않는다. 다만 푸쉬push되고 팝pop될 뿐이다.

태스크는 매니페스트 파일에 있는 클래스나 엘리먼트가 아니라, 액티비티에 대한 스택이다. 그러므로 태스크의 액티비티와 독립적으로 그 값을 설정할 방법은 없다. 태스크에 대한 값은 총괄적으로 루트 액티비티에 설정된다. 예를 들어 다음 섹션에서 태스크의 어피니티affinity에 대해 이야기할 것인데, 태스크의 어피니티affinity 값은 태스크의 루트 액티비티의 어피니티affinity 설정으로 나타내어 진다.

태스크 안에서 모든 액티비티는 하나의 단위로 함께 움직인다. 전체 태스크(전체 액티비티 스택)는 포어그라운드foreground로 가져와 질 수 있으며, 또한 백그라운드background로 보내질 수도 있다. 예를 들어 현재의 태스크가 자신의 스택에 네 개의 액티비티(세 개의 액티비티는 현재 액티비티 아래 있음)를 가지고 있다고 가정해 보자. 사용자가 HOME 키를 누르고 애플리케이션 런처로 간다. 그리고 새로운 애플리케이션(실제로 새로운 태스크)을 선택한다. 현재 태스크는 백그라운드로 가고, 새로운 태스크에 대한 루트 액티비티가 보여진다. 그런 다음 잠시 후, 사용자가 홈 스크린으로 돌아가서 이전 애플리케이션(이전 태스크)을 다시 선택한다. 스택에 모두 네개의 액티비티를 가지고 있는 그 태스크가 앞으로 온다. 그곳에서 사용자가 BACK 키를 누르면, 스크린에는 사용자가 조금 전에 떠났던 이전 태스크의 루트 액티비티가 보여지지는 않는다. 대신에 스택의 최상위의 액티비티가 제거되고 같은 태스크 내의 이전 액티비티가 보여진다.

방금 기술되었던 동작방식은 액티비티와 태스크에 대한 디폴트 동작방식이다. 그러나 이러한 동작방식 대부분의 측면을 수정하는 방법이 존재한다. 액티비티와 태스크 간의 결합과 태스크 내에서의 액티비티의 동작방식은 매니페스트 내의 액티비티의 〈activity〉 엘리먼트에 설정된 애트리뷰트와 액티비티를 시작시켰던 인텐트 오브젝트 내에 설정된 플래그 간의 상호작용에 의해 제어된다. 인텐트를 요청

하는 쪽과 그것에 응답하는 쪽 모두, 일어나는 것에 참견할 권리를 갖는다.
이 점에 있어서 주요 인텐트 플래그는 다음과 같다.

```
FLAG_ACTIVITY_NEW_TASK
FLAG_ACTIVITY_CLEAR_TOP
FLAG_ACTIVITY_RESET_TASK_IF_NEEDED
FLAG_ACTIVITY_SINGLE_TOP
```

주요 〈activity〉 애트리뷰트는 다음과 같다.

```
taskAffinity
launchMode
allowTaskReparenting
clearTaskOnLaunch
alwaysRetainTaskState
finishOnTaskLaunch
```

다음 섹션은 이 플래그와 애트리뷰트가 무엇을 하는지, 어떻게 상호작용하며, 무엇을 고려해서 이것의 사용을 결정해야 하는지에 대해 설명한다.

어피니티(Affinity, 친화력)와 신규 태스크

디폴트로 애플리케이션 내의 모든 액티비티는 서로에 대해 하나의 어피니티affinity를 갖는다. 즉, 동일 태스크에 속하는 그것들 모두에 대해 하나의 설정preference이 존재한다. 하지만 각각의 액티비티에 대한 개별적인 어피니티affinity가 〈activity〉 엘리먼트의 taskAffinity 애트리뷰트를 통해 설정될 수 있다. 다른 애플리케이션들내에 정의된 액티비티들이 하나의 어피니티affinity를 공유할 수도 있으며, 또한 같은 애플리케이션 내에 정의된 액티비티들에게 다른 어피니티affinity가 지정될 수도 있다. 어피니티affinity는 두 가지의 조건하에서 동작하게 된다. 액티비티를 런치하는 인텐트 오브젝트가 FLAG_ACTIVITY_NEW_TASK 플래그를 포함하고 있고, 그리고 액티비티에 allowTaskReparenting 애트리뷰트 값이 “true”로 설정되어 있을 때이다.

FLAG_ACTIVITY_NEW_TASK 플래그

앞에서 설명했듯이, 새로운 액티비티는 디폴트로 startActivity()를 호출한 액티비티의 태스크 안에 런치된다. 액티비티는 호출자caller와 같은 스택에 푸쉬push된다. 하지만 만약 인텐트 오브젝트에 FLAG_ACTIVITY_NEW_

TASK 플래그가 포함되어 startActivity()의 파라미터로 전달된다면, 시스템은 새로운 액티비티를 담기 위해 다른 태스크를 찾는다. 대개 그것은 플래그의 이름이 나타내듯이 신규 태스크가 된다. 하지만 꼭 그렇게 되어야만 하는 것은 아니다. 만약 신규 액티비티와 동일 어피니티affinity를 가지는 태스크가 이미 존재한다면, 그 액티비티는 그 태스크 내에서 런치된다. 만약 그렇지 않다면 신규 태스크가 런치된다.

allowTaskReparenting 애트리뷰트

만약 액티비티의 allowTaskReparenting 애트리뷰트가 "true"로 설정되어 있다면, 액티비티는 그것이 시작된 태스크로부터 어피니티affinity를 갖는 태스크로, 그 태스크가 앞으로 오게 될 때 이동할 수 있다. 예를 들어 선택해 놓은 도시의 날씨 상황을 예보하는 어떤 여행 애플리케이션의 일부를 정의하는 액티비티가 있다고 가정하자. 그것은 동일 애플리케이션 내의 다른 액티비티들과 같은 어피니티(디폴트 어피니티)를 가지고 있고, 리페어런팅(reparenting, 부모 태스크를 변경하는 것)이 허용되고 있다. 만약 여러분의 액티비티들 중의 하나가 일기 예보 액티비티를 시작하면, 그것은 여러분의 액티비티와 같은 태스크에 처음에는 속한다. 하지만 나중에 여행 애플리케이션이 앞으로 오게 되면, 그 일기 예보 액티비티는 다른 태스크에 재지정 될 것이고, 그 태스크에서 보여지게 된다.

만일 .apk 파일이 사용자 관점에서 하나 이상의 "애플리케이션"을 포함하고 있다면, 여러분은 아마도 그것들 각각과 연관된 액티비티들에 대해 다른 어피니티affinity를 지정하고자 할 것이다.

런치(Launch) 모드

네 개의 다른 런치 모드가 〈activity〉 엘리먼트의 launchMode 애트리뷰트에 지정될 수 있다.

```
"standard" (디폴트 모드)
"singleTop"
"singleTask"
"singleInstance"
```

위의 모드mode들은 네 가지 점에서 서로 차이가 있다.

- **어떤 태스크가 인텐트에 응답하는 액티비티를 보유할 것인가?** "standard"와 "singleTop" 모드에 대해서 인텐트 오브젝트가 FLAG_ACTIVITY_NEW_TASK 플래그를 가지지 않았다면, 그것은 인텐트를 만들었던(그리고 startActivity()를 호출했던) 액티비티가 있는 태스크이다. FLAG_ACTIVITY_NEW_TASK 플래그를 인텐트 오브젝트가 포함하고 있는 경우라면, 이전 섹션의 "어피니티와 신규 태스크"에서 설명된 것처럼 다른 태스크가 선택된다.

  반면, "singleTask"와 "singleInstance" 모드는 액티비티가 항상 태스크의 루트에 나타나게 한다. 그것은 태스크를 정의한다; 그것은 결코 다른 태스크 안으로 런치되지 않는다.

- **액티비티에 대한 인스턴스가 여러 개가 존재할 수 있는가?** "standard" 또는 "singleTop" 액티비티는 여러 번 인스턴스화될 수 있다. 그들은 다양한 태스크에 속할 수 있으며, 주어진 태스크는 동일 액티비티에 대한 여러 개의 인스턴스를 가질 수 있다.

  반면, "singleTask"와 "singleInstance" 액티비티는 단지 하나의 인스턴스로 제한된다. 이러한 액티비티는 태스크의 루트에 있기 때문에, 이 제약은 그 태스크의 인스턴스가 한 개 이상 그 디바이스 상에 동시에 있을 수 없다는 것을 의미한다.

- **액티비티의 인스턴스가 그것의 태스크에 다른 액티비티를 가질 수 있는가?** "singleInstance" 액티비티는 그것이 속한 태스크 내에서 하나의 액티비티로 홀로 존재한다. 만약 그것이 다른 액티비티를 시작하면, 그 액티비티는 그 런치 모드와 상관없이 별도의 태스크에서 런치될 것이다. 이것은 마치 FLAG_ACTIVITY_NEW_TASK 플래그가 인텐트 내에 있는 것과 같다. 그 밖의 모든 측면에서, "singleInstance" 모드는 "singleTask"와 동일하다.

  나머지 다른 세 개의 모드는 그 태스크에 여러 개의 액티비티가 속하는 것을 허용한다. "singleTask" 액티비티는 항상 그 태스크의 루트 액티비티가 될 것이다. 그러나 그것은 그 태스크에 지정될 다른 액티비티들을 시작할 수 있다. "standard"와 "singleTop" 액티비티의 인스턴스는 스택의 어디에서든 나타날 수 있다.

- **새로운 클래스 인스턴스가 새로운 인텐트를 처리하기 위해 런치될 것인가?**

디폴트 "standard" 모드에 대해서는, 새로운 인스턴스가 모든 신규 인텐트에 응답하도록 생성된다. 각각의 인스턴스는 단지 하나의 인텐트를 처리한다. "singleTop" 모드에 대해서는, 만약 그것이 타겟target 태스크의 최상위 액티비티 스택에 존재한다면, 이미 존재하는 클래스 인스턴스가 신규 인텐트 처리를 위해 재사용된다. 만약 그것이 최상위에 존재하지 않다면 재사용되지 않는다. 대신 신규 인텐트를 위한 새 인스턴스가 생성되고, 스택에 푸쉬push된다.

예를 들어 태스크의 액티비티 스택이 A-B-C-D 가 되도록, 루트 액티비티 A와 그 위에 순서대로 액티비티 B, C, D로 구성되어 있다고 가정하자. 타입 D 액티비티에 대한 인텐트가 도착한다. 만약에 D가 디폴트 "standard" 런치 모드를 가지고 있다면, 새로운 클래스 인스턴스가 시작되며 스택은 A-B-C-D-D가 된다. 그러나 D의 런치 모드가 "singleTop" 이라면, (그것이 스택의 최상위에 있기 때문에) 기존에 있는 인스턴스가 새로운 인텐트를 처리하게 될 것이고 스택은 A-B-C-D로 남는다. 반면 도착한 인텐트가 타입 B 액티비티에 대한 것이라면, B가 "standard"든 "singleTop"이든 상관없이 (B가 스택의 최상위에 없기 때문에) 새로운 B 인스턴스가 런치될 것이다. 그러므로 결과적으로 스택은 A-B-C-D-B가 될 것이다

위에서 언급했듯이, "singleTask"나 "singleInstance" 액티비티에 대해서는 하나 이상의 인스턴스가 결코 존재하지 않는다. 그러므로 그 인스턴스는 모든 새로운 인텐트를 처리하게 될 것이다. "singleInstance" 액티비티는 (이것이 태스크에 있는 유일한 액티비티이기 때문에) 항상 스택의 최상위에 있다. 그러므로 그것은 항상 인텐트를 처리하기 위한 위치에 있다. 그러나 "singleTask" 액티비티는 스택에서 그것 위에 다른 액티비티를 가질 수도 있고 가지지 않을 수도 있다. 만약 다른 액티비티가 있다면 인텐트를 처리할 위치에 있지 않는 것이고, 결과적으로 그 인텐트는 누락된다(비록 인텐트가 누락된다고 할지라도, 인텐트의 도착은 해당 태스크를 포어그라운드로 오게 할 것이며, 태스크는 그 상태로 남게된다).

이미 존재하는 액티비티가 새로운 인텐트를 처리하도록 요청될 때, 인텐트 오브젝트는 onNewIntent() 호출을 통해서 그 액티비티에 전달된다(최초에 그 액티비티를 시작했던 인텐트 오브젝트는 getIntent()를 호출함으로써 얻어질 수 있다).

액티비티의 신규 인스턴스가 신규 인텐트를 처리하기 위해 만들어질 때, 사용자가 이전의 상태(이전의 액티비티로) 돌아가기 위해 BACK 키를 항상 누를 수 있다는 점

에 주의하라. 하지만 이미 존재하던 액티비티의 인스턴스가 신규 인텐트를 처리할 때, 사용자는 신규 인텐트가 도달하기 전에 그 인스턴스가 하던 작업으로 돌아가기 위해 BACK 키를 누를 수 없다.

시작 모드에 대한 더 자세한 것들에 대해서는 9장. "AndroidManifest.xml"의 〈activity〉 엘리먼트 설명을 보라.

스택(Stack) 제거

만약 사용자가 오랫동안 하나의 태스크를 떠나 있다면, 시스템은 루트 액티비티를 제외한 모든 액티비티를 태스크에서 제거한다. 사용자가 다시 그 태스크로 돌아왔을 때, 오직 최초의 액티비티만이 존재한다는 것 이외에 그것은 사용자가 그것을 떠났을 때와 같다. 이 아이디어는, 어느정도 시간이 지난 후 사용자는 그들이 이전에 하던 것을 포기했을 것이라는 것이고 뭔가 새로운 것을 시작하기 위해 그 태스크로 돌아왔을 거라고 생각하는 것이다.

이것이 디폴트다. 이러한 동작을 제어하고 수정하기 위한 몇몇 액티비티 애트리뷰트가 있다.

alwaysRetainTaskState 애트리뷰트

만약 태스크의 루트 액티비티에 이 애트리뷰트가 "true"로 설정된다면, 앞서 언급된 디폴트 동작은 발생하지 않는다. 태스크는 긴 시간 지난 이후에도 스택 내의 모든 액티비티들을 유지한다.

clearTaskOnLaunch 애트리뷰트

만약 태스크의 루트 액티비티에 이 애트리뷰트가 "true"로 설정된다면, 사용자가 태스크를 떠났다가 다시 돌아올 때마다, 스택은 루트 액티비티를 제외하고 모두 제거된다. 바꾸어 말하면 그것은 alwaysRetainTaskState와 정반대이다. 일시적으로 떠났다고 하더라도, 사용자는 항상 그것의 최초 상태로 태스크에 돌아오게 된다.

finishOnTaskLaunch 애트리뷰트

이 애트리뷰트는 clearTaskOnLaunch와 비슷하다. 그러나 그것은 전체 태

스크가 아닌, 하나의 액티비티에서 동작한다. 그리고 그것은 루트 액티비티를 포함한 모든 액티비티를 제거할 수 있다. 이것이 "true"로 설정되면, 액티비티는 현재 세션에 대한 태스크의 부분으로만 남게된다. 만약 사용자가 떠났다가 태스크에 돌아오면 그것은 더 이상 존재하지 않는다.

스택에서 액티비티가 제거되도록 하는 다른 방법이 있다. 만약 인텐트 오브젝트가 FLAG_ACTIVITY_CLEAR_TOP 플래그를 가지고 있고, 타겟target 태스크가 그것의 스택에 해당 인텐트를 처리해야 하는 타입의 액티비티 인스턴스를 이미 가지고 있다면, 그것이 스택의 최상위 존재해서 그 인텐트에 응답할 수 있도록 하기 위해 그 위의 모든 액티비티들은 제거된다. 만약 지정된 액티비티의 런치 모드가 "standard"라면, 그것 또한 스택에서 제거될 것이고, 새로운 인스턴스는 도착한 인텐트를 처리하기 위해 시작될 것이다. 왜냐하면 런치 모드가 "standard" 일 때 새로운 인스턴스는 새로운 인텐트를 위해 항상 생성되기 때문이다.

FLAG_ACTIVITY_CLEAR_TOP은 거의 대부분 FLAG_ACTIVITY_NEW_TASK와 함께 사용된다. 이것이 함께 사용될 때 이 플래그들은 다른 태스크 내에 있는 기존 액티비티를 지정하고, 그것을 인텐트에 응답할 수 있는 위치에 놓는다.

태스크 시작하기

액티비티는 그것의 인텐트 필터에 액션을 "android.intent.action.MAIN"으로 지정하고 카테고리를 "android.intent.category.LAUNCHER"로 지정함으로써, 태스크에 대한 시작점으로 설정된다(이 필터 타입의 예제가 이전의 인텐트 필터 섹션에 있다). 이런 종류의 필터는 액티비티에 대한 아이콘과 라벨이 애플리케이션 런처에서 보여지도록 하며, 사용자에게 태스크를 런치하고 태스크가 런치된 이후 언제든 그 태스크로 다시 돌아갈 수 있는 두 가지 방법 모두를 제공한다.

이 두 번째 능력은 중요하다. 사용자는 태스크를 떠날 수도 있어야 하고, 나중에 그것으로 돌아갈 수도 있어야 한다. 이런 이유로, 액티비티가 태스크를 항상 시작시키는 두 개의 런치 모드, 즉 "singleTask"와 "singleInstance"는 액티비티가 MAIN 과 LAUNCHER 필터를 가지고 있을 때만 사용되어야 한다. 예를 들어 만약 이 필터가 없다면 무슨 일이 발생할까? 상상해 보라. 인텐트는 새로운 태스

크를 시작하는 “singleTask” 액티비티를 런치하고, 사용자는 그 태스크에서 일을 하면서 약간의 시간을 보낸다. 사용자는 그런 다음 HOME 키를 누른다. 태스크는 이제 뒤에 놓이고, 홈스크린에 의해 가려진다. 그리고 그것은 애플리케이션 런처에서 나타나지 않기 때문에, 사용자는 그곳에 다시 돌아갈 방법이 없게 된다.

FLAG_ACTIVITY_NEW_TASK 플래그에도 비슷한 어려움이 수반된다. 만약 이 플래그가 새로운 태스크를 시작하는 액티비티를 만들고, 사용자가 그곳을 떠나기 위해 HOME 키를 누른다면, 사용자가 그곳으로 다시 돌아갈 수 있는 몇 가지 방법이 있어야 한다. 노티피케이션Notification 매니저와 같은 어떤 엔티티entity들은, 결코 그것들의 소유 영역이 아닌, 외부 태스크에서 액티비티를 시작시킨다. 그러므로 그들은 startActivity()에 파라미터로 전달하는 인텐트에 항상 FLAG_ACTIVITY_NEW_TASK를 놓는다. 만약 여러분이 이 플래그를 사용할 수도 있는 외부 엔티티에 의해 호출될 수 있는 액티비티를 가지고 있다면, 사용자가 그것이 시작된 태스크로 다시 돌아갈 수 있는 독자적인 방법을 갖게 하도록 주의하라.

그곳에서 사용자가 액티비티로 돌아가는 것을 가능하게 하는 것을 여러분이 원하지 않는 경우라면, 〈activity〉 엘리먼트의 finishOnTaskLaunch를 “true”로 설정하라. 앞서서 언급한 “스택 제거”를 보라.

프로세스와 쓰레드

애플리케이션의 컴포넌트들 중 첫 번째 컴포넌트가 실행되어야 할 필요가 있을 때, 안드로이드는 그것에 대한 하나의 실행execution 쓰레드를 가지는 리눅스 프로세스를 시작한다. 디폴트로 모든 애플리케이션의 컴포넌트는 이런 프로세스와 쓰레드에서 실행된다.

하지만 여러분은 컴포넌트가 다른 프로세스에서 실행되도록 배치할 수 있으며, 여러분은 임의의 프로세스에 대한 추가적인 쓰레드들을 만들 수 있다.

프로세스

컴포넌트가 실행되는 장소인 프로세스는 매니페스트 파일에 의해 제어된다. 컴포넌트 엘리먼트 –〈activity〉, 〈service〉, 〈receiver〉, 그리고 〈provider〉 – 각

각은 그 컴포넌트가 실행되어야 하는 프로세스를 지정할 수 있는 process 애트리뷰트를 가진다. 이 애트리뷰트는 각각의 컴포넌트가 자신만의 프로세스에서 실행되도록 설정될 수 있으며, 또한 몇몇 컴포넌트는 다른 것들과 달리 하나의 프로세스를 공유하도록 설정될 수도 있다. 또한 이 애트리뷰트는 동일 프로세스에서 다른 애플리케이션의 컴포넌트가 실행되도록 설정될 수 있다. 이것은 애플리케이션이 동일 리눅스 유저user ID를 공유하고, 동일 인증서로 사인되는 조건하에서 가능하다. 〈application〉 엘리먼트 또한 모든 컴포넌트에 디폴트 값을 설정하기 위한 process 애트리뷰트를 가지고 있다.

모든 컴포넌트는 지정된 프로세스의 메인 쓰레드에서 인스턴스화 되며, 그 컴포넌트에 대한 시스템 호출은 메인 쓰레드로부터 디스패치dispatch된다. 각각의 인스턴스에 대해 별도의 쓰레드가 생성되지는 않는다. 결과적으로, 그러한 호출에 응답하는 메쏘드(사용자의 액션을 리포트 하는 View.onKeyDown()와 "컴포넌트 생명주기" 섹션에서 나중에 논의될 생명주기 노티피케이션과 같은 메쏘드)는 항상 그 프로세스의 메인 쓰레드에서 실행된다. 이것은 컴포넌트가 시스템에 의해 호출될 때 (네트워킹 오퍼레이션이나 계산 루프처럼) 오래 걸리거나 블록킹blocking되는 오퍼레이션을 수행해서는 안된다는 것을 의미한다. 왜냐하면, 이것들이 프로세스 내의 다른 컴포넌트를 방해할 것이기 때문이다. 여러분은 오래 걸리는 오퍼레이션을 위해, 이 다음의 "쓰레드"에서 논의되듯이 별도의 쓰레드를 만들 수 있다.

안드로이드는 메모리가 부족하고 사용자를 더 즉각적으로 지원하고 있는 프로세스에 의해 메모리가 요구될 때, 임의의 시점에 프로세스를 종료하는 것을 결정할 수도 있다. 그 프로세스 안에서 실행되는 애플리케이션 컴포넌트는 결과적으로 파괴된다. 그것을 다시 수행해야 할 때 그 컴포넌트에 대한 프로세스는 다시 시작된다.

어떤 프로세스를 종료할 것인가를 결정할 때, 안드로이드는 사용자에 대한 상대적 중요성을 평가한다. 예를 들어 안드로이드는 보여지고 있는 액티비티를 가진 프로세스보다 스크린 상에서 더 이상 보이지 않는 액티비티를 가진 프로세스를 더 우선적으로 종료시킨다. 그러므로 프로세스에 대한 종료 여부 결정은 그 프로세스에서 실행되고 있는 컴포넌트의 상태에 의존한다. 그 상태는 다음에 나올 "컴포넌트 생명주기" 섹션의 주제이다.

쓰레드(Thread)

비록 애플리케이션을 하나의 프로세스에 제한할 지라도, 여러분은 몇몇 백그라운드 작업을 하기 위한 쓰레드를 생성할 필요가 있을 수 있다. 사용자 인터페이스가 사용자 액션에 항상 빠르게 응답해야 하기 때문에, 액티비티를 보유하는 쓰레드는 네트워크 다운로드와 같은 시간이 소요되는 오퍼레이션을 함께 보유해서는 안 된다. 빠르게 수행이 완료될 수 없는 것은 별도의 쓰레드에 할당되어져야 한다.

쓰레드는 표준 Java 쓰레드Thread 오브젝트를 사용해서 코드에서 생성된다. 안드로이드는 쓰레드 관리를 위한 몇 가지 편리한 클래스들을 제공한다 – 쓰레드 내에서 메시지 루프를 실행하기 위한 루퍼Looper, 메시지 처리를 위한 핸들러Handler, 그리고 메시지 루프를 가진 쓰레드를 설정하기 위한 핸들러쓰레드HanlderThread가 그것이다.

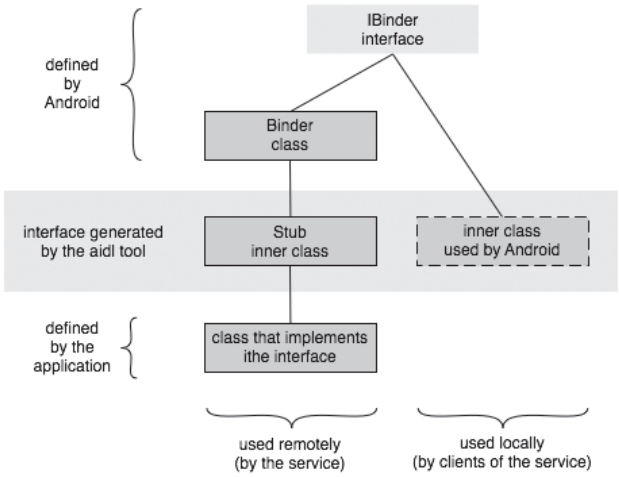
원격 프로시저 호출(Remote procedure call)

안드로이드는 원격 프로시저 호출(remote procedure calls, RPCs)를 위한 경량화된 lightweight 메커니즘을 가지고 있다 – 그곳에서 메쏘드는 지역적으로locally 호출되지만, 호출자에게 다시 리턴되는 결과를 가지는 다른 프로세스 상에서 원격으로remotely 실행된다. 이것은 그 메쏘드 호출과 그것이 수반하는attendant 모든 데이터를 운영체제가 이해할 수 있는 수준으로 분해decomposing하며, 그것을 로컬 프로세스와 어드레스address 공간에서 원격 프로세스와 어드레스 공간으로 전달하고, 그곳에서 그 호출을 재조합reassembling하고 재연reenacting하는 것을 야기한다. 리턴 값은 정반대의 방향에서 전달되어져야 한다. 안드로이드는 여러분이 RPC 인터페이스 자체를 정의하고 구현하는 것에 집중할 수 있도록, 그 작업을 하기 위한 모든 코드를 제공한다.

RPC 인터페이스는 오직 메쏘드만을 포함할 수 있다. 모든 메쏘드는 비록 리턴 값이 없다 할지라도 동기적으로 실행된다(로컬 메쏘드는 원격지의 메쏘드가 끝날 때까지 멈춰 있다).

간단하게 말하면, 그 메커니즘은 다음과 같이 동작한다. 여러분은 간단한 IDL (interface definition language, 인터페이스 정의 언어)을 사용하여 여러분이 구현하고자 하는 RPC 인터페이스를 선언하는 것에서 시작할 수 있다. 그 선언에 기반해서 aidl

툴tool은 로컬과 원격지 프로세스 모두에서 사용되어져야 하는 Java 인터페이스 정의definition를 생성한다. 이것은 아래 다이어그램에서 보여지듯이 다음과 같은 두 개의 내부inner 클래스를 포함한다.



내부 클래스들은 IDL을 가지고 선언했던 인터페이스에 대한 원격 프로시저 호출을 관리하기 위해 필요한 모든 코드를 가지고 있다. 두 개의 내부 클래스 모두는 IBinder 인터페이스를 구현하고 있다. 그들 중 하나는 시스템에 의해 지역적이고locally 내부적으로internally 사용된다; 여러분이 작성한 코드는 그것을 무시할 수 있다. 다른 하나는 스터브Stub라고 불리는 것으로 바인더Binder 클래스를 확장하고extends 있는 것이다. 그것은 IPC 호출을 가능하게 하는 내부 코드뿐만 아니라, 여러분이 선언한 RPC 인터페이스 내의 메쏘드에 대한 선언을 포함하고 있다. 다이어그램에서 보여지듯이, 여러분이 정의한 RPC 인터페이스 내의 메쏘드를 구현하기 위해서는 스터브Stub 서브클래스가 필요할 것이다.

일반적으로 원격지 프로세스는 서비스로 관리될 것이다(왜냐하면 서비스는 그 프로세스와 다른 프로세스에 대한 그것의 커넥션(connection)에 대하여 시스템에 알려줄 수 있기 때문이다). 그것은 aidl 툴tool에 의해 생성된 인터페이스 파일과 RPC 메쏘드를 구현하는 스터브Stub 서브클래스 모두를 가질 것이다. 그 서비스 클라이언트는 단지 aidl 툴tool에 의해 생성된 인터페이스 파일만을 가질 것이다.
여기에 서비스와 그것의 클라이언트 간의 커넥션이 셋업되는 방법이 있다.

- (로컬 쪽에 있는) 서비스 클라이언트는 그 클라이언트가 원격지 서비스에 대한 커넥션을 성공적으로 확립할 때와 그 커넥션이 끊길 때, 그것이 통보될 수 있도록 onServiceConnected()와 onServiceDisconnected() 메쏘드를 구현할 것이다. 그것들은 그런 다음 해당 커넥션을 셋업하기 위해 bindService()를 호출할 것이다.

- 서비스의 onBind() 메쏘드는 그것이 수신하는 인텐트(bindService()에 파라미터로 전달된 인텐트)에 의존해서, 커넥션을 승인하거나 거절하도록 구현될 것이다. 만약 커넥션이 승인된다면, 그것을 스터브Stub 서브클래스에 대한 인스턴스를 리턴한다.

- 만약 그 서비스가 그 커넥션을 수용한다면, 안드로이드는 클라이언트의 onServiceConnected() 메쏘드를 호출하며, 그것에 서비스에 의해 관리되는 스터브Stub 서브클래스에 대한 프락시proxy, IBinder 오브젝트를 전달한다. 그 프락시를 통해, 클라이언트는 원격지 서비스에 호출을 만들 수 있다.

이 간단한 설명은 RPC 메커니즘의 몇 가지 상세한 설명은 생략한다. 더 많은 정보는 15장. "AIDL을 사용하여 원격 인터페이스 디자인하기"와 IBinder 클래스 설명을 보기 바란다.

쓰레드-세이프 메쏘드(Thread-safe method)

몇몇의 컨텍스트context에서, 여러분이 구현한 메쏘드는 한 개 이상의 쓰레드로부터 호출될 수 있다. 그러므로 그것은 쓰레드-세이프thread-safe하게 작성되어져야 한다.

이것은 (이전 섹션에서 논의했던 RPC 메커니즘에서와 같이) 원격으로 호출될 수 있는 메쏘드에 있어서 중요한 사실이다. IBinder 오브젝트에 구현된 메쏘드 호출이 그 IBinder로써 동일 프로세스에서 만들어질 때, 그 메쏘드는 호출자의 쓰레드에서 실행된다. 하지만 그 호출이 다른 프로세스에서 만들어질 때, 그 메쏘드는 IBinder로써 동일 프로세스에서 안드로이드가 유지하고 있는 쓰레드 풀pool로부터 선택된 쓰레드에서 실행된다. 그것은 프로세스의 메인 쓰레드 내에서 실행되지 않는다. 예를 들어 서비스의 onBind() 메쏘드가 서비스의 프로세스의 메인 쓰레드로부터 호출되는 반면에, onBind()가 반환(예를 들어 RPC 메쏘드를 구현하고 있는 스터브(Stub) 서브클래스)하는 오브젝트에 구현된 메쏘드는 풀pool 내의 쓰레드에서 호출된다. 서비스가 하나 이상의 클라이언트를 가질 수 있기 때문에, 하나 이상의 풀pool 쓰레드가 동일

시간에 동일한 IBinder 메쏘드를 사용할 수 있다. 그러므로 IBinder 메쏘드는 쓰레드-세이프thread-safe하도록 구현되어야 한다.

유사하게, 컨텐트 프로바이더는 다른 프로세스에서 만들어진 데이터 요청을 받을 수 있다. 비록 컨텐트 리졸버ContentResolver와 컨텐트 프로바이더ContentProvider 클래스가 그 프로세스 간 통신이 관리되는 자세한 방법을 숨기고 있지만, 그러한 요청(query(), insert(), delete(), update(), getType() 메쏘드)에 응답하는 컨텐트 프로바이더 메쏘드는 컨텐트 프로바이더 프로세스의 쓰레드 풀pool에서 호출된다. 이러한 메쏘드가 동일 시간에 다른 많은 쓰레드에서 호출될 수 있기 때문에, 그들 또한 쓰레드-세이프thread-safe하도록 구현되어야 한다.

컴포넌트 생명주기

애플리케이션 컴포넌트는 생명주기를 가진다 – 그 생명주기는 인텐트에 응답하기 위해 안드로이드가 컴포넌트를 인스턴스화 할 때 시작해서, 그 인스턴스가 파괴될 때 끝난다. 액티비티의 경우에는 그 사이에 활성화되거나 비활성화될 수도 있으며, 사용자에게 보이거나 보이지 않을 수 있다. 이번 섹션은 액티비티, 서비스, 브로드캐스트 리시버에 대한 생명주기를 설명한다 – 이곳에서는 그것들이 생존기간 동안에 있을 수 있는 상태, 상태간의 변화를 여러분이 통보받을 수 있는 메쏘드, 그리고 컴포넌트를 보유하고 있는 프로세스가 종료되고 그 인스턴스가 파괴되는 일이 발생할 때 그들 상태의 영향에 대해 언급한다.

액티비티 생명주기

액티비티는 필수적으로 세 가지 상태를 가지고 있다.

- **활성화(active) 또는 실행 중(running)인 상태**로, 그것이 포어그라운드foreground 화면(현재 태스크에 대한 액티비티 스택의 최상위에)으로 있을 때이다. 이것은 해당 액티비티가 사용자의 액션에 대한 포커스를 가지고 있다.

- **멈춤(paused) 상태**로, 포커스는 보유하지 않았지만 여전히 사용자에게 보여지고 있을 때이다. 즉, 다른 액티비티가 그 위에 위치하지만 그 액티비티가 투명상태 또는 전체 화면을 채우지 못하고 있어서 약간이라도 멈춤 상태의 액티비티를 볼 수 있는 상태이다. 멈춤상태의 액티비티는 완전하게 살아 있다(그것은 모든 상태와 멤버 정보를 유지하고 있으며, 윈도우 매니저에게 붙어서 존속하고 있다). 그러나 극도로

메모리가 부족한 상태에서는 시스템에 의해 종료될 수도 있다.

- **정지(stopped) 상태**는, 그것이 완벽하게 다른 액티비티에 의해서 가려진 상태이다. 그리고 여전히 모든 상태와 멤버 정보는 유지된다. 하지만 이것은 더 이상 사용자에게 보여지지 않는다. 그러므로 해당 액티비티 윈도우는 사라지게 되며, 그것은 다른 곳에서 메모리가 필요하게 될 때 시스템에 의해 가끔 종료될 것이다.

액티비티가 멈춤paused상태이거나 또는 정지stopped상태라면, 시스템은 finish() 메쏘드를 호출함으로써 그것을 종료하게 하거나 또는 간단하게 해당 프로세스를 죽이는 방식으로 그것을 메모리에서 제거할 수 있다. 그리고 그것이 사용자에게 다시 보여지게 될 때, 그것은 완전하게 다시 시작되고 이전의 상태로 복구되어야 한다.

액티비티의 상태가 변할 때면, 그것은 다음과 같은 프로텍티드protected 메쏘드가 호출됨으로써 그 변화가 통보된다.

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onRestart()
void onResume()
void onPause()
void onStop()
void onDestroy()
```

이러한 모든 메쏘드는 여러분이 액티비티의 상태가 변할 때, 적절한 작업을 수행하기 위해 오버라이드override할 수 있는 것이다. 모든 액티비티는 오브젝트가 최초로 인스턴스화 될 때 초기화를 설정하기 위한 onCreate()를 구현해야 한다. 많은 액티비티들이 데이터 변경을 저장하기 위해, 다른 경우로는 사용자와의 상호작용 중지를 준비하기 위해, onPause()를 구현할 것이다.

슈퍼클래스 호출하기

임의의 액티비티 생명주기 메쏘드에 대한 구현은 항상 먼저 슈퍼클래스 버전을 호출해야 한다. 예를 들어 다음과 같다.

```
protected void onPause() {
    super.onPause();
    . . .
}
```

이러한 7가지의 메쏘드들이 함께 사용되어서, 하나의 액티비티에 대한 전체 생명 주기가 정의된다. 이곳에는 여러분이 그것들을 구현함으로써 모니터링할 수 있는 3개의 네스티드 루프nested loop가 존재한다.

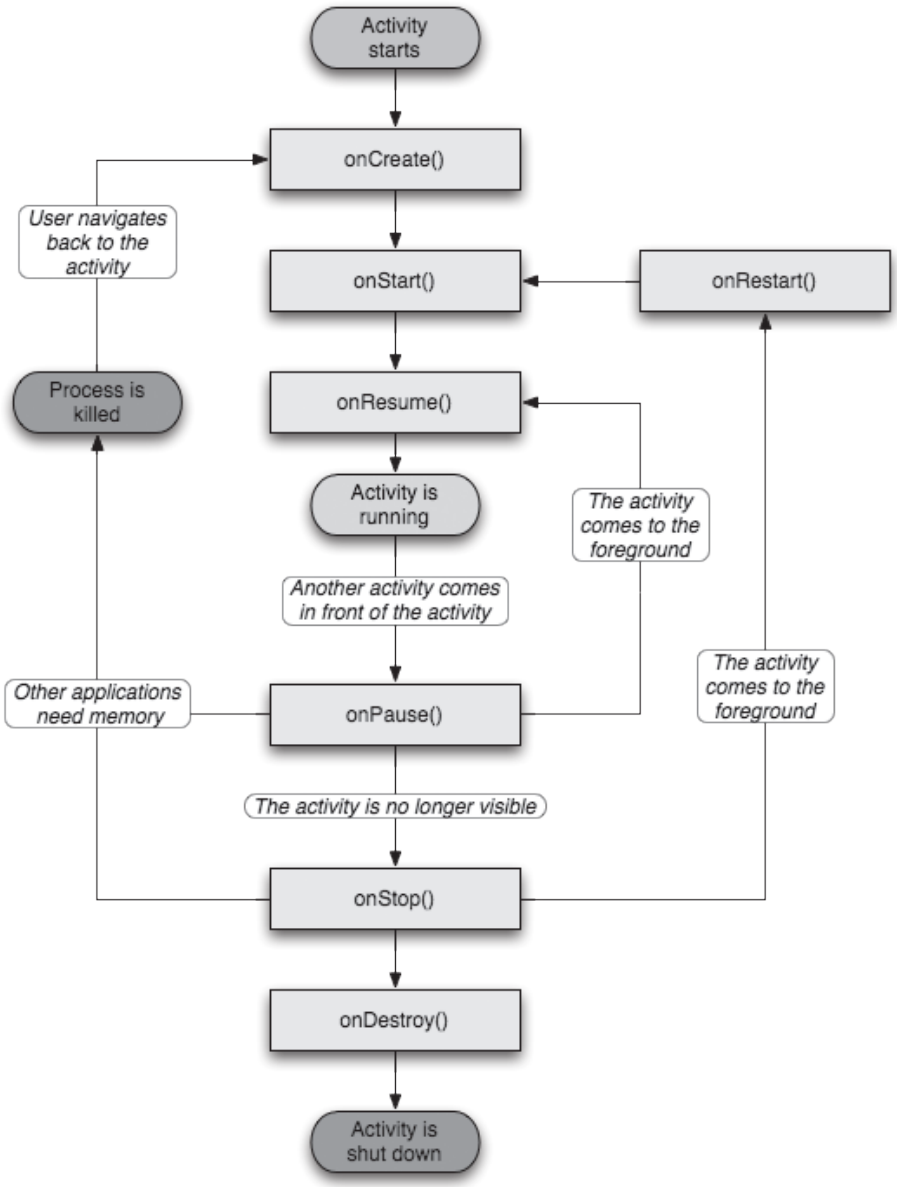
- 액티비티의 **인타이어 라이프타임(entire lifetime)**은 onCreate()에 대한 첫 번째 호출에서 onDestroy()에 대한 한번의 마지막 호출 사이에서 발생한다. 액티비티는 onCreate()에서 "전역global" 상태의 모든 초기 설정을 하며, 그리고 onDestroy()에서 존재하는 모든 리소스를 해제한다. 예를 들어 만일 그것이 네트워크에서 자료를 다운로드하기 위해 백그라운드background로 실행되고 있는 쓰레드를 가지고 있다면, 그 쓰레드는 onCreate()에서 생성되어 onDestroy()에서 종료될 것이다.

- 액티비티의 **비저블 라이프타임(visibile lifetime)**은 onStart()에 대한 호출에서 그에 대응되는 onStop() 호출 사이에서 발생한다. 이 기간 동안 액티비티가 사용자와 상호작용을 하는 포어그라운드foreground 상에 존재하지 않음에도 불구하고, 사용자는 화면 상에서 그 액티비티를 볼 수 있다. 이 두 개의 메쏘드 사이에서, 여러분은 사용자에게 액티비티를 보여주기 위해 필요한 리소스들을 유지할 수 있다. 예를 들어 여러분은 onStart()에서 여러분의 사용자 인터페이스에 영향을 미치는 변화들을 모니터링하기 위해 브로드캐스트 리시버를 등록할 수 있으며, 여러분이 보여주고 있는 것을 더 이상 사용자가 볼 수 없을 때 onStop()에서 그것을 제거할 수 있다. onStart()와 onStop() 메쏘드는 액티비티가 사용자에게 보여지게 되고 숨겨지게 되는 것 사이의 선택적 상황에 따라 여러 번 호출될 수 있다.

- 액티비티의 **포어그라운드 라이프타임(foreground lifetime)**은 onResume()의 호출과 그에 대응되는 onPause() 호출 사이에서 발생한다. 이 기간 동안 액티비티는 화면 상의 다른 모든 액티비티들의 앞에 놓이며, 사용자와 상호작용을 하게 된다. 액티비티는 재개resumed 상태와 멈춤paused 상태 사이에서 자주 바뀔 수 있다. 예를 들어 onPause()는 디바이스가 절전상태로 되거나 또는 새로운 액티비티가 시작되었을 때 호출되며, onResume()은 액티비티가 복귀될 때 또는 새로운 인텐트가 도착했을 때 호출된다. 그러므로 이 두 개의 메쏘드 내의 코드는 상당히 가벼워야 한다.

아래의 다이어그램은 이러한 루프들과 하나의 액티비티가 상태들을 이동해 가는 경로를 설명한다. 채색된 타원들은 액티비티가 존재하게 되는 주요 상태들이다.

직사각형들은 액티비티가 상태들 사이를 변화해 갈 때 오퍼레이션들을 수행하기 위해 여러분이 구현할 수 있는 콜백callback 메쏘드를 나타낸다.



아래의 테이블은 이러한 메쏘드들 각각에 대한 좀 더 자세한 설명하며, 해당 메쏘드들을 전체 액티비티의 생명주기 내에서 배치하고 있다.

메쏘드	설명	강제종료 가능?	다음 메쏘드
onCreate()	액티비티가 최초로 생성될 때 호출된다. 이곳은 여러분이 뷰 생성, 리스트에 대한 데이터 바인딩, 기타 등등과 같은 모든 일반적인 정적static 설정을 해야 하는 곳이다. 만일 액티비티의 상태가 보관되었다면**(나중에 액티비티 상태 저장하기를 보기 바람)**, 이 메쏘드는 액티비티의 이전 상태를 보유하고 있는 Bundle 오브젝트를 전달받는다. 항상 onStart()가 뒤에 이어진다.	아니오	onStart()
onRestart()	액티비티가 정지된 후, 그것이 다시 시작되기 바로 전에 호출된다. 항상 onStart()가 뒤에 이어진다.	아니오	onStart()
onStart()	액티비티가 사용자에게 보여지기 바로 직전에 호출된다. 만약 액티비티가 포어그라운드foreground로 오면 onResume()이, 또는 액티비티가 숨겨지게 되면 onStop()이 이어진다.	아니오	onResume() 또는 onStop()
onResume()	액티비티가 사용자와 상호작용을 시작하기 바로 직전에 호출된다. 이 시점에서 액티비티는 사용자 입력을 받게 되는 액티비티 스택의 최상위에 있게 된다. 항상 onPause()가 이어진다.	아니오	onPause()
onPause()	시스템이 다른 액티비티를 재개resume하기 시작하려 할 때 호출된다. 이 메쏘드는 일반적으로 저장되지 않은 변경들에 대한 영구적 데이터로 저장, 애니메이션과 CPU를 소모할 수 있는 다른 것들에 대한 중지, 그리고 기타 등등을 위해 사용된다. 이곳에서 무엇을 하던 간에 아주 빨리 해야만 한다. 왜냐하면 이 메쏘드가 리턴될 때까지 다음번 액티비티가 재기(resume)되지 않기 때문이다. 만약 액티비티가 프론트로 다시 리턴되면 onResume()이, 또는 액티비티가 사용자에게 보이지 않게 되면 onStop()이 이어진다.	예	onResume() 또는 onStop()
onStop()	액티비티가 사용자에게 더이상 보여지지 않을 때 호출된다. 이것은 액티비티가 파괴될 것이기 때문에 발생하거나, 또는 다른 액티비티**(기존의 액티비티든, 신규 액티비티든)**가 재개resume되어 해당 액티비티를 덮기 때문에 발생한다. 액티비티가 사용자와 상호작용하기 위해 돌아 온다면 onRestart()가, 또는 이 액티비티가 사라지게 된다면 onDestory()가 이어진다.	예	onRestart() 또는 onDestroy()
onDestroy()	액티비티가 파괴되기 전에 호출된다. 이것은 액티비티가 받게 될 마지막 호출이다. 이것은 액티비티가 끝나기 때문이거나**(누군가가 액티비티에서 finish()를 호출함)**, 시스템이 일시적으로 공간을 절약하기 위해 액티비티의 인스턴스를 없애려고 하기 때문에 호출될 수 있다. 여러분은 isFinishing() 메쏘드를 사용해서 이러한 두 개의 시나리오 사이를 구분할 수 있다.	예	없음

위의 테이블에서 "강제종료가능?" 열에 주의하라. 그것은 메쏘드가 리턴된 후 어떤 시점에라도 액티비티 코드의 다른 라인을 실행하지 않고, 시스템이 액티비티를 담고 있는 프로세스를 강제종료할 수 있는지 여부를 가리킨다. 3개의 메쏘드(onPause(), onStop(), 그리고 onDestroy())에는 "예"라고 표시되어 있다. onPause()는 이 3개 중 첫 번째 것이기 때문에, 그것은 프로세스가 강제종료되기 전에 호출되는 것이 보증되어야 하는 유일한 것이다 – onStop()과 onDestory()는 아닐 수 있다. 그러므로 여러분은 (사용자의 편집과 같은) 임의의 영구 데이터를 저장공간에 쓰기 위해서는 onPause()를 사용해야 한다.

"강제종료가능?" 열에 "아니오"라고 표시된 메쏘드들은 그것들이 호출된 순간부터 액티비티를 보유하고 있는 프로세스가 강제종료되지 않도록 보호한다. 따라서, 예를 들어 onPause()가 리턴된 시각부터 onResume()이 호출된 시각까지 액티비티는 강제종료가능 상태에 있다. 그것은 onPause()가 다시 리턴할 때까지는 다시 강제종료가능하지 않을 것이다.

나중에 나오는 "프로세스와 생명주기" 섹션에서 언급하듯이, 이 정의에 의해서 기술적으로 "강제종료가능" 하지 않는 액티비티도 시스템에 의해 여전히 강제종료될 수 있다 – 하지만 이것은 다른 수단이 없을 때, 극도로 극단적인 상황에서만 발생할 것이다.

액티비티 상태 저장하기

사용자라기 보다는 시스템이 메모리를 절약하기 위해 액티비티를 종료시킬 때 사용자는 액티비티로 돌아가길 원할 수 있고, 그곳에서 액티비티의 이전 상태를 찾을 수도 있다.

액티비티가 강제종료되기 전의 상태를 보관하기 위해서, 여러분은 액티비티에 대한 onSaveInstanceState() 메쏘드를 구현할 수 있다. 안드로이드는 액티비티가 파괴될 수 있도록 만들기 전에 이 메쏘드를 호출한다 – 즉, onPause()가 호출되기 전에 호출한다. 그것은 여러분이 이름-값의 쌍key-value pair으로 액티비티의 동적 상태를 기록할 수 있는 번들Bundle 오브젝트를 그 메쏘드에 전달한다. 액티비티가 다시 시작될 때, 그 번들Bundle은 onCreate()와 onStart() 이후 호출되는 메쏘드인 onRestoreInstanceState() 메쏘드 둘 다에게 전달된다. 그러므

로 그것들 중 하나 또는 둘 모두는 저장된 상태를 다시 만들 수 있다.

onPause()와 앞에서 논의되었던 다른 메쏘드들과는 다르게, onSaveInstanceState()와 onRestoreInstanceState()는 생명주기 메쏘드가 아니다. 그것들은 항상 호출되지는 않는다. 예를 들어 안드로이드는 액티비티가 시스템에 의해 파괴되는 취약함에 도달하기 전에 onSaveInstanceState()를 호출한다. 그러나 인스턴스가 (BACK 키를 누르는 것 같은) 사용자 액션에 의해 실제로 파괴되고 있을 때에는 그것을 호출하려고 노력하지는 않는다. 그런 경우, 사용자는 그 액티비티로 돌아가는 것을 기대하지 않는다. 그러므로 그 상태를 저장할 이유가 없다.

onSaveInstanceState()는 언제나 호출되지 않기 때문에, 여러분은 영구적인 데이터를 저장하기 위해서가 아니라 액티비티의 일시적인 상태를 기록하기 위해서만 사용해야 한다. 그런 영구적인 데이터를 저장하는 목적을 위해서는 대신 onPause()를 사용하라.

액티비티 조화시키기 Coordinating activities

하나의 액티비티가 또 다른 액티비티를 시작할 때, 두 액티비티 모두는 생명주기 변화를 경험하게 된다. 하나는 멈추고pause 정지stop하게 되며, 반면에 다른 하나는 시작start된다. 경우에 따라 여러분은 이 액티비티들을 서로 간에 조화시키는 것이 필요할 수도 있다.

생명주기 콜백callback의 순서는 특히 두 개의 액티비티가 동일 프로세스 안에 있을 때 잘 정의된다.

1. 현재 액티비티의 onPause() 메쏘드가 호출된다.
2. 다음번에 시작되는 액티비티의 onCreate(),onStart(),그리고 onResume() 메쏘드가 차례로 호출된다.
3. 그런 다음에, 만약 시작되는 액티비티가 화면에 더 이상 보이지 않으면, 그것의 onStop() 메쏘드가 호출된다.

서비스 생명주기

서비스는 두 가지 방법으로 사용될 수 있다.

- 서비스는 시작될 수 있으며, 누군가가 그것을 중지시키거나 또는 스스로 중지할 때까지 실행되는 것이 가능하다. 이 모드에서 서비스는 Context.startService()와 Context.stopService()의 호출에 의해 시작되고 중지된다. 서비스는 Service.stopSelf()나 Service.stopSelfResult()의 호출에 의해 스스로 중지할 수 있다. startService()가 얼마나 많이 호출되었는지와 상관없이, 오직 한 번의 stopService() 호출만이 그 서비스를 중지하기 위해 필요하다.

- 서비스는 그것이 정의하고 제시하는 인터페이스를 사용해서 프로그램적으로 움직일 수 있다. 클라이언트는 서비스 오브젝트에 커넥션을 만들고, 서비스 내부를 호출하기 위해 커넥션을 사용한다. 커넥션은 Context.bindService()의 호출에 의해 만들어지며 Context.unbindService()의 호출에 의해 끊긴다. 많은 클라이언트가 동일한 서비스에 바인드 할 수 있다. 만약 서비스가 이미 런치되지 않았다면, bindService()는 선택적으로 그것을 런치할 수 있다.

이런 두 가지 모드는 전체적으로 분리되지 않는다. 여러분은 startService()로 시작된 서비스에 바인드할 수 있다. 예를 들어 백그라운드background 뮤직 서비스는 재생할 음악을 식별해주는 인텐트 오브젝트를 파라미터로 가진 startService() 호출에 의해 시작될 수 있다. 그리고 나중에, 아마도 사용자가 플레이어를 약간 제어하려 하거나 또는 현재 재생중인 음악에 대한 정보를 얻고자 할 때, bindService()를 호출함으로써 액티비티가 그 서비스에 대한 커넥션을 만들게 할 수 있다. 이와 같은 경우, stopService()는 마지막 바인딩이 닫힐 때까지 그 서비스를 실제로 중지하지 않을 것이다.

액티비티와 같이 서비스도 여러분이 그것의 상태 변화를 모니터링하기 위해 구현할 수 있는 생명주기 메쏘드를 가지고 있다. 그것은 단지 세 개이며, 프로텍티드protected가 아닌 퍼블릭public이다.

```
void onCreate()
void onStart(Intent intent)
void onDestroy()
```

이 메쏘드들을 구현함으로써 여러분은 서비스 생명주기에 있는 두 개의 네스티드 루프nested loop를 모니터링할 수 있다.

- 서비스의 **인타이어 라이프타임(entire lifetime)**은 onCreate()가 호출된 시

간으로부터 onDestroy()가 리턴된 시간 사이에서 발생한다. 액티비티처럼, 서비스는 onCreate()에서 초기 설정을 하며 onDestroy()에서 존재하는 모든 리소스를 해제한다. 예를 들어 음악 재생 서비스는 onCreate()에서 재생할 음악이 있는 곳에 쓰레드를 만들며, 그런 다음 onDestroy()에서 해당 쓰레드를 종료시킨다.

- 서비스의 **액티브 라이프타임(active lifetime)**은 onStart()를 호출함으로써 시작된다. 이 메쏘드에 startService()에 전달된 인텐트 오브젝트가 전달된다. 뮤직 서비스는 재생할 음악을 찾기 위해 그 인텐트를 오픈할 것이고, 그리고 나서 재생을 시작한다.

  서비스가 중지될 때에 해당되는 콜백, 즉 onStop() 메쏘드같은 것은 없다.

서비스가 Context.startService() 또는 Context.bindService() 어느 것에 의해 시작되었는지와 상관없이, 모든 서비스에서 onCreate()와 onDestroy() 메쏘드는 호출된다. 하지만 onStart()는 오직 startService()에 의해 시작된 서비스에서만 호출된다.

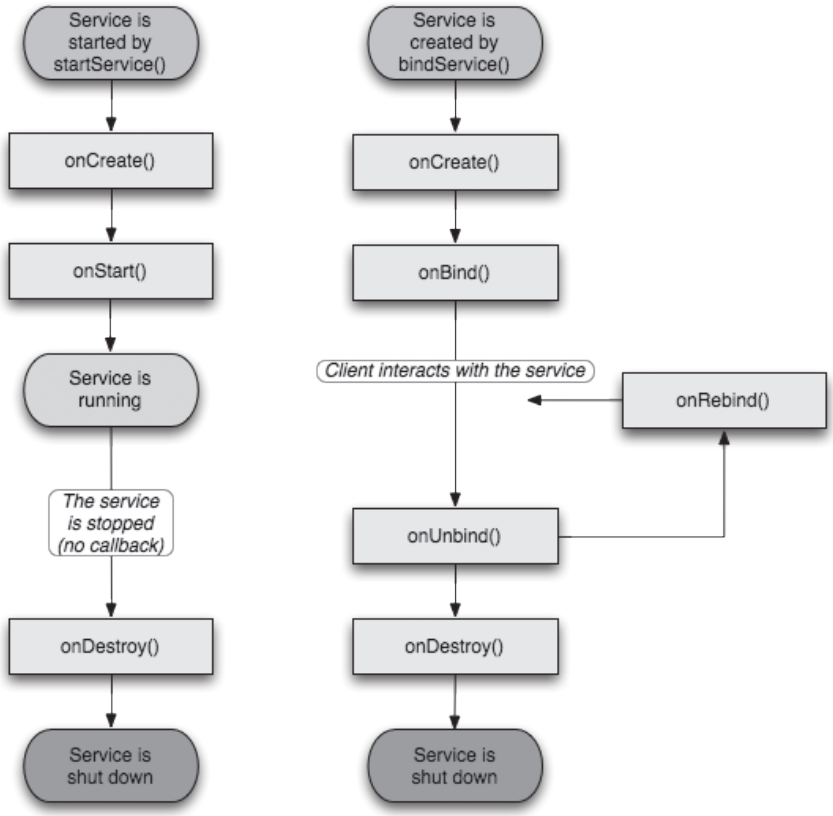
만일 서비스가 다른 것들이 자신에게 바인드하는 것을 허용한다면, 그것을 위해 추가적으로 구현해야 할 콜백 메쏘드들이 존재한다.

```
IBinder onBind(Intent intent)
boolean onUnbind(Intent intent)
void onRebind(Intent intent)
```

onBind() 콜백에는 bindService()에게 전달되었던 인텐트 오브젝트를 전달되고, onUnbind()에는 unbindService()에 전달되었던 인텐트가 전달된다. 만일 서비스가 바인딩을 허용한다면, onBind()는 클라이언트가 서비스와 상호작용하기 위해 사용할 커뮤니케이션 채널을 리턴한다. 만일 새로운 클라이언트가 서비스로 접속한다면, onUnbind() 메쏘드는 onRebind()가 호출되도록 요청할 수 있다.

아래의 다이어그램은 서비스에 대한 콜백 메쏘드들을 설명한다. 그것은 startService()에 의해 생생된 서비스와 bindService()에 의해 생성된 서비스를 구분하고 있음에도 불구하고, 서비스가 어떻게 시작되었다 할지라도, 임의의 서비스는 잠재적으로 클라이언트가 서비스에 바인드하는 것을 허용할 수 있다는 것에 유의하기 바란다. 그러므로 어떤 서비스든 onBind() 와 onUnbind() 호출

을 받을 수도 있다.



브로드캐스트 리시버 생명주기

브로드캐스트 리시버는 하나의 콜백 메쏘드를 가진다.

```
void onReceive(Context curContext, Intent broadcastMsg)
```

리시버에게 브로드캐스트 메시지가 도착할 때, 안드로이드는 onReceive() 메쏘드를 호출하며 메시지를 포함하고 있는 인텐트 오브젝트를 onReceive() 메쏘드에 전달한다. 브로드캐스트 리시버는 이 메쏘드를 실행하는 동안에만 활성화되는 것으로 간주된다. onReceive()가 리턴될 때, 리시버는 비활성화된다.

활성화된 브로드캐스트 리시버를 가지는 프로세스는 강제종료되는 것으로부터 보호된다. 그러나 단지 비활성화된 컴포넌트들을 가지는 프로세스는 그것이 낭비

하고 있는 메모리가 다른 프로세스에 의해 필요할 때, 언제든 시스템에 의해 강제 종료될 수 있다.

이것은 브로드캐스트 메시지에 대한 응답이 시간이 소요될 때 문제를 만든다. 그러므로 어떤 것은 사용자 인터페이스에 대한 다른 컴포넌트가 있는 메인 쓰레드를 벗어나 별도의 쓰레드에서 처리되어야 한다. 만약 onReceive()가 쓰레드를 생성하고 그 다음 리턴된다면, 새로운 쓰레드를 포함하는 전체 프로세스는 비활성화(다른 애플리케이션 컴포넌트가 프로세스 안에서 액티브이지 않는 한)로 판정되고, 그것은 강제종료 되는 위험에 놓여진다. 이 문제의 해결책은 시스템이 프로세스 안에서 진행되는 활성화된 작업이 아직도 있다는 것을 알고 있도록, onReceive()에서 서비스를 시작시키고 해당 서비스가 이 작업을 하게끔 하는 것이다.

다음 섹션에서 프로세스가 강제 중지되는 취약성에 대해서 더 자세히 다룬다.

프로세스와 생명주기

안드로이드 시스템은 가능한 한 오랫동안 애플리케이션 프로세스가 유지될 수 있도록 노력한다. 그러나 메모리가 부족한 상태에 이르면, 결국에는 오래된 프로세스를 제거하는 것이 필요하게 될 것이다. 안드로이드는 어떤 프로세스를 유지하고 어떤 프로세스를 죽일 것인가를 결정하기 위해, 각각의 프로세스를 각각의 프로세스 내에서 실행되는 컴포넌트와 컴포넌트의 상태에 기반하는 "중요성 계층구조(importance hierarchy)" 속에서 유지한다. 가장 낮은 중요성을 가진 프로세스가 첫 번째로 제거되며, 그 다음에는 다음으로 낮은 중요성을 가진 것 순서로 제거된다. 그 계층구조에는 다섯 개의 단계가 존재한다. 아래의 목록은 그것들을 중요성 순서로 보여준다.

1. **포어그라운드(foreground)** 프로세스는 사용자가 현재 수행하고 있는 것을 위해 요구되는 프로세스이다. 만약 하나의 프로세스가 아래와 같은 조건을 보유하고 있다면, 그것은 포어그라운드 프로세스로 간주된다.
   - 프로세스가 사용자와 상호작용하는 액티비티(액티비티 오브젝트의 onResume() 메쏘드가 호출되었던 적이 있음)를 실행하고 있다.
   - 프로세스가 사용자와 상호작용하고 있는 액티비티에 연결되어진 서비스를

포함하고 있다.

- 프로세스가 생명주기 콜백(onCreate(), onStart() 또는 onDestroy()) 중 하나를 실행하고 있는 서비스Service 오브젝트를 가지고 있다.
- 프로세스가 onReceive() 메쏘드를 실행하고 있는 브로드캐스트 리시버BroadcastReceiver 오브젝트를 가지고 있다.

오직 최소한의 프로세스들이 어떤 주어진 시간에 존재할 것이다. 그것은 그것들을 계속 수행하지 못할 만큼 메모리가 작은 경우와 같은 최후의 상황에서만 강제 종료된다. 일반적으로, 그런 시점에 디바이스는 메모리 페이징 상태에 이르게 되어 사용자 인터페이스에 대한 응답을 유지하기 위해 요구되는 약간의 포어그라운드 프로세스를 강제 종료시킨다.

2. **비저블(visible) 프로세스**는 어떤 포어그라운드foreground 컴포넌트도 보유하지 않은 프로세스이다. 그러나, 그것은 여전히 스크린 상에서 사용자가 보는 것에 영향을 미칠 수 있다. 만약 아래와 같은 조건을 보유하고 있다면 이 프로세스는 비쥬얼한 것으로 간주된다.

   - 프로세스가 포어그라운드가 아니지만 여전히 사용자에게 보이는 액티비티(액티비티의 onPause() 메쏘드가 호출되었던 것)를 보유하고 있다. 예를 들어 만약 포어그라운드 액티비티가 그것의 뒤에 이전 액티비티가 보이는 것을 허용하는 다이얼로그인 경우에 이것은 발생할 수 있다.
   - 프로세스가 비저블 액티비티에 연결된 서비스를 포함하고 있다.

   비저블 프로세스는 매우 중요한 것으로 간주되며, 모든 포어그라운드foreground 프로세스가 계속 실행되도록 유지하기 위해 그것을 죽여야 하지 않는 한 강제 종료되지 않을 것이다.

3. **서비스(service) 프로세스**는 startService() 메쏘드를 사용해서 시작된 서비스를 실행하고 있으면서, 위의 두 개의 카테고리에 속하지 않는 프로세스이다. 비록 서비스 프로세스는 사용자가 보는 어떤 것과도 직접 결부되지는 않지만, 그것들은 일반적으로 사용자가 신경쓰는 (백그라운에서 mp3를 재생하거나 네트워크에서 데이터를 다운로드하는 것 같은) 것들을 하고 있다. 그래서 시스템은 모든 포어그라운드foreground 프로세스와 비저블visible 프로세스를 계속 유지할 정도로 메모리가 충분치 않는 경우를 제외하고는, 이 프로세스를 실행 상태로 유지한다.

4. **백그라운드(background) 프로세스**는 현재 사용자의 눈에 보이지 않는 액티비티(액티비티 오브젝트의 onStop() 메쏘드가 호출되었던 것)를 보유하고 있는 프로세스이다. 이들 프로세스는 직접적으로 사용자의 행위에 영향을 주지 않으며, 포어그라운드foreground, 비저블visible, 서비스service 프로세스를 위한 메모리 재요청을 위해 언제든 강제 종료될 수 있다. 통상적으로 많은 백그라운드background 프로세스가 존재하고 있으며, 그러므로 그것들은 가장 최근에 사용자에게 보여진 액티비티를 가진 프로세스가 가장 최후에 강제 종료될 수 있도록 하기 위한 LRU(Least Recently Used, 최근에 가장 적게 사용된) 목록으로 관리된다. 만약 액티비티가 정확한 생명주기 메쏘드들을 구현하고 있고 그것들의 현재 상태를 정확하게 포착하고 있다면, 그것이 속한 프로세스가 강제종료되는 것은 사용자에게 유해한 영향을 끼치지 않을 것이다.

5. **엠프티(empty) 프로세스**는 어떤 형태의 활성화된 컴포넌트도 보유하고 있지 않는 프로세스이다. 이러한 프로세스를 유지하는 유일한 이유는, 그 안에서 다음 번에 실행되어야 하는 컴포넌트의 구동시간을 개선하기 위한 캐쉬cache로써의 역할 때문이다. 시스템은 프로세스 캐쉬와 하단에 놓여있는 커널 캐쉬 간의 시스템 전역에 걸친 리소스들에 대한 균형을 유지하기 위해 가끔 이런 프로세스를 강제 종료시킨다.

안드로이드는 프로세스 내에 현재 활성화되어 있는 컴포넌트의 중요성에 기초해서, 가능한 한 가장 높은 등급에 프로세스를 위치지운다. 예를 들어 만약 프로세스가 하나의 서비스와 하나의 비저블visible 액티비티를 보유하고 있다면, 그 프로세스는 서비스 프로세스가 아니라 비저블visible 프로세스로 위치지워질 것이다.

추가적으로, 프로세스의 등급은 그것에 관계된 다른 프로세스로 인해서 높여질 수도 있다. 다른 프로세스를 지원하는 프로세스는, 그것이 지원하는 프로세스보다 결코 더 낮은 등급을 가져서는 안된다. 예를 들어 만약 프로세스 A에 속하는 컨텐트 프로바이더가 프로세스 B에 속하는 하나의 클라이언트를 지원하거나, 또는 프로세스 A에 속하는 서비스가 프로세스 B에 속하는 어떤 컴포넌트에 연결되어 있다면 적어도 프로세스 A는 항상 프로세스 B가 만큼 중요하게 간주되어 한다.

서비스를 실행하는 프로세스가 백그라운드 액티비티를 보유하고 있는 프로세스보다 더 높은 등급을 보유하기 때문에, 오랜시간 동안 실행되는 오퍼레이션은 (특

히 액티비티보다 더 오래 오퍼레이션이 지속될 것이라면) 간단하게 쓰레드를 생성하는 것보다 해당 오퍼레이션을 위한 서비스를 시작하는 것이 더 좋을 수 있다. 이런 예제들에는, 백그라운드background로 음악을 재생하는 것과 카메라로 찍은 사진을 웹 사이트로 업로드하는 것이 있다. 서비스를 사용하는 것은 액티비티에서 일어나는 것과 상관없이 그 오퍼레이션에 최소한 "서비스 프로세스" 우선순위를 보증할 것이다. 앞의 "브로드캐스트 리시버 생명주기" 섹션에서 언급했듯이, 이것이 바로 시간이 걸리는 오퍼레이션을 쓰레드로 구현하기보다는 오히려 서비스를 사용해서 구현해야 하는 같은 이유이다.

Section I 3

사용자 인터페이스

3.1. 개요

3.2. 레이아웃 선언하기

3.3. 메뉴 생성

3.4. 다이얼로그 생성

3.5. UI 이벤트 처리

3.6. 사용자에게 통보하기

3.7. 스타일과 테마 적용

3.8. 커스텀 컴포넌트 제작

3.9. 어댑터뷰를 사용한 데이터 바인딩

3.10. 일반적인 레이아웃 오브젝트

3.11. 안드로이드가 뷰를 그리는 방법

3. 사용자 인터페이스

개요

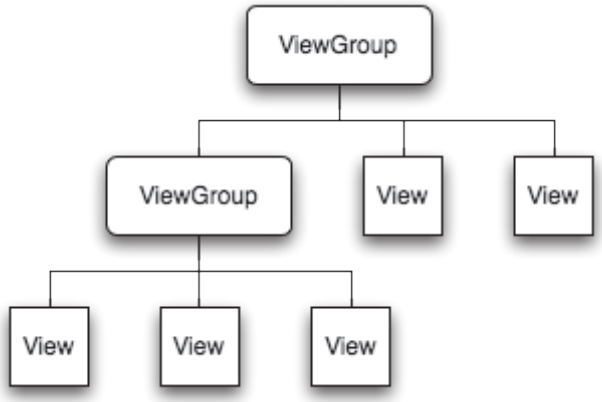
안드로이드 애플리케이션 안에는 사용자 인터페이스는 뷰View와 뷰그룹ViewGroup 오브젝트를 사용하여 만들어진다. 많은 종류의 뷰와 뷰그룹이 있는데, 그들 각각은 뷰View 클래스의 자손들descendant이다.

뷰 오브젝트들은 안드로이드 플랫폼 상의 사용자 인터페이스를 표현하는 기본 단위이다. 뷰 클래스는 텍스트 필드와 버튼처럼 완전하게 구현된 UI 오브젝트들을 제공하는 "위젯widget"이라 불리는 서브클래스에 대한 기반을 제공한다. ViewGroup 클래스는 리니어linear, 태뷸러tabular, 그리고 랠러티비relative 같은 다른 종류의 레이아웃 아키텍처를 제공하는 "레이아웃layouts"이라 불리는 서브클래스에 대한 기반을 제공한다.

뷰 오브젝트는 하나의 데이터 구조인데, 그 구조의 속성들에는 스크린의 특정 직사각형 영역에 대한 레이아웃 파라미터들과 컨텐트를 저장하고 있다. 뷰 오브젝트는 그 안에 있는 직사각형의 스크린 영역에 대해 측정measurement, 레이아웃, 그리기drawing, 포커스 변화, 스크롤링, 그리고 키/제스처key/gesture 상호작용을 제어한다. 또한 뷰는 사용자 인터페이스 내의 오브젝트처럼 사용자와의 상호작용 지점이며, 해당 상호작용 이벤트의 리시버receiver이다.

뷰 계층구조(Hierarchy)

안드로이드 플랫폼 상에서, 여러분은 다음 페이지에 나오는 그림에서 보여지듯이 뷰와 뷰그룹 노드들의 계층구조를 사용하는 액티비티의 UI를 정의한다. 이 계층구조는 여러분의 필요 형태에 따라 단순하거나 복잡할 수 있으며, 여러분은 안드로이드에 미리 정의된 위젯과 레이아웃의 집합 또는 여러분이 직접 만든 커스텀 뷰를 사용하여 그것을 만들 수 있다.



액티비티의 스크린에 렌더링을 위한 뷰계층구조 트리를 결합하기 위해서는 액티비티에서 setContentView() 메쏘드를 호출해야 하며, 해당 메쏘드에 뷰 계층구조의 루트root 노드 오브젝트에 대한 레퍼런스reference를 파라미터로 전달해야 한다. 안드로이드 시스템은 이 레퍼런스를 전달받으며, 트리tree에 대한 무효화invalidate, 측정measure, 그리기draw를 위해 그것을 사용한다. 계층구조의 루트root 노드는 자신의 자식들children에게 스스로를 그리도록 요청한다 – 순서대로 각각의 뷰그룹 노드는 그것의 자식child 뷰 각자가 스스로를 그리도록 호출하는 것에 대한 책임을 진다. 자식들children은 부모parent 안에서의 크기와 위치를 요청할 수 있다. 하지만 그 부모parent 오브젝트는 각각의 자식들children이 그곳에서 얼마나 커질 수 있는가에 대한 최종 결정을 한다. 안드로이드는 계층구조 트리tree의 최상위에서부터 순서대로 여러분의 레이아웃의 엘리먼트element를 분석parse해서 뷰들을 인스턴스화하며, 그리고 그것들을 그것들의 부모parent에 추가한다. 이것들은 순서대로 그려지기 때문에, 만약 엘리먼트의 위치가 겹쳐지는 것이 있다면 마지막에 그려진 것은 이전에 그 공간에 그려진 다른 것들 위에 놓이게 된다.

뷰 계층구조가 어떻게 측정(뷰의 너비와 높이를 계산)되고 그려지는 지에 대한 보다 상세한 논의에 대해서는 "안드로이드가 뷰를 그리는 방법"을 읽어라.

레이아웃(Layout)

여러분의 레이아웃을 정의하고 뷰계층구조를 표현하는 가장 일반적인 방법은 XML 레이아웃 파일을 사용하는 것이다. XML은 HTML과 매우 비슷하게 레이

아웃에 대한 사람이 읽을 수 있는 구조human-readable structure를 제공한다. XML의 각 엘리먼트는 뷰 또는 뷰그룹 오브젝트이다(또는 그들을 상속받은 자손들이다). 뷰 오브젝트는 트리tree에서 최하단의 노드이며, 뷰그룹 오브젝트는 트리에서 브랜치(branch, 하단에 뷰 또는 뷰그룹을 보유하는 가지)이다(앞의 뷰계층구조 그림을 보라).

XML 엘리먼트의 이름은 그것이 나타내는 Java 클래스 각각에 해당한다. 그러므로 〈TextView〉 엘리먼트는 여러분의 UI에 텍스트뷰TextView를 만들고, 〈LinearLayout〉 엘리먼트는 LinearLayout 뷰그룹을 만든다. 여러분이 레이아웃 리소스를 로드할 때, 안드로이드 시스템은 여러분의 레이아웃 내의 엘리먼트에 해당되는 클래스를 런타임 오브젝트로 초기화한다.

예를 들어 하나의 텍스트뷰와 하나의 버튼을 가지는 간단한 버티컬vertical 레이아웃은 다음과 같다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
               android:layout_width="fill_parent"
               android:layout_height="fill_parent"
               android:orientation="vertical" >
    <TextView android:id="@+id/text"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

LinearLayout 엘리먼트가 TextView와 Button 모두를 포함하고 있다는 것에 주목하라. 뷰 계층구조를 더 길고 더 복잡한 레이아웃으로 만들기 위해 여러분은 이 안에서 또 다른 LinearLayout (또는 다른 유형의 뷰그룹)을 가질 수 있다.

UI 레이아웃을 만드는 것에 대한 더 자세한 것은 3장. "사용자 인터페이스"의 "레이아웃 선언하기"를 보라.

> **Tip** 여러분은 또한 새로운 뷰와 뷰그룹 오브젝트를 동적으로 추가하기 위해, addView(View) 메쏘드를 사용해서 여러분의 Java 코드에서 뷰와 뷰그룹 오브젝트를 그릴 수 있다.

여러분의 뷰를 레이아웃하는 데에는 다양한 방법이 있다. 더 많은 그리고 다른 종류의 뷰그룹을 사용해서, 여러분은 자식child 뷰와 뷰 그룹을 다양한 방법으로 구조화할 수 있다. 안드로이드에는 (레이아웃이라 불리는) 미리 정의된 몇몇 뷰그룹이 존재한다. 그것은 LinearLayout, RelativeLayout, TableLayout, Grid Layout, 그리고 다른 것들을 포함하고 있다. 이들 각각은 단일한 레이아웃 파라미터 집합을 제공하는데, 이것은 자식child 뷰들의 위치와 레이아웃 구조를 정의하기 위해 사용된다.

레이아웃을 위해 사용된 다른 종료의 몇몇 뷰그룹들을 더 배우고자 한다면, "일반적 레이아웃 오브젝트"를 읽어라.

위젯(Widget)

위젯은 사용자와의 상호작용을 위한 인터페이스를 제공되는 뷰 오브젝트이다. 안드로이드는 버튼button, 체크박스checkbox, 그리고 텍스트 입력 필드text-entry field와 같은 풍부하게 구현된 위젯 집합set을 제공하며, 이것들로 여러분은 자신의 UI를 빠르게 만들 수 있다. 안드로이드에 의해 제공되는 어떤 위젯들은 날짜 설정date picker, 시계clock, 그리고 줌 컨트롤zoom control과 같이 보다 복잡하다. 그러나 여러분은 안드로이드 플랫폼에 의해 제공되는 다양한 종류의 위젯에 제약받지 않는다. 만약 여러분이 좀 더 커스터마이즈된 어떤 것을 하고자 한다면, 여러분 자신의 실행가능한actionable 엘리먼트를 만들어라. 여러분은 커스텀 뷰 오브젝트를 정의하거나 또는 이미 존재하는 위젯을 확장extending하고 조합combining해서 그것을 할 수 있다.

"커스텀 컴포넌트 만들기"에서 더 많은 것을 읽어라. 안드로이드에 의해 제공되는 위젯 목록에 대해서는 android.widget 패키지를 보라.

UI 이벤트

일단, 여러분이 몇몇 뷰와 위젯을 UI에 추가했다면, 액션을 수행하기 위해 그것들과 사용자 간의 상호작용에 대해 알고자 할 것이다. UI 이벤트를 통보받기 위해 여러분은 아래의 두 가지 중 한 가지를 하는 것이 필요하다.

- **이벤트 리스너를 정의하고, 그것을 뷰에 등록하기.** 이것이 일반적으로 이벤트를 청취하는 방법이다. 뷰 클래스는 각각 On<something>Listener 형태로 명명된 네스티드nested 인터페이스 메쏘드들의 집합을 포함하고 있으며, 그들 각각은 On<something>() 형태로 호출되는 콜백 메쏘드를 가진다. 예를 들어 뷰에 대한 "클릭click"을 제어하기 위한 View.OnClickListener, 뷰에서의 터치 스크린 이벤트를 제어하기 위한 View.OnTouchListener, 그리고 뷰에서의 키 입력을 제어하기 위한 View.OnKeyListener가 그것이다. 그러므로 만약 버튼이 선택된 것과 같이 그것이 "클릭click"되었을 때, 그것이 뷰에 통보되길 원한다면, OnClickListener 구현하고 그것의 onClick() 콜백 메쏘드(이곳에서 클릭에 대한 액션을 수행한다.)를 정의하고 setOnClickListener()를 사용해서 뷰에 그것을 등록하라.
- **뷰에 대해 이미 존재하는 콜백 메쏘드를 오버라이드 하기.** 이것은 자신의 뷰 클래스를 구현하고, 그 안에서 발생하는 특정 이벤트를 청취하길 원할 때 사용하는 방법이다. 제어될 수 있는 예제 이벤트로는 스크린이 터치되었을 때(onTouchEvent()), 트랙볼이 움직였을 때(onTrackballEvent()), 또는 디바이스의 키가 입력됐을 때(onKeyDown()) 등이 있다. 이것은 커스텀 뷰 내부의 각각의 이벤트에 대한 디폴트 동작으로 정의될 수도 있으며, 몇 몇 다른 자식child 뷰에 이벤트를 전달하는 것을 결정할 수도 있다. 이것은 다시 뷰 클래스로 콜백된다. 그러므로, 그것을 정의할 수 있는 유일한 기회는 커스텀 컴포넌트를 만드는 경우이다.

이후 나오는 "UI 이벤트 처리" 섹션에서 뷰와 사용자 간의 상호작용을 제어하는 것에 대해 계속 읽어라.

메뉴(Menu)

애플리케이션 메뉴는 애플리케이션의 UI 중의 또 다른 중요한 일부이다. 메뉴는 애플리케이션 기능과 셋팅setting을 보여주는 신뢰할 만한 인터페이스를 제공한다. 가장 일반적인 애플리케이션 메뉴는 디바이스 상에서 MENU 키를 누름으로써 나타나는 것이다. 하지만 여러분은 사용자가 아이템을 누르고 있을 때 보여질 수 있는 컨텍스트 메뉴Context Menus 또한 추가할 수 있다.

메뉴 또한 뷰 계층구조를 사용해서 구조화된다. 하지만, 이러한 구조를 직접 정의할 필요는 없다. 대신에, 액티비티activity에 대한 onCreateOptionsMenu() 또는 onCreateContextMenu() 콜백 메쏘드를 정의하고, 메뉴에 포함되길 원하는 아이템들을 선언하면 된다. 그러면 적절한 시점에, 안드로이드가 자동으로 메뉴를 위해 필요한 뷰 계층구조를 만들 것이고 그곳에 메뉴 아이템 각각을 그릴 것이다.

메뉴는 또한 자기 자신의 이벤트를 처리한다. 그러므로 메뉴 안의 아이템에 대한 이벤트 리스너를 등록할 필요는 없다. 메뉴 안의 아이템이 사용자에 의해 선택되면, 안드로이드 프레임워크framework는 onOptionsItemSelected() 또는 onContextItemSelected() 메쏘드를 호출해 준다.

그리고 애플리케이션 레이아웃과 같이, XML 파일로 메뉴에 대한 아이템을 선언하는 것도 선택사항으로 가능하다.

더 많이 배우려면 이후 나오는 “메뉴 생성하기”를 읽어라.

고급 주제(Advanced Topics)

여러분이 일단 사용자 인터페이스를 생성하는 기초원리를 파악했다면, 보다 복잡한 애플리케이션 인터페이스를 생성하는 것에 대한 약간의 고급 기능을 살펴볼 수도 있다.

어댑터(Adapter)

가끔 여러분은 코드상에 직접 기입hard-coded될 수 없는 몇 가지 정보를 보유하는

뷰그룹을 배치하길 원할 수 있다. 즉, 외부 데이터 소스에 뷰를 바인드하고자 할 수 있다. 이것을 하기 위해 사용하는 것이, 뷰그룹과 각각의 자식child 뷰를 초기화하고 어댑터Adapter에서 제공되는 데이터로 그것을 채우는 어댑터뷰AdapterView이다.

어댑터뷰AdapterView 오브젝트는 주어진 어댑터Adapter 오브젝트를 기반으로 그것의 자식child 뷰를 결정하는 뷰그룹ViewGroup의 구현implementation 형태이다. 어댑터Adapter는 데이터 소스(아마도 외부 문자열 배열)와 그것을 표현하는 어댑터뷰AdapterView사이의 전달수단 같은 역할을 한다. 커서Cursor로부터 데이터베이스 데이터를 읽어오기 위한 커서 어댑터CursorAdapter, 또는 임의의 배열로부터 읽어오기 위한 어레이 어댑터ArrayAdapter와 같은 특정 업무를 위한 몇 가지 어댑터Adapter 클래스에 대한 구현 형태들이 있다.

뷰를 채우는 어댑터Adapter의 사용에 대해 더 많이 알고자 한다면, "어댑터뷰AdapterView에 데이터 바인딩하기"를 읽어라.

스타일과 테마(Theme)

아마도 여러분은 표준 위젯의 외형에 대해 만족하지 못할 수도 있다. 그것을 개선하기 위해 여러분 자신의 몇 가지 스타일과 테마를 생성하는 것이 가능하다.

- 스타일은 레이아웃에 있는 개별적인 엘리먼트 단위에 적용할 수 있는 하나 또는 그 이상의 포맷팅 애트리뷰트의 집합이다. 예를 들어 여러분은 임의의 텍스트 사이즈와 컬러를 명세하는 스타일을 정의해서, 특정 뷰 엘리먼트에만 그것을 적용할 수 있다.
- 테마는 애플리케이션 내의 모든 액티비티 또는 하나의 액티비티 단위에 적용할 수 있는 하나 또는 그 이상의 포맷팅 애트리뷰트의 집합이다. 예를 들어 여러분은 윈도우 프레임과 패널 백그라운드에 대한 특정 컬러를 설정하고, 메뉴에 대한 텍스트 크기와 컬러를 설정하는 테마를 정의할 수 있다. 그러면, 이 테마는 특정 액티비티 또는 전체 애플리케이션에 적용될 것이다.

스타일과 테마는 리소스들이다. 안드로이드는 여러분이 사용할 수 있는 약간의 디폴트 스타일과 테마 리소스를 제공하며, 또한 자기만의 커스텀 스타일과 테마 리소스를 선언할 수도 있다.

"스타일과 테마 적용하기" 문서에서 스타일과 테마를 사용하는 것에 대해 더 배워라.

레이아웃 선언하기

레이아웃은 액티비티 내의 사용자 인터페이스에 대한 아키텍처이다. 그것은 레이아웃 구조를 정의하고, 사용자에게 보여지는 모든 엘리먼트를 보유한다. 여러분은 두 가지 방법으로 레이아웃을 선언할 수 있다.

- **XML로 UI 엘리먼트를 선언하기.** 안드로이드는 위젯과 레이아웃과 같은, 뷰 클래스와 서브클래스에 해당되는 간결한 XML 어휘를 제공한다.
- **런타임에서 레이아웃 엘리먼트를 인스턴스화 하기.** 애플리케이션은 프로그램적으로 뷰와 뷰그룹 오브젝트를 만들 수 있다(그리고 그들의 속성들을 조작할 수 있다).

안드로이드 프레임워크는 애플리케이션 UI를 선언하는 것, 그리고 그것을 관리하는 것에 대한 메쏘드를 제공하며, 그것들 중 어느 한 쪽, 또는 둘 다를 사용할 수 있는 유연성 또한 제공한다. 예를 들어 여러분은 엘리먼트와 엘리먼트의 속성들로 표현되는 스크린 엘리먼트들을 포함하는 XML로 애플리케이션의 디폴트 레이아웃을 선언할 수 있다. 그런 다음에 런타임에서 XML 안에 선언된 스크린 오브젝트의 상태를 수정하도록 애플리케이션에 코드를 추가할 수 있다.

이클립스에 대한 안드로이드 개발 툴(Android Development Tools, ADT) 플러그인은 이러한 XML에 대한 레이아웃 미리보기를 제공한다 – XML 파일을 오픈해서 Layout 탭을 선택하라.

여러분은 레이아웃을 디버깅하기 위해 계층구조 뷰어 툴(Hierarchy Viewer tool) 또한 사용할 수 있다 – 그것은 여러분이 에뮬레이터 또는 디바이스에서 디버깅하는동안 레이아웃 속성 값들을 보여주며, 패딩/마진 정보를 가지는 와이어프레임wireframe과 전체 스크린 뷰를 그린다.

layoutopt 툴은 여러분이 빠르게 레이아웃과 계층구조의 비효율성 또는 그밖의 문제들을 분석할 수 있게 해 준다.

XML로 UI를 선언하는 것의 장점은, 애플리케이션의 프리젠테이션과 그것의 동

작을 제어하는 코드를 더 잘 구분할 수 있게 한다는 것이다. 이 경우, UI에 대한 기술description은 애플리케이션 코드 외부에 있게 된다. 이것은 소스코드를 수정하고 재컴파일recompile하지 않고도 그것을 수정 또는 개조할 수 있다는 것을 의미한다. 예를 들어 여러분은 서로 다른 스크린 오리엔테이션, 서로 다른 디바이스 스크린 사이즈, 그리고 서로 다른 언어에 대한 별도의 XML 레이아웃들을 만들 수 있다. 더불어 XML로 레이아웃을 선언하는 것은, UI 구조가 보다 쉽게 눈에 보이도록 한다. 그러므로 문제점을 디버깅하기 더 쉬어진다. 그와 같은 입장에서, 이 문서는 XML로 레이아웃을 선언하는 방법을 가르치는 것에 집중한다. 만약 여러분이 런타임에서 뷰View 오브젝트를 인스턴스화 하는 것에 관심이 있다면, 뷰그룹ViewGroup과 뷰View 클래스 레퍼런스를 참고하기 바란다.

일반적으로, UI 엘리먼트를 선언하기 위한 XML 어휘는 클래스과 메쏘드의 구조와 이름에 밀접하게 연관된다. XML에서 엘리먼트의 이름은 클래스 이름에 대응되며, 애트리뷰트 이름은 메쏘드에 대응된다. 사실, 이러한 대응관계는 여러분이 XML 애트리뷰트가 어떤 클래스 메쏘드에 대응하는지, 또는 주어진 XML 엘리먼트에 대응하는 클래스가 무엇인지를 추측할 수 있을 정도로 대부분의 경우 매우 직접적으로 연관된다. 하지만 모든 어휘가 정확하게 클래스 또는 메쏘드 명과 일치하지 않다는 점에 주의하라. 몇 가지 경우에는 사소한 이름 차이가 있다. 예를 들어 EditText 엘리먼트는 EditText.setText()에 대응되는 text 애트리뷰트를 가진다.

Tip "일반적인 레이아웃 오브젝트"에서 다른 레이아웃 타입에 대해 더 많이 배워라. 또한, Hello Views 튜토리얼 가이드에 다양한 레이아웃을 만드는 튜토리얼 모음이 있다.

XML 작성

안드로이드의 XML 어휘를 사용해서, 여러분은 HTML로 웹페이지를 생성하는 것과 비슷한 방법으로 네스티드nested 엘리먼트들의 시리즈로 UI 레이아웃과 그것들이 포함하는 스크린 엘리먼트들을 쉽게 디자인할 수 있다.

> 편의 상, 클래스와 관련된 UI에 대한 API 레퍼런스 문서는 클래스 메쏘드에 대응되는 사용가능한 XML 애트리뷰트를 나열하고 있다. 그것은 상속된 애트리뷰트를 포함한다. XML 파일의 포맷과 더불어, 사용가능한 XML 엘리먼트와 애트리뷰트에 대해 더 많이 배우려면 4장. "리소스와 에셋"의 "사용가능한 리소스 타입" 중 "레이아웃 리소스"를 보라.

각각의 레이아웃 파일은 정확하게 하나의 루트root 엘리먼트를 포함해야 하며, 그것은 뷰View 또는 뷰그룹ViewGroup 오브젝트여야 한다. 일단 여러분이 루트root 엘리먼트를 정의했다면, 레이아웃을 정의하는 뷰 계층구조를 순차적으로 만들어내는 자식child 엘리먼트로 추가적인 레이아웃 오브젝트 또는 위젯을 추가할 수 있다. 예를 들어 여기에 TextView와 Button을 보유한 버티컬 LinearLayout을 사용하는 XML 레이아웃이 아래에 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
               android:layout_width="fill_parent"
               android:layout_height="fill_parent"
               android:orientation="vertical" >
    <TextView android:id="@+id/text"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

XML로 레이아웃을 선언한 뒤, 그것이 적절하게 컴파일 되도록 여러분의 안드로

이드 프로젝트의 res/layout/ 디렉토리에 .xml 확장자를 가진 파일을 저장하라. 우리는 앞에서 보여진 각각의 애트리뷰트들에 대해서는 조금 뒤에 논의할 것이다.

XML 리소스 로드(Load)

여러분이 애플리케이션을 컴파일할 때, 각각의 XML 레이아웃 파일은 뷰 리소스로 컴파일된다. 여러분은 Activity.onCreate() 콜백을 구현하는 곳에서, 애플리케이션 코드에서 레이아웃 리소스를 로드해야 한다. 이것은 setContentView()를 호출하고 그 메쏘드에 R.layout.layout_file_name 형태의 레이아웃 리소스에 대한 레퍼런스를 파라미터로 전달함으로써 가능하다. 예를 들어 만약 여러분의 XML 레이아웃이 main_layout.xml로 저장되어 있다면, Activity는 다음과 같이 그것을 로드할 수 있다.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView.(R.layout.main_layout);
}
```

액티비티 내의 onCreate() 콜백 메쏘드는 그 액티비티가 시작됐을 때, 안드로이드 프레임워크에 의해 호출된다(이것에 대한 더 자세한 것을 위해선, 2장. "애플리케이션 기초" 안의 "생명주기"를 보라).

애트리뷰트(Attribute)

모든 뷰와 뷰그룹 오브젝트는 그것들 자신의 다양한 XML 애트리뷰트를 지원한다. 몇 가지 애트리뷰트는 뷰 오브젝트(예를 들어 TextView는 textSize 애트리뷰트를 지원한다)에 국한된다. 그러나 이들 애트리뷰트는 이 클래스를 상속받는 임의의 뷰 오브젝트에게도 상속된다. (id 애트리뷰트 같은) 몇 가지는 모든 뷰 오브젝트에서 보편적으로 사용가능하다. 왜냐하면, 그것들이 루트root 뷰View 클래스로부터 상속되었기 때문이다. 그리고 그 밖의 애트리뷰트로는 "레이아웃 파라미터(layout parameters)"가 중요하다. 그것은 뷰 오브젝트에 대한 특정 레이아웃 오리엔테이션을 설명하는 애트리뷰트이며, 이것은 해당 오브젝트의 부모parent 뷰그룹 오브젝트에 의해 정의된다.

ID

어떤 뷰 오브젝트든지 트리 안에서 그 뷰를 유일하게 식별하기 위한 뷰와 결합된 정수integer ID를 가질 수 있다. 애플리케이션이 컴파일될 때, 이 ID는 정수integer로써 레퍼런스된다. 하지만 일반적으로 ID는 레이아웃 XML 파일에서 id 애트리뷰트에 있는 문자열로 할당된다. 이것은 (뷰 클래스에 의해 정의된) 모든 뷰 오브젝트에게 공통으로 사용되는 XML 애트리뷰트이며, 여러분은 그것을 매우 자주 사용할 것이다. XML 태그에서 ID에 대한 구문은 다음과 같다.

```
android:id="@+id/my_button"
```

문자열의 시작 위치에 있는 @심볼(at-symbol)는, XML 구문분석기parser가 그것을 분석하고 ID 문자열의 나머지 부분을 확장(expand, 역주: 치환하기 위해 분리해 냄)해서, 그것을 ID 리소스로써 인식해야 한다는 것을 지시한다. +심볼(plus-symbol)은 (R.java 파일에 있는) 리소스들에 새로운 리소스 이름이 생성되어 추가되어야 한다는 것을 의미한다. 안드로이드 프레임워크에 의해 제공되는 다른 많은 ID 리소스들이 있다. 안드로이드 리소스 ID를 레퍼런스할 때는, +심볼을 필요로 하지 않는다. 하지만 다음과 같이 android 패키지 네임스페이스를 추가해야 한다.

```
android:id="@android:id/empty"
```

그곳에 android 패키지 네임스페이스가 있으면, 안드로이드는 이제 ID를 로컬 리소스 클래스가 아니라 android.R 리소스 클래스에서 레퍼런스한다.

뷰를 생성하고 애플리케이션에서 그것을 레퍼런스 하는 일반적인 패턴은 다음과 같다.

1. 레이아웃 파일 안에서 뷰/위젯을 정의하고, 그것에 고유한 ID를 할당한다.

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

2. 그런 다음, 뷰 오브젝트의 인스턴스를 생성하고 레이아웃에서 그것을 캡쳐한다(일반적으로 onCreate() 메쏘드에서 이 일을 한다).

```
Button myButton = (Button) findViewById(R.id.my_button);
```

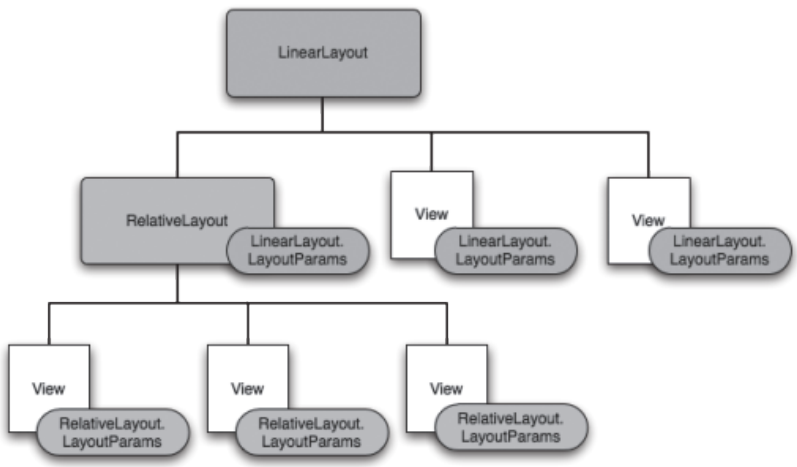
뷰 오브젝트에 대한 ID를 정의하는 것은 RelativeLayout을 생성할 때 중요하다. RelativeLayout에서, 형제sibling뷰들은 그것들의 레이아웃을 다른 형제sibling뷰에 상대적으로 정의할 수 있다. 그것은 고유한 ID에 의해 레퍼런스된다.

ID는 전체 트리에 걸쳐 유일할 필요는 없으나, 여러분이 검색하는 트리의 영역 내에서는 유일해야 한다(검색하는 트리 영역은 종종 전체 트리가 될 수 있다. 그러므로 가능하면 완벽하게 유일하게 되어 있는 것이 최선이다).

레이아웃 파라미터(Layout Parameters)

layout_something 형식으로 명명된 XML 레이아웃 애트리뷰트는, 뷰에 대한 레아아웃 파라미터를 정의한다. 이것은 뷰그룹 내에 뷰를 적절하게 위치하게 한다.

모든 뷰그룹 클래스는 ViewGroup.LayoutParams를 확장하는 네스티드nested 클래스를 구현한다. 이 서브클래스는 뷰그룹에 대해 알맞게, 각각의 자식child 뷰에 대한 크기와 위치를 정의하는 속성 타입을 포함한다. 아래의 그림에서 여러분이 볼 수 있듯이, 부모parent 뷰그룹은 각각의 자식child 뷰(뷰그룹을 포함함)에 대한 레이아웃 파라미터를 정의한다.



모든 LayoutParams 서브클래스는 그 값을 설정하기 위한 자기 자신의 구문을 가진다는 것에 주의하라. 각각의 자식child 엘리먼트는 또한 자신의 자식들children에 대해 다른 LayoutParams을 정의할 수 있지만, 그것의 부모parent에 알맞게 LayoutParams을 정의해야 한다.

모든 뷰 그룹들은 너비와 높이(layout_width와 layout_height)를 포함하며, 각각의 뷰는 그것들을 정의하는 것이 필수사항이다. 많은 LayoutParams은 또한 마진margin과 보더border를 선택사항으로 포함한다(역주 : 이 문장에서 보더border는 패딩padding으로 바뀌어야 할 듯 함). 정확한 수치로 이 너비와 높이를 지정하는 것이 가능하다. 하지만, 아마도 이것은 자주하고 싶지는 않을 것이다. 대개는, (wrap_content와 fill_parent 값을 사용해서) 뷰가 자신이 보유한 컨텐트에 필요한 크기에 맞춰지게 하거나 또는 그것의 부모parent 뷰그룹이 허용하는 한 커지도록 하는 것이 사용될 것이다. 사용가능한 치수measurement 타입은 4장. "리소스와 에셋"의 "사용가능한 리소스"에 정의되어 있다.

레이아웃 포지션(Position)

뷰의 기하학적 모습은 직사각형이다. 뷰는 왼쪽left과 상단top 좌표의 쌍으로 표현되는 위치와 너비와 높이로 표시되는 2차원dimension을 가진다. 그 위치와 크기dimension에 대한 단위는 픽셀pixel이다.

메쏘드 getLeft()와 getTop()을 호출함으로써, 뷰의 위치를 얻는 것이 가능하다. 전자는 뷰를 나타내는 직사각형의 왼쪽, 또는 X 좌표를 리턴한다. 후자는 뷰를 나타내는 직사각형의 상단, 또는 Y 좌표를 리턴한다. 이 메쏘드 모두는 그것의 부모parent에 대해 상대적인 뷰의 위치를 리턴한다. 예를 들어 getLeft()가 20을 리턴하면, 그것이 의미하는 바는 해당 뷰가 자신의 직접적인 부모parent의 왼쪽 가장자리에서 20 픽셀 오른쪽에 위치한다는 것이다.

추가적으로 몇 가지 편리한 메쏘드가 불필요한 계산, 즉 getRight()와 getBottom()를 피하기 위하여 제공된다. 이 메쏘드들은 뷰를 표시하는 직사각형의 우측과 하단 가장자리의 좌표를 반환한다. 예를 들어 getRight()를 호출하는 것은 다음과 같은 계산과 비슷하다.

```
getLeft() + getWidth()
```

크기(Size), 패딩(Padding) 그리고 마진(Margin)

뷰의 크기는 너비와 높이로 표현된다. 뷰는 실제로 두 쌍의 너비와 높이 값을 가지고 있다.

첫 번째 쌍은 측정된 너비measured width와 측정된 높이measured height로 일컬어진다. 이 크기는 하나의 뷰가 자신의 부모parent내에서 얼마나 커지고자 하는지를 정의한다. 측정된measured 크기는 getMeasuredWidth()와 getMeasuredHeight()를 호출해서 얻을 수 있다.

두 번째 쌍은 너비width와 높이height로 간단하게 일컬어진다. 또는 가끔은 드로잉 너비drawing width 또는 드로잉 높이drawing height로 일컬어지기도 한다. 이 크기는 레이아웃 이후 그리는 시점의 실제 스크린 상의 뷰의 크키를 정의한다. 그러나 이 값은 그래서는 안되지만, 측정된 너비measured width와 측정된 높이measured height와 다른 것을 가질 수도 있다. 너비width와 높이height는 getWidth()와 getHeight()를 호출해서 얻을 수 있다.

뷰는 자신의 크기dimension를 측정하기 위해, 패딩padding을 계산에 넣는다. 패딩은 뷰의 왼쪽, 상단, 오른쪽, 그리고 하단 영역에 대한 픽셀로 표현된다. 패딩은 특정 픽셀의 양만큼 뷰의 컨텐트 오프셋offset을 지정하기 위해 사용될 수 있다. 예를 들어 왼쪽 패딩 2는 왼쪽 가장자리에서 오른쪽으로 2 픽셀만큼 뷰 컨텐츠를 밀 것이다. 패딩은 setPadding(int,int,int,int) 메쏘드를 사용하여 설정될 수 있으며, getPaddingLeft(),getPaddingTop(),getPaddingRight(),그리고 getPaddingBottom()을 호출함으로써 질의query된다.

비록 뷰가 패딩을 정의할 수 있음에도 불구하고, 그것은 마진margin에 대한 어떤 지원도 제공하지 않는다. 하지만 뷰그룹은 그러한 지원을 제공한다. 추가 정보를 위하여 ViewGroup 과 ViewGroup.MarginLayoutParams를 참조하라.

크기Dimensions에 관한 더 많은 정보는 4장. “리소스와 에셋”의 “사용가능한 리소스 타입”중 크기 값을 보라.

메뉴 생성

메뉴는 모든 애플리케이션의 중요한 부분이다. 그들은 애플리케이션 기능과 셋팅setting을 보여주는 익숙한 인터페이스를 제공한다. 안드로이드는 다양한 상황에 대한 표준화된 애플리케이션 메뉴를 제공하기 위해, 개발자를 위한 손쉬운 프로그래밍 인터페이스를 제공한다.

안드로이드는 세 가지의 기본적 유형의 애플리케이션 메뉴를 제공한다.

옵션(Option) 메뉴

이것은 액티비티를 위한 메뉴 아이템의 기본적인 집합set이다. 그것은 디바이스 MENU 키를 누르는 것에 의해 나타난다. 옵션 메뉴Options Menu 안에는 두 가지 그룹의 메뉴 아이템이 있다.

아이콘 메뉴(Icon Menu)

이것은 MENU 키의 눌렀을 때, 스크린 하단에 최초로 보여지는 아이템들의 모음이다. 그것은 최대 여섯 개의 메뉴 아이템을 지원한다. 이것이 아이콘을 지원하는 유일한 메뉴 아이템이며, 또한 체크박스나 라디오 버튼을 지원하지 않는 유일한 메뉴 아이템이기도 하다.

확장 메뉴(Expanded Menu)

이것은 아이콘 메뉴의 메뉴 아이템 “More”를 선택했을 때 보여지는 수직적 아이템 목록이다. 그것은 아이콘 메뉴가 과적재over-loaded 될 때만 존재하고, 여섯 번째 옵션 메뉴Option Menu 아이템과 나머지로 구성되어 있다.

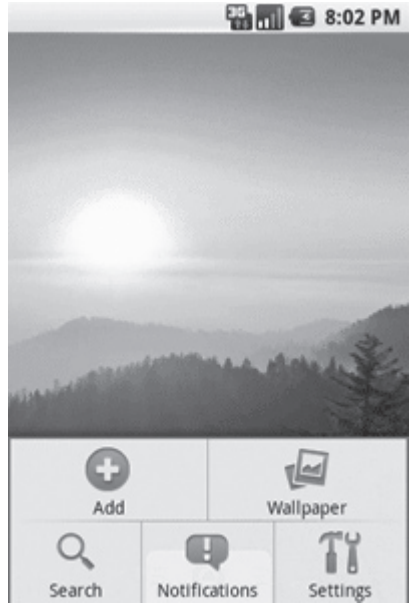
컨텍스트(Context) 메뉴

이것은 여러분이 (리스트 아이템과 같은) 뷰를 길게 누르는 것long-press을 수행할 때 나타날 수 있는 스크린 상에 떠있게 되는floating 메뉴 아이템 목록이다.

서브메뉴(Submenu)

이것은 옵션 메뉴Options Menu 또는 컨텍스트 메뉴Context Menu 안의 아이템에 의해 보여지게 되는 스크린 상에 떠있는floating 메뉴 아이템 목록이다. 서브메뉴Submenu 아이템은 그 안에 서브메뉴Submenu를 지원할 수 없다.

옵션(Option) 메뉴



옵션 메뉴는 디바이스 MENU 키를 누름으로써 오픈된다. 오픈될 때 아이콘icon 메뉴가 보여지는데, 그것은 최초 여섯 개의 메뉴 아이템을 보유하고 있다. 만약에 여섯 개 이상의 아이템이 옵션 메뉴에 추가되면, 아이콘icon 메뉴 안에 맞춰지지 않는 메뉴 아이템은 "More" 메뉴 아이템을 통해 확장expanded 메뉴에서 표시된다. 확장expanded 메뉴는 여섯 개 이상의 아이템이 있을 때 자동으로 추가된다.

옵션 메뉴는 여러분이 기본적인 애플리케이션의 기능과 임의로 필요한 네비게이션 아이템(예를 들어 홈스크린이나 애플리케이션 설정으로 이동)을 포함시켜야 하는 곳이다. 여러분은 또한 토픽topic을 구성하고 여분의 메뉴 기능을 포함하기 위해 서브메뉴Submenus를 추가할 수 있다.

이 메뉴가 최초 시점에 오픈될 때, 안드로이드 시스템은 해당 액티비티의 onCreateOptionsMenu() 콜백 메쏘드를 호출할 것이다. 여러분의 액티비티에서 이 메쏘드를 오버라이드하고, 그 메쏘드에 파라미터로 전달되는 메뉴Menu 오브젝트를 채워라populate. 여러분은 XML로 정의된 메뉴 리소스를 인플레이트inflate시키거나 또는 여러분이 메뉴 안에 보유하고 싶은 각각의 아이템에 대해 add()를 호출함으로써 메뉴를 채울 수 있다. 이 메쏘드는 메뉴아이템MenuItem을 추가하며, 여러분에게 새롭게 생성된 그 오브젝트를 리턴한다. 여러분은 아이콘, 키보드 숏컷shortcut, 인텐트, 그리고 아이템에 대한 또 다른 설정과 같은 추가적인 속성들을 설정하기 위해 리턴된 메뉴아이템MenuItem을 사용할 수 있다.

여러 개의 add() 메쏘드들이 있다. 보통 여러분은 itemId 아규먼트argument를 받아들이는 것을 사용하고 싶을 것이다. 이것은 여러분이 콜백 메쏘드에서 아이템을 식별할 수 있게 하는 고유한 정수integer이다.

옵션 메뉴에서 메뉴아이템이 선택될 때, onOptionsItemSelected() 메쏘드가 여러분의 액티비티에서 콜백 될 것이다. 이 콜백은 선택된 메뉴아이템MenuItem을 여러분에게 파라미터로 전달한다. 여러분은 add() 메쏘드에 할당된 정수integer를 리턴해 주는 getItemId()를 사용해서 해당 itemId를 요청함으로써 아이템을 식별할 수 있다. 일단 여러분이 메뉴 아이템을 식별하면, 여러분은 적절한 액션을 취할 수 있다.

이러한 프로시저의 예제가 여기에 있다. 액티비티 안에서 옵션 메뉴 를 생성하고 아이템 선택을 제어하는 예제이다.

```
/* Creates the menu items */
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, MENU_NEW_GAME, 0, "New Game");
    menu.add(0, MENU_QUIT, 0, "Quit");
    return true;
}

/* Handles item selections */
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case MENU_NEW_GAME:
        newGame();
        return true;
    case MENU_QUIT:
        quit();
        return true;
    }
    return false;
}
```

위의 예제에서 사용된 add() 메쏘드는 groupId, itemId, order, 그리고 title 이라는 네 개의 아규먼트를 가지고 있다. groupId는 여러분이 이 메뉴 아이템을 다른 아이템의 그룹과 결합시키는 것을 가능하게 한다(Menu groups에 대한 자세한 것은 아래에 있다) – 위의 예제에서, 우리는 그것을 무시한다. itemId는 우리가 다음 번 콜백에서 아이템을 식별할 수 있도록 메뉴아이템MenuItem에 부여한 고유한 정수integer이다. order는 우리가 아이템의 표시 순서를 정의하는 것을 가능하게 한다 – 디폴트로 그것들은 추가된 순서대로 표시된다. title은 당연히 메뉴 아이템

에 제공하는 이름이다(이것은 또한 문자열 리소스일 수도 있으며, 우리는 여러분이 더 쉽게 로컬라이제이션localization을 할 수 있는 이 방법을 권장한다).

Tip 만약 여러분이 타이틀과 함께 그룹화될 수 있는 몇 개의 메뉴 아이템을 가지고 있다면, 서브메뉴로 그것들을 배치하는 것을 고려하라.

아이콘 추가하기(Adding icons)

아이콘은 또한 setIcon()를 사용해서 아이콘 메뉴에 나타나는 아이템에 추가될 수 있다. 예를 들어 다음과 같다.

```
menu.add(0, MENU_QUIT, 0, "Quit")
    .setIcon(R.drawable.menu_quit_icon);
```

메뉴 수정하기(Modifying the menu)

만약 여러분이 가끔 오픈된 옵션 메뉴를 다시 작성하고자 한다면, 메뉴가 오픈될 때마다 호출되는 onPrepareOptionsMenu() 메쏘드를 오버라이드 하라. 이것은 onCreateOptionsMenu() 콜백처럼 여러분에게 메뉴Menu 오브젝트를 파라미터로 전달할 것이다. 이것은 여러분이 애플리케이션 또는 게임game의 현재 상태에 의존적인 메뉴 옵션들을 추가하거나 제거하고자 할 때 유용하다.

NOTE

메뉴에 있는 아이템을 바꿀 때, 현재 선택된 아이템에 기반해서 하는 것은 나쁜 사례이다. 터치 모드일 때는 선택된 (또는 포커스 된) 아이템이 없다는 것을 유념하라. 대신 여러분이 UI에 있는 특정 아이템에 기반한 기능을 제공하고 싶을 때는, 그런 동작을 위한 컨텍스트 메뉴(Context Menu)를 이용해야 한다.

컨텍스트(Context) 메뉴

안드로이드 컨텍스트 메뉴는 PC에서 마우스 우측 버튼을 클릭할 때 보여지는 메뉴와 개념적으로 유사하다. 뷰가 컨텍스트 메뉴에 등록되어 있을 때, 그 오브젝트를 "길게 누르는 것long-press"(대략 2초 정도 누르고 있으면)을 수행하면 그 아이템과 관련된 기능을 제공하는 떠 있는floating 메뉴가 나타날 것이다. 컨텍스트 메뉴는 어떤

뷰 오브젝트에든 등록될 수 있다. 하지만 그것은 리스트뷰ListView 아이템이 눌려질 때, 그것의 배경색을 바꿈으로써 컨텍스트 메뉴의 존재를 쉽게 나타내 주는 리스트뷰Listview에서 매우 자주 사용된다(폰의 컨택contact 리스트의 아이템은 이런 기능의 예제를 제공한다).

NOTE 컨텍스트 메뉴 아이템은 아이콘 또는 숏컷(shortcut) 키를 지원하지 않는다.

컨텍스트 메뉴를 만들기 위해, 여러분은 액티비티의 컨텍스트Context 메뉴 콜백 메쏘드인, onCreateContextMenu()와 onContextItemSelected()를 오버라이드 해야 한다. onCreateContextMenu() 콜백 메쏘드 안에서, 여러분은 add() 메쏘드들 중 하나를 사용하거나 또는 XML로 정의된 메뉴 리소스를 인플레이트inflate함으로써 메뉴 아이템들을 추가할 수 있다. 그리고 나서, registerForContextMenu()를 사용해서 뷰View에 대한 컨텍스트 메뉴ContextMenu를 등록하라.

예를 들어 여기에 리스트 상의 각 노트에 대해 컨텍스트 메뉴를 추가하는 노트패드 애플리케이션에서 사용될 수 있는 약간의 코드가 있다.

```
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  menu.add(0, EDIT_ID, 0, "Edit");
  menu.add(0, DELETE_ID, 0,  "Delete");
}
public boolean onContextItemSelected(MenuItem item) {
  AdapterContextMenuInfo info
      = (AdapterContextMenuInfo) item.getMenuInfo();
  switch (item.getItemId()) {
  case EDIT_ID:
    editNote(info.id);      return true;
  case DELETE_ID:
    deleteNote(info.id);     return true;
  default:
    return super.onContextItemSelected(item);
  }
}
```

onCreateContextMenu()에서, 우리는 메뉴아이템MenuItems을 추가할 ContextMenu을 얻을 수 있을 뿐만 아니라 선택된 뷰와 선택된 오브젝트에 대한 추가적인 정보를 제공하는 ContextMenuInfo 오브젝트를 얻게 된다. 위 예제에서는, onCreateContextMenu()에서 아무런 특별한 것도 수행되지 않는다 – 다만 통상적인 아이템 한 쌍이 추가되었다. onContextItemSelected() 콜백에서, 우리는 MenuItem으로 부터 AdapterContextMenuInfo를 요청하며 그것은 현재 선택된 아이템에 대한 정보들을 제공한다. 우리가 이것으로부터 필요로 하는 모든 것은 선택된 아이템에 대한 리스트 ID이다. 그러므로 노트를 편집하든 또는 그것을 삭제하던 간에, 우리는 오브젝트의 AdapterContextMenuInfo.info 필드에서 그 ID를 찾는다. 이 ID는 대응되는 액션을 수행하기 위한 editNote()와 deleteNote() 메쏘드에 전달된다.

이제 리스트뷰ListView에 있는 모든 아이템에 이 컨텍스트 메뉴를 등록하기 위해, 우리는 registerForContextMenu(View) 메쏘드에 전체 리스트뷰ListView를 전달한다.

```
registerForContextMenu(getListView());
```

컨텍스트 메뉴를 등록하기 위해, 여러분이 어떤 뷰 오브젝트든 전달할 수 있다는 것을 기억하라. 위에서 getListView()는 노트패드Notepad 애플리케이션의 ListActivity에서 사용된 ListView 오브젝트를 리턴한다. 따라서 그 리스트 안의 각 아이템들은 이 컨텍스트 메뉴에 등록된다.

서브메뉴(Submenu)

서브 메뉴는 또 다른 서브 메뉴를 제외한 어떤 메뉴에든 추가될 수 있다. 이것은 여러분의 애플리케이션이 PC 애플리케이션의 메뉴 바(파일,수정,보기 등)에 있는 아이템들처럼 토픽 별로 구성될 수 있는 많은 기능들을 가질 때 매우 유용하다.

서브 메뉴는 addSubMenu()를 사용해서 기존에 있는 메뉴에 그것을 추가함으로써 만들어진다. 이것은 SubMenu 오브젝트(Menu를 상속받는 오브젝트)를 리턴한다. 그 때 여러분은 add() 메쏘드를 사용해서 일반적인 절차대로 이 메뉴에 추가적인 아이템을 추가할 수 있다. 예를 들어 다음과 같다.

```
public boolean onCreateOptionsMenu(Menu menu) {
  boolean result = super.onCreateOptionsMenu(menu);

  SubMenu fileMenu = menu.addSubMenu("File");
  SubMenu editMenu = menu.addSubMenu("Edit");
  fileMenu.add("new");
  fileMenu.add("open");
  fileMenu.add("save");
  editMenu.add("undo");
  editMenu.add("redo");
  return result;
}
```

서브 메뉴에서 선택된 아이템에 대한 콜백은 부모parent 메뉴의 콜백 메쏘드로 만들어진다. 위의 예제에서, 서브 메뉴에서 사용자에 의한 메뉴 아이템의 선택은 onOptionsItemSelected() 콜백에 의해서 처리될 것이다.

여러분은 또한 XML에 부모parent 메뉴를 정의할 때 서브메뉴Submenus를 추가할 수 있다.

XML로 메뉴 정의하기

여러분은 마치 안드로이드 UI 레이아웃처럼, XML로 애플리케이션 메뉴를 정의할 수 있다. 그런 다음에 여러분의 메뉴 onCreate...() 콜백 메쏘드에서 그것들을 인플레이트inflate하라. 이것은 여러분의 애플리케이션 코드를 더욱 깨끗하게 만들고 더 많은 인터페이스 디자인을 XML로 분리시키며, 이는 시각화하기visualize가 더 쉽다.

이 방식을 사용하기 위해서는, 우선 여러분 프로젝트의 res/ 디렉토리에 menu 이름의 신규 폴더를 만들어라. 이곳이 애플리케이션 메뉴를 정의한 모든 XML 파일이 보관되어야 하는 곳이다.

메뉴 XML 레이아웃에는 세 개의 유효한 엘리먼트가 있다. 〈menu〉, 〈group〉, 그리고 〈item〉이 그것이다. item과 group 엘리먼트는 menu의 자식들children이 되어야 한다. 하지만 item 엘리먼트 또한 group의 자식들children이 될 수 있으며, 그리고 또 다른 menu 엘리먼트가 (서브메뉴를 만들기 위해) item의 자식child이

될 수 있다. 물론, 모든 파일의 루트root 노드는 menu 엘리먼트여야 한다.

하나의 예제로, 우리는 위의 옵션 메뉴 섹션에서 만들었던 것과 같은 메뉴를 정의할 것이다. 우리는 res/menu/ 폴더 안에 options_menu.xml 이라는 이름의 XML 파일을 가지고 시작한다.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game" android:title="New Game" />
    <item android:id="@+id/quit" android:title="Quit" />
</menu>
```

그리고 나서 onCreateOptionsMenu() 메쏘드에서, 우리는 MenuInflater.inflate()를 사용해서 이 리소스를 인플레이트inflate한다.

```
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.options_menu, menu);
    return true;
}
```

getMenuInflater() 메쏘드는 액티비티의 컨텍스트에 대한 MenuInflater를 리턴한다. 그러면 우리는 inflate()를 호출하며, 그 메쏘드에 우리의 메뉴 리소스에 대한 포인터와 콜백에 의해 파라미터로 받은 Menu 오브젝트를 전달한다.

이 작은 샘플은 onCreateOptionsMenu() 메쏘드에서 메뉴 아이템을 생성하는 것과 비교해서 더 많은 노력이 필요한 것처럼 보일 수 있지만, 이것은 보다 더 많은 아이템을 다룰 때의 많은 문제를 줄일 것이고 애플리케이션 코드를 깨끗하게 만들 것이다.

여러분은 group 엘리먼트로 item 엘리먼트를 감싸서 메뉴 그룹을 정의할 수 있으며, 그리고 item 안에 또 다른 menu를 포함시킴으로써 서브메뉴를 생성할 수 있다. 각각의 엘리먼트는 또한 숏컷shortcut 키, 체크박스checkboxes, 아이콘icons 등과 같은 특성들을 제어하기 위한 필요한 모든 애트리뷰트들을 지원한다. 이 애트리뷰트들과 XML 문법에 대해 더 많이 배우려면 4장. "리소스와 에셋"에서의 "사용가능한 리소스 타입" 중 "메뉴"를 보라.

메뉴 특성(Feature)

대부분의 메뉴 아이템에 적용될 수 있는 약간의 그 밖의 특성들이 여기 있다.

메뉴 그룹

메뉴에 신규 아이템을 추가할 때, 여러분은 선택적으로 하나의 그룹에 각각의 아이템을 포함할 수 있다. 메뉴 그룹은 그들이 보일 수 있는지visible, 활성화될 수 있는지enable, 또는 체크 가능한지checkable 같은 어떤 특징들을 공유할 수 있는 메뉴 아이템의 모음이다.

그룹은 정수integer (또는 XML 리소스 id)로 정의되어 진다. add(int,int,int,int)와 같이 아규먼트argument로 groupId를 받아들이는 add() 메쏘드들 중 하나를 사용해서 메뉴에 아이템이 추가될 때, 메뉴 아이템이 해당 그룹에 추가된다.

여러분은 setGroupVisible()을 사용해서 전체 그룹을 보이게 하거나 숨기게 할 수 있다. setGroupEnabled()를 사용해서 해당 그룹을 활성화enable 또는 비활성화disable 할 수 있다. 그리고 setGroupCheckable()를 사용해서 아이템들이 체크 될 수checkable 있는 지 설정할 수 있다.

체크가능한(Checkable) 메뉴 아이템

어떤 메뉴 아이템은 선택option을 켜고on 끄는off 인터페이스로써 사용될 수도 있다. 이것은 독립적인 선택이 가능한 체크박스, 또는 상호 배타적인 선택이 가능한 라디오 버튼을 가지고 표시될 수 있다(오른쪽에 있는 스크린샷을 보라).

> **NOTE**
> 아이콘 메뉴에 있는 메뉴 아이템은 체크박스나 라디오 버튼을 표시할 수 없다. 만약 아이콘 메뉴에 있는 아이템을 체크 가능하게 하려면, 여러분은 개인적으로 온/오프 상태가 변할 때마다 아이콘 및/또는 텍스트를 교체함으로써 그 상태를 표시해야 한다.

하나의 아이템을 체크 가능하게 만들고자 한다면, 다음과 같이 setCheckable() 메쏘드를 사용하라.

```
menu.add(0, VIBRATE_SETTING_ID, 0, "Vibrate")
    .setCheckable(true);
```

이것은 메뉴 아이템을 체크박스로 보여줄 것이다(단, 그것이 아이콘 메뉴에 있지 않는 한). 아이템이 선택될 때, onOptionsItemSelected() 콜백이 보통의 경우처럼 호출된다. 이곳이 바로 여러분이 체크박스의 상태를 설정해야 하는 곳이다. 여러분은 isChecked()를 사용해서 아이템의 현재 상태를 질의query할 수 있으며, 그리고 setChecked()를 사용해서 체크 상태를 설정할 수 있다. 아래의 내용은 onOptionsItemSelected() 콜백 내에 있을 수 있는 것이다.

```
switch (item.getItemId()) {
case VIBRATE_SETTING_ID:
  if (item.isChecked()) item.setChecked(false);
  else item.setChecked(true);
  return true;
...
}
```

상호 배타적인 라디오 버튼 아이템 그룹을 만들기 위해서는, 간단하게 동일한 그룹 ID를 각 메뉴 아이템에 할당하고 setGroupCheckable()를 호출하라. 이런 경우, 그룹 전체가 체크 가능한 것으로 설정되기 때문에 각 메뉴 아이템에서 setCheckable()를 호출할 필요는 없다. 서브메뉴 안에 있는 두 개의 상호 배타적인 선택option 예제가 여기 있다.

```
SubMenu subMenu = menu.addSubMenu("Color");
subMenu.add(COLOR_MENU_GROUP, COLOR_RED_ID, 0, "Red");
subMenu.add(COLOR_MENU_GROUP, COLOR_BLUE_ID, 0, "Blue");
subMenu.setGroupCheckable(COLOR_MENU_GROUP, true, true);
```

setGroupCheckable() 메쏘드에서, 첫 번째 아규먼트는 우리가 체크 가능한 것으로 설정하고자 하는 그룹에 대한 ID이다. 두 번째 아규먼트는 그룹 아이템들이

체크 가능한 지에 대한 것이다. 마지막은 각 아이템이 상호배타적으로 체크되어야 하는 지에 대한 것이다(만약 우리가 이것을 거짓false으로 설정한다면, 모든 아이템은 라디오 버튼 대신에 체크박스가 될 것이다). 그룹이 상호배타적인 것으로 설정될 때(라디오 버튼) 새로운 아이템이 선택될 때마다 나머지 모든 것들은 자동으로 선택이 해제된다.

> NOTE
> 체크 가능한Checkable 메뉴 아이템은 오직 세션 별 기반에서만 사용되도록 의도되었다. 그래서 그것은 디바이스에 저장되지 않는다(예를 들어 Maps 애플리케이션 안에서 Map 모드 설정은 저장되지 않는다. - 위에 있는 스크린샷). 만약 여러분이 사용자를 위해 저장하고자 하는 애플리케이션 설정이 있다면, 여러분은 프레퍼런스Preferences를 사용해서 데이터를 저장해야 하며, 그리고 PreferenceActivity를 사용해서 그것들을 관리해야 한다.

숏컷(Shortcut, 단축) 키

문자 및(또는) 숫자를 사용하는 빠른 접근 숏컷 키shortcut key들은 setAlphabeticShortcut(char)(문자 숏컷shortcut을 설정하기 위해), setNumericShortcut(int)(숫자 숏컷을 설정하기 위해), 또는 setShortcut(char,int) (두 가지 모두를 설정하기 위해)을 사용해서 메뉴 아이템들에 추가될 수 있다. 예를 들어 다음과 같다.

```
menu.add(0, MENU_QUIT, 0, "Quit")
    .setAlphabeticShortcut('q');
```

이제, 메뉴가 오픈될 때(또는 MENU 키가 눌려져 있는 동안), "q" 키를 누르면 이 아이템을 선택할 것이다

이 숏컷 키shortcut key는 메뉴 아이템에 하나의 팁처럼 메뉴 아이템 이름 아래에 표시될 것이다(단, 아이콘 메뉴에 있는 아이템에서는 표시되지 않는다).

> NOTE 숏컷은 컨텍스트 메뉴 안의 아이템에 추가될 수 없다.

메뉴 아이템 인텐트

만약 여러분이 2장. "애플리케이션 기초"를 읽었다면, 안드로이드 인텐트에 적어도 조금은 익숙해 졌을 것이다. 이것은 애플리케이션들이 서로 서로 연결되고, 정보를 공유하고, 그리고 사용자 태스크를 협력해서 수행할 수 있게 한다. 마치 여러분의 애플리케이션이 웹 브라우저, 이메일 클라이언트, 또는 여러분의 애플리케이션 내의 또 다른 액티비티를 시작시키기 위해 인텐트를 발생시키는 것과 같이, 여러분은 메뉴에서도 그런 액션을 수행할 수 있다. 이것에는 두 가지 방법이 있다. 하나는 인텐트를 정의하고 단일 메뉴 아이템에 그것을 할당하는 것이다. 다른 하나는 인텐트를 정의하고 안드로이드에게 액티비티를 디바이스에서 검색하게 하고, 그 인텐트 규격에 맞는 각각의 것을 메뉴 아이템에 동적으로 추가하는 것이다.

인텐트 생성과 다른 애플리케이션에 애플리케이션 서비스를 제공하는 것에 대한 더 많은 정보에 대해서는, 5장. "인텐트와 인텐트 필터" 문서를 읽어라.

단일 메뉴 아이템에 인텐트 설정하기

만약 여러분이 새로운 액티비티를 런치하는 특정 메뉴 아이템을 제공하자고 한다면, 여러분은 setIntent() 메쏘드를 사용해서 해당 메뉴 아이템에 대한 인텐트를 명확하게 정의할 수 있다.

예를 들어 onCreateOptionsMenu() 메쏘드 내에서, 아래와 같은 인텐트를 가지는 신규 메뉴 아이템을 정의할 수 있다.

```
MenuItem menuItem = menu.add(0, PHOTO_PICKER_ID, 0, "Select Photo");
menuItem.setIntent(new Intent(this, PhotoPicker.class));
```

아이템이 선택될 때, 안드로이드는 자동으로 액티비티를 런치할 것이다.

> **NOTE**
> 이것은 여러분의 액티비티에 결과를 리턴하지 않을 것이다. 만약 여러분이 결과를 되돌려 받기를 바란다면, setIntent()를 사용하지 마라. 대신 onOptionsMenuItemSelected() 또는 onContextMenuItemSelected() 콜백에서 통상 하듯이 선택된 아이템을 처리하고, 그런 다음에 startActivityForResult()를 호출하라.

동적으로 인텐트 추가하기

만약 여러분의 현재 액티비티 또는 선택된 아이템과 관련된 다수의 잠재적 액티비티가 있다면, 애플리케이션은 동적으로 다른 서비스들을 실행하는 메뉴 아이템을 추가할 수 있다.

메뉴를 생성하는 동안, Intent.ALTERNATIVE_CATEGORY 카테고리 및/또는 Intent.SELECTED_ALTERNATIVE 카테고리, (만약 있다면)현재 선택된 MIME 타입, 그리고 그 밖의 다른 요건들을 정의하라. 즉 새로운 액티비티를 오픈하기 위해 인텐트 필터를 충족시켜야 하는 것처럼 그렇게 하라. 그런 다음에 안드로이드가 그 요구조건에 일치하는 임의의 서비스를 검색하도록 하기 위해 addIntentOption()을 호출하고, 그것들을 여러분의 메뉴에 추가하라. 만약 해당 인텐트를 만족시키는 설치된 애플리케이션이 없다면, 어떤 추가적인 메뉴 아이템도 추가되지 않는다.

> **NOTE**
>
> SELECTED_ALTERNATIVE는 스크린에서 현재 선택된 엘리먼트를 처리하기 위해 사용된다. 그러므로 그것은 옵션 메뉴가 오픈될 때 항상 호출되는 onCreateContextMenu() 또는 onPrepareOptionsMenu()에서 메뉴를 만들 때만 사용되어야 한다.

여기에 애플리케이션이 자신의 메뉴에 보여줄 추가적인 서비스들을 검색하는 방법을 설명하는 예제가 있다.

```
public boolean onCreateOptionsMenu(Menu menu){
    super.onCreateOptionsMenu(menu);
    // Create an Intent that describes the requirements to fulfill,
    // to be included
    // in our menu.
    // The offering app must include a category value
    // of Intent.CATEGORY_ALTERNATIVE.
    Intent intent = new Intent(null, getIntent().getData());
    intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
    // Search for, and populate the menu with,
    // acceptable offering applications.
    menu.addIntentOptions(
       thisClass.INTENT_OPTIONS,  // Menu group
```

```
        0,      // Unique item ID (none)
        0,      // Order for the items (none)
        this.getComponentName(),   // The current Activity name
        null,   // Specific items to place first (none)
        intent, // Intent created above that describes our requirements
        0,      // Additional flags to control items (none)
        null);  // Array of MenuItems that corrolate
                // to specific items (none)
    return true;
}
```

정의된 인텐트와 일치하는 인텐트 필터를 제공하는 검색되어진 각각의 액티비티가 메뉴 아이템이 추가될 것이다. 그 메뉴 아이템에는 그 액티비티의 인텐트 필터의 android:label 값이 메뉴 아이템 텍스트로 보이게 된다. addIntentOptions() 메쏘드는 또한 추가된 메뉴 아이템들의 개수를 리턴할 것이다.

addIntentOptions()가 호출될 때, 그것은 첫 번째 아규먼트에 명시된 메뉴 그룹 안에 있는 모든 메뉴 아이템들을 오버라이드 할 것이라는 것을 또한 주목하라.

만약 여러분이 다른 애플리케이션의 메뉴에 여러분의 액티비티의 서비스를 제공하고자 한다면, 여러분은 보통의 경우처럼 인텐트 필터만 정의하면 된다. 인텐트 필터에 있는 〈category〉 엘리먼트의 name 애트리뷰트에 ALTERNATIVE 및/또는 SELECTED_ALTERNATIVE 값들을 포함시키도록 하라. 예를 들어 다음과 같다.

```
<intent-filter label="Resize Image">
    ...
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category
       android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    ...
</intent-filter>
```

5장. "인텐트와 인텐트 필터"에서 인텐트 필터를 작성하는 것에 대해 더 자세히 보라. 이 기술을 사용하는 샘플 애플리케이션에 대해서는 (역주 : sdk에 포함되어 있는) 노트패드NotePad 샘플 코드를 보라.

다이얼로그 생성

다이얼로그는 통상 현재 액티비티 위에 나타나는 작은 윈도우이다. 그 아래에 놓인 액티비티는 포커스를 잃으며, 해당 다이얼로그는 사용자와의 상호작용을 수용한다. 다이얼로그는 보통 노티피케이션notification과 진행중인 애플리케이션과 직접적으로 연관된 짧은 행위activity들을 위해 사용된다.

안드로이드 API는 다음과 같은 유형의 다이얼로그 오브젝트를 지원한다.

AlertDialog

0 개, 한 개, 두 개, 또는 세 개의 버튼 및(또는) 체크 박스나 라디오 버튼을 포함하는 선택가능한 아이템들을 관리할 수 있는 다이얼로그. AlertDialog는 대부분의 다이얼로그 사용자 인터페이스를 만들 수 있는 능력을 가지고 있으며, 권장되는 다이얼로그 타입이다. 다음에 나오는 "다이얼로그 생성하기"를 보라.

ProgressDialog

진행상태바퀴progress wheel 또는 진행상태바progress bar를 보여주는 다이얼로그. 이것은 AlertDialog의 확장extension이기 때문에 이것 또한 버튼을 지원한다. 다음에 나오는 "ProgressDialog 생성하기"를 보라.

DatePickerDialog

사용자가 날짜를 선택하는 것을 가능하게 하는 다이얼로그. Hello Date Picker 튜토리얼을 보라.

TimePickerDialog

사용자가 시간을 선택할 수 있게 하는 다이얼로그. Hello TimePicker 튜토리얼을 보라.

만약 여러분이 자신의 다이얼로그를 커스터마이즈하고자 한다면, 여러분은 기본 다이얼로그 오브젝트 또는 위에서 나열된 임의의 서브클래스를 확장extension할 수 있으며 새로운 레이아웃을 정의할 수 있다. 그리고 다음에 나오는 "커스텀 다이얼로그 생성하기" 섹션을 보아라.

다이얼로그 표시하기

다이얼로그는 항상 액티비티의 일부로써 생성되고 보여진다. 여러분은 보통 액티비티의 onCreateDialog(int) 콜백 메쏘드 내에서 다이얼로그를 생성해야 한다. 여러분이 이 콜백을 사용할 때, 안드로이드 시스템은 자동으로 각각의 다이얼로그의 상태를 관리하며, 액티비티가 효과적으로 각각의 다이얼로그의 "소유자owner"가 되도록 액티비티에 다이얼로그를 연결hook한다. 그렇게 함으로써 각각의 다이얼로그는 액티비티의 특정 속성들을 상속한다. 예를 들어 다이얼로그가 오픈될 때, 메뉴 키는 액티비티를 위해 정의된 메뉴들을 보여주게 되며, 볼륨 키는 액티비티의 의해 사용되는 오디오 스트림을 수정하게 된다.

만약 여러분이 onCreateDialog() 메쏘드의 외부에서 다이얼로그를 생성하기로 결정한다면, 그것은 액티비티에 첨부(attach)되지 않을 것이다. 하지만 여러분은 setOwnerActivity(Activity)를 사용해서 액티비티에 그것을 첨부할 수 있다.

여러분이 다이얼로그를 표시하고자 할 때, showDialog(int)를 호출하라. 그리고 그것에 여러분이 표시하고자 하는 다이얼로그의 고유한 식별자인 정수integer를 전달하라.

다이얼로그가 최초로 요청될 때, 안드로이드는 여러분 액티비티로부터 onCreateDialog(int)를 호출하며, 그 메쏘드에서 여러분은 다이얼로그를 인스턴스화해야 한다. 이 콜백 메쏘드는 여러분이 showDialog(int)에 파라미터로 전달했던 것과 동일한 ID를 전달받는다. 그곳에서 다이얼로그를 생성한 후, 그 메쏘드가 끝나는 곳에서 오브젝트를 리턴하라.

다이얼로그가 보여지기 전에, 안드로이드는 또한 선택사항으로 onPrepareDialog(int,Dialog) 콜백 메쏘드를 호출한다. 만약 여러분이 다이얼로그가 오픈될 때마다 다이얼로그의 어떤 속성들properties을 바꾸고자 한다면 이 메쏘드를 정의하라. 이 메쏘드는 다이얼로그가 오픈될 때마다 호출된다. 반면에, onCreateDialog(int)는 오직 다이얼로그가 최초로 오픈될 때만 호출된다. 만약 여러분이 onPrepareDialog()를 정의하지 않는다면, 다이얼로그는 이전에 오픈되었을 때와 똑같은 모습으로 남아 있을 것이다. 이 메쏘드 또한

onCreateDialog()에서 여러분이 생성한 다이얼로그 오브젝트와 함께 다이얼로그의 ID를 전달받는다.

onCreateDialog(int)와 onPrepareDialog(int, Dialog) 콜백 메쏘드를 정의하기 위한 최선의 방법은 그 메쏘드에 전달되는 id 파라미터를 체크하는 switch 구문을 만드는 것이다. 각각의 case는 고유한 다이얼로그 ID를 체크해야 하며, 그런 다음 각각에 대응되는 다이얼로그를 생성하고 정의하라. 예를 들어 하나는 게임이 멈추었다는 것을 가리키고, 다른 하나는 게임이 종료되었다는 것을 가리키는 두 개의 다른 다이얼로그를 사용하는 게임을 상상해 보라. 우선 각각의 다이얼로그에 대한 정수integer ID를 정의하라.

```
static final int DIALOG_PAUSED_ID = 0;
static final int DIALOG_GAMEOVER_ID = 1;
```

그런 다음, 각각의 ID에 대한 switch case를 가지고 onCreateDialog(int) 콜백을 정의하라.

```
protected Dialog onCreateDialog(int id) {
    Dialog dialog;
    switch(id) {
    case DIALOG_PAUSED_ID:
        // do the work to define the pause Dialog
        break;
    case DIALOG_GAMEOVER_ID:
        // do the work to define the game over Dialog
        break;
    default:
        dialog = null;
    }
    return dialog;
}
```

이 예제에서, 각각의 case 안에는 어떤 코드도 없다. 왜냐하면 여러분의 다이얼로그를 정의하는 프로시저는 이 섹션의 범위 밖이기 때문이다. 다음에 나오는 "AlertDialog 생성하기" 섹션을 보라. 이 예제에 대한 적합한 코드를 제공한다.

다이얼로그 중 하나를 보여줄 때가 되면, 다이얼로그의 ID를 사용해서 showDialog(int)를 호출하라.

```
showDialog(DIALOG_PAUSED_ID);
```

다이얼로그 닫기(Dismissing)

여러분이 다이얼로그를 닫을 준비가 되었을 때, 여러분은 다이얼로그에서 dismiss()를 호출함으로써 그것을 닫을 수 있다. 만약 필요하다면 여러분은 또한 액티비티로부터 dismissDialog(int)를 호출할 수 있으며, 이것은 여러분을 위해 효과적으로 다이얼로그의 dismiss()를 호출한다.

만약 여러분이 (앞의 섹션에서 논의했던 대로) 여러분의 다이얼로그의 상태를 관리하기 위해 onCreateDialog(int)를 사용한다면, 다이얼로그 오브젝트의 상태는 다이얼로그가 닫혀질 때마다 액티비티에 의해 보관된다. 만약 여러분이 더 이상 이 오브젝트를 필요로 하지 않는다고 결정한다거나 또는 상태가 제거되는 것이 중요하다면, 그러면 여러분은 removeDialog(int)를 호출해야 한다. 이것은 오브젝트에 대한 모든 내부적인 레퍼런스들을 제거할 것이며, 만약 다이얼로그가 표시되고 있다면 그것을 닫을 것이다.

닫힘 리스너(dismiss listeners) 사용하기

만약 여러분의 애플리케이션이 다이얼로그가 닫히는 시점에 어떤 절차를 수행하길 원한다면, 그러면 여러분의 다이얼로그에 on-dismiss 리스너listener 첨부해야 한다.

우선 DialogInterface.OnDismissListener 인터페이스를 정의하라. 이 인터페이스는 단지 하나의 메쏘드인 onDismiss(DialogInterface)를 가지는데, 그것은 다이얼로그가 사라질 때 호출될 것이다. 그런 다음 간단하게 여러분이 구현한 OnDismissListener를 setOnDismissListener()에 파라미터로 전달하라.

하지만 다이얼로그 또한 "취소cancel" 될 수 있다는 것에 주의하라. 이것은 특별한 경우인데, 그것은 다이얼로그가 명시적으로 사용자에 의해 취소되었다는 것

을 가리킨다. 이것은 만약 사용자가 다이얼로그Dialog를 닫기 위해 "back" 버튼을 누르거나, 또는 다이얼로그가 명시적으로 cancel()을 호출(아마도 다이얼로그의 "취소cancel"로 부터)한다면 일어난다. 다이얼로그가 취소될 때, onDismisssListener는 여전히 통보를 받는다. 하지만 여러분의 다이얼로그가 명시적으로 취소된 것임(보통의 닫힘이 아님, 역주 : 오직 다이얼로그의 취소 버튼에 의해서만 닫힌 경우)을 알고자 한다면, 여러분은 setOnCancelListener()를 사용해서 DialogInterface.OnCancelListener를 등록해야 한다.

AlertDialog 생성

AlertDialog는 다이얼로그Dialog 클래스의 확장extension이다. 그것은 대부분의 다이얼로그 사용자 인터페이스를 생성할 수 있고 권장되는 다이얼로그 타입이다. 여러분은 다음과 같은 어떤 형태를 사용하는 다이얼로그를 위해서는 AlertDialog를 사용해야 한다.

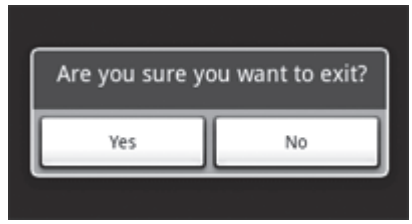
- 타이틀
- 텍스트 메시지
- 한 개, 두 개, 또는 세 개 버튼
- 선택 가능한 아이템의 목록(선택사항으로 체크박스 또는 라디오 버튼을 가짐)

AlertDialog를 생성하기 위해서는, AlertDialog.Builder 서브클래스를 사용하라. AlertDialog.Builder(Context)를 사용해서 Builder를 얻어라. 그런 다음에 AlertDialog 속성 전부를 정의하는 그 클래스의 퍼블릭public 메쏘드들을 사용하라. Builder를 사용해서 그것을 한 후, create()를 사용해서 AlertDialog 오브젝트를 얻어라.

다음의 주제들은 AlertDialog.Builder 클래스를 사용해서 다양한 속성의 AlertDialog를 정의하는 방법을 보여준다. 만약 여러분이 onCreateDialog() 콜백 메쏘드 안에서 다음의 샘플 코드 중 어떤 것을 사용한다면, 여러분은 다이얼로그를 표시하는 결과를 초래하는 Dialog 오브젝트를 리턴할 수 있다.

버튼 추가하기(Adding buttons)

우측의 스크린샷에서 보여지는 것과 같은 나란히 놓인 버튼을 가진 AlertDialog를 생성하기 위해서는, set...Button() 메쏘드들을 사용하라.




```
AlertDialog.Builder builder
                = new AlertDialog.Builder(this);

builder.setMessage("Are you sure you want to exit?")
   .setCancelable(false)
   .setPositiveButton("Yes",
       new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                MyActivity.this.finish();
           }})
   .setNegativeButton("No", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
           }});
AlertDialog alert = builder.create();
```

먼저, setMessage(CharSequence)를 사용해서 다이얼로그를 위한 메시지를 추가하라. 그런 다음 메쏘드-체이닝method-chaining을 시작하고, setCancelable(boolean)을 사용해서 다이얼로그가 취소될 수 없도록(사용자가 back 버튼을 사용해서 다이얼로그를 닫을 수 없도록) 설정하라. 각각의 버튼에 대한 이름과 사용자가 그 버튼을 선택할 때 수행할 액션을 정의하는 DialogInterface.OnClickListener를 파라미터로 받는 setPositiveButton()과 같은 set...Button() 메쏘드들 중 하나를 사용하라.

주의 여러분은 AlertDialog에 각각의 버튼 타입 중 하나만을 추가할 수 있다. 즉, 여러분은 "긍정positive" 버튼을 한 개 이상 가질 수 없다. 이것은 사용 가능한 버튼의 수를 세 개로 제한한다: 긍정positive, 중립neutral, 그리고 부정negative이 그것이다. 이런 이름들은 버튼의 실제 기능과 기술적인 연관성을 가지지는 않는다. 하지만 그것은 여러분으로 하여금 어느 것이 무엇을 하는지 알아내는 것을 돕는다.

리스트 추가하기(Adding a list)

위에서 보여지는 것과 같이 선택가능한 아이템들의 리스트를 가지는 AlertDialog를 생성하기 위해서는 setItems() 메쏘드를 사용하라.

```
final CharSequence[] items
       = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new Alert-
Dialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item],
               Toast.LENGTH_SHORT).show();
    }});
AlertDialog alert = builder.create();
```

먼저, setTitle(CharSequence)을 사용해서 다이얼로그에 타이틀을 추가하라. 그런 다음 표시할 아이템의 배열과 사용자가 하나의 아이템을 선택할 때 수행할 액션을 정의하는 DialogInterface.OnclickListener를 파라미터로 받는 setItems()를 사용해서 선택가능한 아이템 리스트를 추가하라.

체크 박스와 라디오 버튼 추가하기

다이얼로그 안에 복수 선택 아이템(체크박스) 또는 단일 선택 아이템(라디오 버튼) 리스트를 생성하기 위해서는, 각각에 해당되는 setMultiChoiceItems()와 setSingleChoiceItems() 메쏘드를 사용하라. 만약 onCreateDialog() 콜백 메쏘드에 이러한 선택가능한 리스트들 중 하나를 생성한다면, 안드로이드는 여러분을 위해 리스트의 상태를 관리한다. 액티비티가 활성화되어 있는 한, 다이얼로그는 이전에 선택된 아이템을 기억한다. 하지만 사용자가 액티비티를 종료할 때 그 선택 정보는 잃게 된다.

주의 사용자가 액티비티를 떠나거나 멈출 때 그 선택 정보를 저장하기 위해서, 여러분은 액티비티 생명주기의 곳곳에서 적절하게 설정을 저장하고 복구해야 한다. 그 선택 정보를 영구적으로, 심지어는 액티비티 프로세스가 완전하게 종료될 때 조차도 저장하기 위해서, 여러분은 데이터 저장공간 기술 중 하나를 사용해서 그 설정들을 저장하는 것이 필요하다.

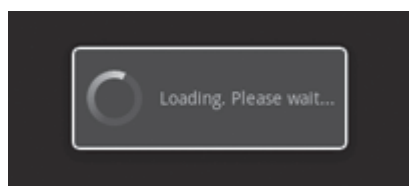
위에서 보여진 것과 같은 단일 선택 아이템들의 목록을 사용하는 AlertDialog를 생성하기 위해서는, 이전의 예제와 같은 코드를 사용하라. 하지만 setSingleChoiceItems()로 setItems() 메쏘드를 대체하라.

```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setSingleChoiceItems(items, -1, new DialogInterface.OnClick-
Listener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item],
               Toast.LENGTH_SHORT).show();
    }
});
AlertDialog alert = builder.create();
```

setSingleChoiceItems()의 두 번째 파라미터는 체크된 아이템에 대한 정수integer 값이다. 그것은 0zero을 베이스로 디폴트로 선택되어야 하는 아이템의 위치를 가리킨다. 디폴트로 어떤 아이템도 선택되지 않게 하기 위해서는 "-1"을 사용하라.

ProgressDialog 생성



ProgressDialog는 AlertDialog 클래스의 확장extension이다. 그것은 명확하지 않은 진행상태를 가진 태스크에 대해 회전하는 바퀴wheel로 진행상태 애니메이션을 보여줄 수 있으며, 또한 명확한 진행상태를 가진 태스크에 대해 진행상태바

progress bar를 보여줄 수 있다. 이 다이얼로그는 또한 다운로드를 취소하기 위한 것과 같은 버튼들을 제공할 수 있다.

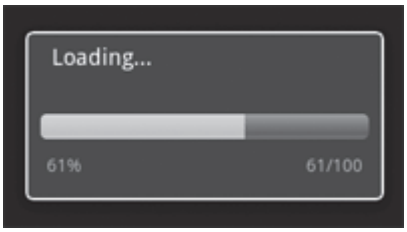
Progress 다이얼로그를 오픈하는 것은 ProgressDialog.show()를 호출하는 것 만큼 간단하게 될 수 있다. 예를 들어 위에 보이는 진행상태 다이얼로그는 아래에서 보여지듯이 onCreateDialog(int) 콜백을 통해 다이얼로그를 관리하지 않고 쉽게 만들어질 수 있다.

```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "",
                        "Loading. Please wait...", true);
```

첫 번째 파라미터는 애플리케이션 컨텍스트Context이고, 두 번째는 다이얼로그에 대한 타이틀(빈 상태로 남음), 세 번째는 메시지, 그리고 마지막 파라미터는 진행상태의 무한 진행여부(이것은 오직 진행상태 바(progress bar)를 생성할 때와 관련되며, 그것은 다음 섹션에서 논의된다)이다.

Progress 다이얼로그의 디폴트 스타일은 회전하는 바퀴이다. 만약 여러분이 낟알모양으로 진행상태를 로드하는 것을 표시하는 상태바progress bar를 생성하고자 한다면, 다음 섹션에서 논의되듯이 약간의 코드가 더 요구된다.

진행상태바 표시하기 (Showing a progress bar)



애니메이션되는 진행상태바를 사용해서 진행율progression을 표시하기 위해서는 다음과 같이 하라.

1. 클래스 생성자, ProgressDialog(Context)를 사용해서 ProgressDialog를 초기화하라.
2. setProgressStyle(int)을 사용해서 진행상태 스타일에 "STYLE_HORIZONTAL"을 설정하고, 메시지 같은 그 밖의 속성들을 설정하라.
3. 여러분이 다이얼로그를 보여줄 준비가 되었을 때, show()를 호출하거나 또는

onCreateDialog(int) 콜백에서 ProgressDialog를 리턴하라.

4. 여러분은 현재까지 완료된 전체 백분율 값을 가지고 setProgress(int)을 호출하거나, 또는 현재까지 완료된 전체 백분율에 증가분 값을 더하는 incrementProgressBy(int)를 호출함으로써 바에 표시되는 진행양amount of progress을 증가시킬 수 있다.

예를 들어 여러분의 설정은 다음과 같을 수 있다.

```
ProgressDialog progressDialog;
progressDialog = new ProgressDialog(mContext);
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
```

그 설정은 간단하다. Progress 다이얼로그를 만들기 위해 필요한 대부분의 코드는 실제로 그것을 업데이트하는 과정에 포함된다. 여러분은 이 작업을 위해 애플리케이션 내에 두 번째 쓰레드를 생성하는 것이 필요하다는 것을 발견할 수 있다. 그런 다음에 핸들러 오브젝트를 사용해서 액티비티의 사용자 인터페이스 쓰레드에게 진행상태를 역으로 레포트하라. 만약 여러분이 핸들러를 가지는 추가적인 쓰레드를 사용하는 것에 익숙하지 않다면 아래의 예제 액티비티를 보라. 그것은 액티비티에 의해 관리되는 Progress 다이얼로그를 증가시키기 위해 두 번째 쓰레드를 사용한다.

두 번째 쓰레드를 사용하는 ProgressDialog 예제

이 예제는 어떤 과정(이것은 실제로 단지 100까지 셀 뿐이다)의 진행상태를 추적하기 위해 두 번째 쓰레드를 사용한다. 그 쓰레드는 진행이 이루어질 때마다 핸들러를 통해서 main 액티비티에 역으로 메시지를 보낸다. 그러면 그 main 액티비티가 Progress 다이얼로그를 업데이트한다.

```
package com.example.progressdialog;
import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
```

```
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class NotificationTest extends Activity {
    static final int PROGRESS_DIALOG = 0;
    Button button;
    ProgressThread progressThread;
    ProgressDialog progressDialog;
    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // Setup the button that starts the progress dialog
        button = (Button) findViewById(R.id.progressDialog);
        button.setOnClickListener(new OnClickListener(){
            public void onClick(View v) {
                showDialog(PROGRESS_DIALOG);
            }
        });
    }
       protected Dialog onCreateDialog(int id) {
        switch(id) {
        case PROGRESS_DIALOG:
            progressDialog = new ProgressDialog(NotificationTest.this);
            progressDialog
               .setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            progressDialog.setMessage("Loading...");
            progressThread = new ProgressThread(handler);
            progressThread.start();
            return progressDialog;
        default:
            return null;
        }
    }

    // Define the Handler that receives messages
    //  from the thread and update the progress
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            int total = msg.getData().getInt("total");
```

```
            progressDialog.setProgress(total);
            if (total >= 100){
                dismissDialog(PROGRESS_DIALOG);
                progressThread.setState(ProgressThread.STATE_DONE);
            }
        }
    };
    /** Nested class that performs progress calculations (counting) */
    private class ProgressThread extends Thread {
        Handler mHandler;
        final static int STATE_DONE = 0;
        final static int STATE_RUNNING = 1;
        int mState;
        int total;

        ProgressThread(Handler h) {
            mHandler = h;
        }

        public void run() {
            mState = STATE_RUNNING;
            total = 0;
            while (mState == STATE_RUNNING) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    Log.e("ERROR", "Thread Interrupted");
                }
                Message msg = mHandler.obtainMessage();
                Bundle b = new Bundle();
                b.putInt("total", total);
                msg.setData(b);
                mHandler.sendMessage(msg);
                total++;
            }
        }

    /* sets the current state for the thread, used to stop the thread */
        public void setState(int state) {
            mState = state;
        }
    }
}
```

커스텀(Custom) 다이얼로그 생성

만약 여러분이 다이얼로그에 대한 커스터마이즈된 디자인을 원한다면, 여러분은 레이아웃과 위젯 엘리먼트를 사용해서 다이얼로그 윈도우에 대한 여러분 자신의 레이아웃을 생성할 수 있다. 여러분의 레이아웃을 정의한 후에 루트root 뷰 오브젝트나 레이아웃 리소스 ID를 setContentView(View)에 전달하라.

예를 들어 위에 보이는 다이얼로그를 생성하기 위해서는 다음과 같이 하라.

1. Custom_dialog.xml 로 저장된 XML 레이아웃을 생성하라.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/layout_root"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="10dp"
              />
     <ImageView android:id="@+id/image"
               android:layout_width="wrap_content"
               android:layout_height="fill_parent"
               android:layout_marginRight="10dp"
               />
     <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="fill_parent"
              android:textColor="#FFF"
              />
</LinearLayout>
```

이 XML은 LinearLayout안에 ImageView와 TextView를 정의한다.

2. 다이얼로그의 컨텐트 뷰에 위의 레이아웃을 지정하라. 그리고 ImageView와 TextView 엘리먼트에 대한 내용을 정의하라.

```
Context mContext = getApplicationContext();
Dialog dialog = new Dialog(mContext);

dialog.setContentView(R.layout.custom_dialog);
dialog.setTitle("Custom Dialog");

TextView text = (TextView) dialog.findViewById(R.id.text);
text.setText("Hello, this is a custom dialog!");
ImageView image = (ImageView) dialog.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
```

여러분이 다이얼로그를 인스턴스화한 후, setContentView(int)에 레이아웃 리소스 ID를 파라미터로 전달해서 여러분의 커스텀 레이아웃을 다이얼로그의 컨텐트 뷰로 설정하라. 이제 그 다이얼로그는 정의된 레이아웃을 가지기 때문에 여러분은 findViewById(int)를 사용해서 레이아웃으로부터 뷰 오브젝트를 가져올 수 있으며 그 안의 내용을 수정하라.

3. 이것으로 끝이다. 여러분은 이제 앞에서 보여졌던 다이얼로그에서 설명되듯이 그 다이얼로그를 보일 수 있다.

베이스 다이얼로그Dialog 클래스를 사용해서 만들어진 다이얼로그는 타이틀을 가져야 한다. 만약 여러분이 setTitle()을 호출하지 않는다면, 그 공간은 타이틀이 비어 있기는 하지만 여전히 빈 상태로 보여지게 된다. 만약 여러분이 타이틀을 전혀 원하지 않는다면, 여러분은 AlertDialog 클래스를 사용해서 여러분의 커스텀 다이얼로그를 생성해야 한다. 하지만 AlertDialog는 AlertDialog.Bulder 클래스를 가지고 매우 쉽게 생성되기 때문에, 여러분은 위에서 사용된 setContentView(int) 메쏘드에 접근할 필요는 없다. 대신 여러분은 setView(View)를 사용해야 한다. 이 메쏘드는 뷰 오브젝트를 받아들인다. 그러므로 여러분은 XML 레이아웃의 루트root 뷰 오브젝트를 인플레이트inflate하는 것이 필요하다.

XML 레이아웃을 인플레이트 하기 위해서는, getLayoutInfalter()(또는 getSystemService())를 사용해서 LayoutInflater를 얻어라. 그리고 그런 다음에 inflate(int, ViewGroup)을 호출하라. 이것의 첫 번째 파라미터는 레이아웃 리소스 ID이며 두 번째는 루트root 뷰의 ID이다. 이 시점에서, 여러분은 그 레

이아웃에서 뷰 오브젝트를 찾기 위해 인플레이트된 레이아웃을 사용할 수 있다. 그리고 ImageView와 TextView에 대한 내용을 정의하라. 그런 다음에 AlertDialog.Builder를 인스턴스화하고 setView(View)를 사용해서 다이얼로그에 대한 인플레이트inflate된 레이아웃을 지정하라.

여기에 AlertDialog에 커스텀 레이아웃을 생성하는 예제가 있다.

```
AlertDialog.Builder builder;
AlertDialog alertDialog;

Context mContext = getApplicationContext();
LayoutInflater inflater
       = (LayoutInflater) mContext.getSystemService(LAYOUT_INFLATER);
View layout = inflater.inflate(R.layout.custom_dialog,
                 (ViewGroup) findViewById(R.id.layout_root));
TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Hello, this is a custom dialog!");
ImageView image = (ImageView) layout.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
builder = new AlertDialog.Builder(mContext);
builder.setView(layout);
alertDialog = builder.create();
```

여러분의 커스텀 레이아웃을 위해 AlertDialog를 사용하는 것은, 여러분에게 관리되는managed 버튼, 선택가능한selectable 리스트, 타이틀, 아이콘 등과 같은 내장된built-in AlertDialog 특징들의 장점을 이용 가능하게 한다.

더 많은 정보에 대해서는, Dialog와 AlertDialog.Builder 클래스들에 대한 레퍼런스 문서를 참고하라.

UI 이벤트 처리

안드로이드에는, 여러분의 애플리케이션과 사용자 간의 상호작용으로부터 이벤트를 가로채기 위한 한 가지 이상의 방법이 있다. 여러분의 사용자 인터페이스 내의 이벤트를 고려할 때, 그 접근법은 사용자가 상호작용하는 특정 뷰 오브젝트의 이벤트를 획득하는 것이다. 뷰 클래스는 그렇게 하기 위한 수단을 제공한다.

레이아웃을 구성하기 위해 사용하는 다양한 뷰 클래스 안에서, 여러분은 아마도 UI 이벤트를 처리하는 데 도움이 되는 몇 가지 퍼블릭public 콜백 메쏘드를 주목할 것이다. 이러한 메쏘드들은 그 오브젝트에 각각의 액션이 발생할 때, 안드로이드 프레임워크에 의해 호출된다. 예를 들어 버튼과 같은 뷰가 터치되게 될 때, onTouchEvent() 메쏘드가 그 오브젝트에서 호출된다. 하지만 이것을 가로채기 위해서는 해당 클래스를 확장extend해서 메쏘드를 오버라이드해야 한다. 명백하게(단지 이벤트만을 제어하기 위해) 여러분이 사용하고자 하는 모든 뷰 오브젝트를 확장하는 것은 불합리할 수 있다. 이것이 뷰View 클래스가 여러분이 훨씬 더 쉽게 정의할 수 있는 콜백을 가진 네스티드nested 인터페이스 모음을 또한 포함하는 이유이다. 이벤트 리스너event listener라고 불리는 이러한 인터페이스들은 여러분의 UI와의 사용자 간의 상호작용을 캡쳐하기 위한 여러분의 티켓이다.

여러분은 사용자와의 상호작용을 청취하기 위해 보다 일반적으로는 이벤트 리스너를 사용하려 하겠지만, 커스텀 컴포넌트를 만들기 위해 뷰 클래스를 확장하고자 할 때가 있을 수 있다. 아마 여러분은 버튼을 더 보기 좋게 하기 위해 버튼Button 클래스를 확장extend하고자 할 수 있다. 이런 경우에, 여러분은 이벤트 핸들러event handler 클래스를 사용하여 여러분의 클래스에 대한 디폴트 이벤트 동작 구조를 정의할 수 있을 것이다.

이벤트 리스너(Listener)

이벤트 리스너listener는 하나의 콜백 메쏘드를 포함하는 뷰 클래스 내의 인터페이스이다. 이 메쏘드는 리스너가 등록된 뷰의 UI 내의 아이템과 사용자 간의 상호작용이 발생할 때, 안드로이드 프레임워크에 의해 호출된다.

아래의 콜백 메쏘드들이 이벤트 리스너 인터페이스에 포함된다.

onClick()

View.OnClickListener로부터 콜백됨. 이것은 터치모드일 때 사용자가 아이템을 터치하거나 또는 네비게이션 키나 트랙볼trackball로 그 아이템을 포커스하여 "enter" 키를 누르거나 트랙볼을 눌렀을 때 호출된다.

onLongClick()

View.OnFocusChangeListener로부터 콜백됨. 이것은 터치모드일 때 사용자가 아이템을 터치해서 잡거나 또는 네비게이션-키나 트랙볼trackball로 해당 아이템을 포커스하여 적당한 "enter" 키를 누르고 있거나 또는 1초동안 트랙볼을 누르고 있을 때 호출된다.

onFocusChange()

View.OnFocusChangeListener로부터 콜백됨. 이것은 사용자가 네비게이션-키 또는 트랙볼trackball을 사용하여 아이템 위로 움직이거나, 또는 그것으로부터 벗어날 때 호출된다.

onKey()

View.OnKeyListener로부터 콜백됨. 이것은 사용자가 아이템을 포커스를 하여, 디바이스에 있는 키를 누르거나 놓았을 때 호출된다.

onTouch()

View.OnTouchListener로부터 콜백됨. 이것은 사용자가 아이템의 경계내에서 스크린을 누르고, 놓고, 또는 어떤 움직임 행위를 포함하는 터치 이벤트로써 적격한 액션을 수행할 때 호출된다.

onCreateContextMenu()

View.OnCreateContextMenuListener로부터 콜백됨. 이것은 일관된 "오랜시간 클릭long click"의 결과로, 컨텍스트 메뉴가 만들어져 있을 때 호출된다. 컨텍스트 메뉴에 대한 더 많은 정보에 대해서는 "메뉴 생성하기"를 보라.

이러한 메쏘드들은 그것들에 대응되는 인터페이스에 대한 단독 상주물sole inhabitant이다. 이러한 메쏘드 중 하나를 정의하고 여러분의 이벤트를 처리하기 위해, 여러분의 액티비티 내에 네스티드nested된 인터페이스를 구현하거나 익명의 클래스로 그것을 정의하라. 그런 다음에, 대응되는 View.set...Listener() 메쏘드에 여러분이 구현한 것을 파라미터로 전달하라(예를 들어 setOnClickListener()를 호출하

고, 그것에 여러분이 구현한 OnClickListener를 파라미터로 전달하라).

아래의 예제는, 버튼에 대한 on-click 리스너를 등록하는 방법을 보여준다.

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
      // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedValues) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

여러분은 또한 OnClickListener를 액티비티의 일부로써 구현하는 것이 더 편리하다는 것을 아마도 알게 될 것이다. 이것은 별도의 클래스 로드load와 오브젝트 할당을 피할 것이다. 예를 들어 다음과 같다.

```
public class ExampleActivity extends Activity
                                implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

앞에 있는 예제 안의 onClick() 콜백은 어떤 리턴 값도 가지지 않음에 주의하라. 하지만 다른 어떤 이벤트 리스너 메쏘드는 불리언Boolean 값을 반환해야 한다. 그 이유는 해당 이벤트에 의존한다. 그렇게 하는 몇가지 것에 대한 이유가 여기 있다.

- **onLongClick()** – 이것은 여러분이 그 이벤트를 소비해서 그것이 더 이상 전달되어서는 안되는지 여부를 가리키는 불리언Boolean 값을 리턴한다. 그것은 여러분이 그 이벤트를 처리해서 그것이 여기에서 멈춰야 한다는 것을 가리키기 위해서는 참true을 리턴한다. 만약 여러분이 그것을 처리하지 않았다면, 그리고/또는 그 이벤트가 어떤 다른 on-click 리스너에게 계속되어야 한다면 거짓false을 리턴한다.
- **onKey()** – 이것은 여러분이 그 이벤트를 소비해서 그것이 더 이상 전달되어서는 안되는지 여부를 가리키는 불리언Boolean 값을 리턴한다. 그것은 여러분이 해당 이벤트를 처리해서 그것이 여기에서 멈춰야한다는 것을 가리키기 위해서는 참true를 리턴한다. 만약 여러분이 그것을 처리하지 않았다면, 그리고/또는 그 이벤트가 어떤 다른 on-key 리스너에게 계속되어야 한다면 거짓false를 리턴한다.
- **onTouch()** – 이것은 여러분이 그 이벤트를 사용했는지 여부를 가리키는 불리언Boolean 값을 리턴한다. 중요한 것은, 이 이벤트는 각기 다른 것을 수반하는 복수의 액션을 가질 수 있다는 것이다. 그러므로 여러분이 다운down 액션 이벤트가 수신될 때 거짓false을 리턴하면, 여러분은 그 이벤트를 사용하지 않았고, 또한 이 이벤트에 수반되는 액션에 관심이 없음을 가리킨다. 따라서, 여러분은 손가락 제스쳐gesture나 업up 액션 이벤트와 같은 어떤 다른 액션도 그 이벤트 내에서 호출받지 못할 것이다.

핵심key 이벤트들이 항상 현재 포커스되어 있는 뷰에 전달된다는 것을 잊지 말아라. 그것들은 뷰 계층구조의 최상위에서 시작해서, 그것이 적절한 목적지에 도착할 때까지 내려가면서 전달dispatch된다. 만약 현재 여러분의 뷰 또는 그 뷰의 자식child이 포커스를 가지고 있다면, 여러분은 dispatchKeyEvent() 메쏘드를 통해 이벤트 전달 경로를 볼 수 있다. 여러분의 뷰에서 키 이벤트들을 얻기 위한 별도의 대안으로써, 여러분은 또한 onKeyDown()와 onKeyUp()를 사용해서 여러분의 액티비티 내부의 모든 키Key 이벤트를 수신할 수 있다.

NOTE

안드로이드는 첫 번째로 이벤트 핸들러를 호출할 것이다. 그리고 나서 그 클래스 정의로부터 적합한 디폴트 핸들러를 두 번째로 호출할 것이다. 따라서, 이러한 이벤트 리스너로부터 참true을 리턴하는 것은 다른 이벤트 리스너들에게 이벤트를 전달하는 것을 중지시킬 것이며, 뷰에 있는 기본 이벤트 핸들러에 대한 콜백도 차단할 것이다. 그러므로 여러분이 참true을 리턴할 때, 여러분이 이벤트를 종료시키길 원하는 지를 확인하라.

이벤트 핸들러(Handler)

만약 여러분이 뷰View에 대한 커스텀 컴포넌트를 만든다면, 여러분은 디폴트 이벤트 핸들러로 사용되는 몇몇 콜백 메쏘드를 정의하는 것이 가능할 것이다. "커스텀 컴포넌트 만들기" 섹션에서, 여러분은 아래와 같은 이벤트 처리를 위해 사용되는 몇 가지 일반적인 콜백들을 보고 배울 것이다.

- **onKeyDown(int, KeyEvent)** – 신규 키 이벤트가 발생할 때 호출된다.
- **onKeyUp(int, KeyEvent)** – 키 업(up) 이벤트가 발생할 때 호출된다.
- **onTrackballEvent(MotionEvent)** – 트랙볼trackball 모션 이벤트가 발생할 때 호출된다.
- **onTouchEvent(MotionEvent)** – 터치 스크린 모션 이벤트가 발생할 때 호출된다.
- **onFocusChanged(boolean, int, Rect)** – 뷰가 포커스를 얻거나 잃게 될 때 호출된다.

여러분이 알아야 하는 약간의 그 밖의 메쏘드들이 있다. 그것은 뷰 클래스의 일부분은 아니지만, 이벤트를 처리할 수 있는 방법에 직접적으로 영향을 미칠 수 있다. 그러므로 레이아웃 내부에서 보다 더 복잡한 이벤트를 관리할 때, 이러한 또 다른 메쏘드들을 고려하라.

- **Activity.dispatchTouchEvent(MotionEvent)** – 이것은 터치 이벤트가 윈도우로 전달되기 전에 여러분의 액티비티가 그것들을 가로채는 것을 허용한다.

- ViewGroup.onInterceptTouchEvent(MotionEvent) – 이것은 이벤트가 자식child 뷰에게 전달될 때 해당 이벤트를 그것의 부모parent 뷰가 감시하는 것을 허용한다.
- ViewParent.requestDisallowInterceptTouchEvent(boolean) –부모 뷰가 onInterceptTouchEvent(MotionEvent)를 사용해서 터치 이벤트를 가로채서는 안된다는 것을 가리키기 위해서는 이것을 호출하라.

터치(Touch) 모드

사용자가 방향 키 또는 트랙볼을 사용해서 사용자 인터페이스를 네비게이션할 때, 입력을 받아들일 수 있는 것을 사용자가 볼 수 있도록 하기 위해 버튼과 같이 액션이 가능한 아이템에 포커스를 주는 것이 필요하다. 하지만 만약 디바이스가 터치 기능을 가지고 있고 사용자가 그 디바이스를 터치함으로써 인터페이스와 상호작용을 시작한다면, 더 이상 아이템들을 하이라이트하거나 특정 뷰에 포커스를 줄 필요는 없다. 따라서 "터치 모드touch mode"라고 명명된 상호작용에 대한 모드가 있다.

터치 기능을 가진 디바이스에서, 일단 사용자가 스크린을 터치하면 디바이스는 터치 모드로 진입하게 될 것이다. 이 시점부터는, 텍스트 편집 위젯과 같이 isFocusableInTouchMode()가 참true인 뷰들만이 포커스될 것이다. 버튼들과 같이 터치가 가능한 그 밖의 뷰들은 터치되었을 때, 포커스를 가지지 않을 것이다; 그것들이 눌려질 때, 그것들은 자신의 on-click 리스너를 간단하게 실행할 것이다.

사용자가 방향 키를 누르거나 또는 트랙볼을 가지고 스크롤을 하면 언제든 해당 디바이스는 터치모드를 빠져나갈 것이고, 그리고 나서 포커스를 가질 뷰를 찾을 것이다. 이제 사용자는 스크린 터치없이 사용자 인터페이스와의 상호작용을 재개resume할 수 있다.

터치 모드 상태는 전체 시스템(모든 윈도우들과 액티비티들)에 걸쳐 유지된다. 여러분은 현재 상태를 질의query하기 위해, 해당 디바이스가 현재 터치 모드에 있는지 여부를 알 수 있게 해 주는 isInTouchMode()를 호출할 수 있다.

포커스(Focus) 처리

프레임워크는 사용자 입력에 반응하여 일상적인 포커스 움직임을 처리할 것이다. 이것은 뷰가 제거되거나 숨겨질 때 또는 새로운 뷰가 존재할 때, 포커스를 변경하는 것을 포함한다. 뷰는 isFocusable() 메쏘드를 통해 포커스를 가지고자 하는 의향이 있는가를 표현한다. 뷰가 포커스를 가질 수 있는지 없는지를 변경시키기 위해서 setFocusable()을 호출한다. 터치 모드일 때, 여러분은 isFocusableInTouchMode()를 사용해서 뷰가 포커스를 허용하는 지 여부를 질의query할 수 있다. 여러분은 setFocusableInTouchMode()를 가지고 이것을 바꿀 수 있다.

포커스 이동은 주어진 방향에서 가장 가깝게 인접한 것을 찾는 알고리즘에 기반한다. 특이한 경우, 디폴트 알고리즘은 개발자가 의도했던 동작과 일치하지 않을 수 있다. 이런 상황에서, 여러분은 레이아웃 파일 안에서 nextFocusDown, nextFocusLeft, nextFocusRight, nextFocusUp과 같은 XML 애트리뷰트를 사용해서 명시적인 오버라이드를 제공할 수 있다. 포커스가 떠나게 되는 뷰에 이러한 애트리뷰트 중 하나를 추가하라. 그 애트리뷰트의 값은 포커스가 건네져야 하는 뷰의 id가 되도록 정의하라. 예를 들어 다음과 같다.

```
<LinearLayout
    android:orientation="vertical"    ... >
  <Button android:id="@+id/top"
          android:nextFocusUp="@+id/bottom"
          ... />
  <Button android:id="@+id/bottom"
          android:nextFocusDown="@+id/top"
          ... />
</LinearLayout>
```

보통 위와 같은 버티컬vertical 레이아웃에서는, 첫 번째 버튼으로부터 위로 움직여서는 어디로도 갈 수 없고, 두 번째 버튼으로부터 아래로 움직여서도 어디로도 갈 수 없다. 이제 최상위 버튼은 nextFocusUp으로 아래쪽 버튼을 정의했고 그 반대도 마찬가지이기 때문에, 네비게이션 포커스는 위로부터 아래로top-to-bottom 그리고 아래로부터 위로bottom-to-top 사이클cycle을 돌 것이다.

만약 UI에서 (전통적으로는 포커스가 되지 않지만) 뷰를 포커스 가능한 것으로 선언하고 싶다면 여러분의 레이아웃 선언에 android:focusable XML 애트리뷰트를 뷰에 추가하라. 그리고 값을 참true로 설정하라. 여러분은 또한 android:focusableInTouchMode를 사용해서 터치모드 동안에도 뷰를 포커스 가능한 것으로 선언할 수 있다.

특정 뷰가 포커스를 가지도록 요청하기 위해서는, requestFocus()를 호출하라.

(뷰가 포커스를 얻거나 놓을 때 통보되는) 포커스 이벤트를 청취할려면, 위의 이벤트 리스너 섹션에서 설명된 것과 같이 onFocusChange()를 사용하라.

사용자에게 통보하기

몇몇 유형의 상황들이 여러분으로 하여금 사용자에게 애플리케이션에서 발생한 이벤트를 통보하게 하는 일을 만들 수 있다. 약간의 이벤트들은 사용자의 응답을 요구하며, 그 밖의 것들은 그렇지 않다. 예를 들어 다음과 같다.

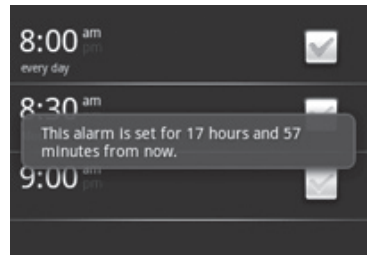
- 파일을 저장하는 것과 같은 이벤트가 완료되었을 때는, 저장이 성공했다는 것을 확인시켜 주기 위해 작은 메시지를 표시해야 한다.
- 만약 애플리케이션이 백그라운드로 실행되고 있고 사용자의 주의attention를 필요로 한다면, 그 애플리케이션은 사용자로 하여금 편한 시간에 응답할 수 있게 하는 노티피케이션notification을 생성해야 한다.
- 만약 애플리케이션이 (파일을 로드(load)하는 것과 같이) 사용자가 대기해야 하는 작업을 수행하고 있다면, 그 애플리케이션은 떠 있는hovering 진행상태 바퀴progress wheel 또는 진행상태바progress bar를 보여주어야 한다.

이러한 노티피케이션 업무 각각은 다른 기법을 사용해서 이루어질 수 있다.

- **토스트 노티피케이션(Toast Notification)**, 백그라운드로부터 오는 짧은 메시지를 위한 것.
- **상태바 노티피케이션(Status Bar Notification)**, 백그라운드로부터 와서 사용자에게 응답 요청을 지속적으로 상기reminder시키기 위한 것.
- **다이얼로그 노티피케이션(Dialog Notification)**, 액티비티와 관련된 노티피케이션

이곳에서는 사용자에게 통보하는 것에 대한 이러한 각각의 기법을 요약하며, 몇 가지에 대해서는 조금 더 상세하게 설명한다.

토스트 노티피케이션(Toast Notification)

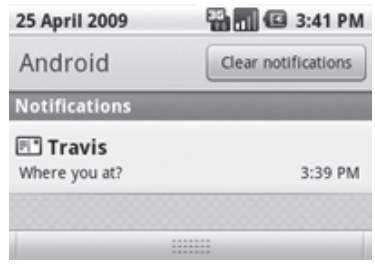


Toast 노티피케이션은 윈도우 표면에 팝업되는 하나의 메시지이다. 그것은 오직 메시지를 위해 요구되는 양 만큼의 공간space을 채우며, 사용자의 현재 액티비티는 보여지며 상호작용 가능한 상태로 유지된다. 노티피케이션은 자동으로 나타났다가 사라진다. 그리고 그것은 상호작용 이벤트를 받아들이지는 않는다. Toast는 백그라운드 서비스로부터 생성될 수 있기 때문에, 그것은 애플리케이션이 보이지 않을 때 조차도 나타난다.

Toast는 사용자가 스크린에 주의를 기울이고 있는 것을 여러분이 명확하게 확신할 때, "파일이 저장되었습니다File saved."와 같은 짧은 텍스트 메시지를 표시하는 데에는 최선이다. Toast는 사용자와의 상호작용 이벤트를 수용할 수 없다. 만약 여러분이 사용자의 응답과 액션을 얻고자 한다면, 대신 상태바Status Bar 노티피케이션을 사용하는 것을 고려하라.

더 많은 정보에 대해서는 "Toast 노티피케이션 생성하기"를 참고하라.

상태바(Status Bar) 노티피케이션

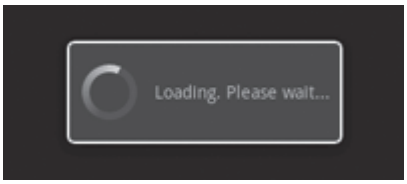


상태바Status Bar 노티피케이션은 시스템의 상태 바(선택사항으로 조회 텍스트(ticker-text) 메시지를 가짐)에 아이콘을, 그리고 "노티피케이션notifications" 윈도우에는 확장 메시지를 추가한다. 그 사용자가 확장 메시지를 선택할 때, 안드로이드는 노티피케이션에 정의된 인텐트를 발생시킨다(보통 액티비티를 런치하기 위해). 여러분은 또한 디바이스 상의 사운드sound, 진동vibration, 그리고 발광 빛flashing light을 사용해서 사용자에게 경보alert하도록 노티피케이션을 설정할 수 있다.

이런 종류의 노티피케이션은 여러분의 애플리케이션이 백그라운드 서비스Service에서 작업하고 있으면서 사용자에게 이벤트를 통보하는 것이 필요할 때 이상적이다. 만약 여러분의 액티비티가 여전히 포커스되어 있는 동안 발생하는 이벤트를 사용자에게 경보alert하는 것이 필요하다면, 대신에 다이얼로그Dialog 노티피케이션을 사용하는 것을 고려하라.

더 많은 정보에 대해서는, "상태바status bar 노티피케이션"을 참고하라.

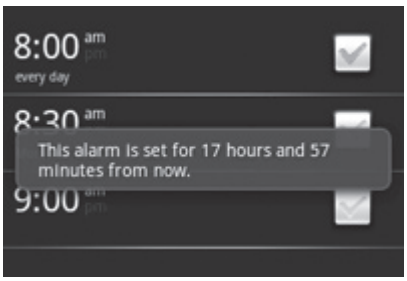
다이얼로그(Dialog) 노티피케이션



다이얼로그는 보통 현재 액티비티 앞에 나타나는 하나의 작은 윈도우이다. 그 아래 놓이게 되는 액티비티는 포커스를 잃으며(역주 : 액티비티가 포커스를 잃음에도 불구하고 이는 액티비티의 생명주기에는 영향을 미치지 않음), 그리고 그 다이얼로그는 사용자와의 모든 상호작용을 받아들인다. 다이얼로그는 보통 노티피케이션과 진행중인 애플리케이션에 직접적으로 관련된 짧은 액티비티들을 위해 사용된다.

여러분이 진행상태바progress bar 또는 ("OK" 와 "Cancel" 버튼과 같은) 사용자로 부터의 확인을 요구하는 짧은 메시지를 보여주는 것이 필요할 때, 다이얼로그를 사용해야 한다. 여러분은 또한 노티피케이션 이외의 다른 목적을 위해, 여러분의 애플리케이션 UI에서 필수적인 컴포넌트로 다이얼로그를 또한 사용할 수 있다. 노티피케이션을 위한 다이얼로그의 사용을 포함해서, 사용가능한 모든 타입의 다이얼로그에 대한 완전한 논의에 대해서는, "다이얼로그 생성하기"를 참고하라.

토스트(Toast) 노티피케이션 생성



Toast 노티피케이션은 윈도우 표면에 팝업되는 하나의 메시지이다. 그것은 오직 메시지를 위해 요구되는 양 만큼의 공간space을 채우며, 사용자의 현재 액티비티는 보여지며 상호작용 가능한 상태로 유지된다. 노티피케이션은 자동으로 나타났

다가 사라진다. 그리고 그것은 상호작용 이벤트를 받아들이지는 않는다.

앞에 있는 스크린샷은 알람Alarm 애플리케이션의 Toast 노티피케이션 예제를 보여준다. 일단 알람이 켜지면, 여러분에게 알람이 설정되었다는 것을 확인시키기 위해 Toast가 표시된다.

Toast는 액티비티Activity 또는 서비스Service로부터 생성되고 표시될 수 있다. 만약 여러분이 서비스로부터 Toast 노티피케이션을 생성한다면, 그것은 현재 포커스된 액티비티 앞에 보여진다.

만약 노티피케이션에 대한 사용자의 응답이 요구된다면, 상태바Status Bar 노티피케이션을 사용하는 것을 고려하라.

기초(The Basics)

먼저, makeText() 메쏘드들 중의 하나를 사용해서 Toast 오브젝트를 인스턴스화하라. 이 메쏘드는 세 개의 파라미터를 갖는다: 애플리케이션 컨텍스트, 텍스트 메시지, 그리고 Toast의 존속 기간이 그것이다. 그것은 적절하게 초기화된 Toast 오브젝트를 리턴한다. 여러분은 아래의 예제에서 보여지듯이, show()를 사용해서 Toast 노티피케이션을 표시할 수 있다.

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

이 예제는 대부분의 Toast 노티피케이션을 위해 필요한 모든 것을 여러분에게 보여준다. 여러분은 좀처럼 그외의 어떤 것을 필요하지 않는다. 하지만 여러분은 Toast의 위치를 다르게 설정하길 원하거나 또는 심지어 간단한 텍스트 메시지 대신에 여러분 자신의 레이아웃을 사용하길 원할 수도 있다. 아래의 섹션은 여러분이 이러한 것을 할 수 있는 방법을 설명한다.

여러분은 또한 아래와 같이 메쏘드들을 연결chain해서 Toast 오브젝트를 보유하

는 것을 피할 수 있다.

```
Toast.makeText(context, text, duration).show();
```

토스트 위치 부여하기(Positioning your Toast)

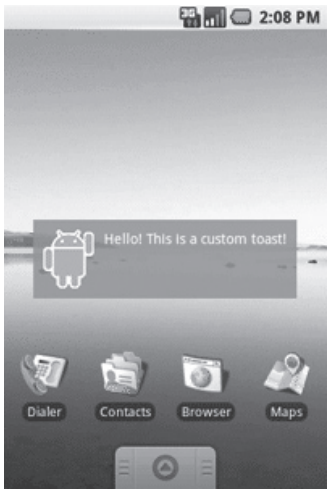
표준 Toast 노티피케이션은 스크린의 하단 가까이 수평으로 가운데 위치에 표시된다. 여러분은 setGravity(int,int,int) 메쏘드를 사용해서 이 위치를 변경할 수 있다. 이것은 세 개의 파라미터를 받아들인다: 그라비티gravity 상수, x 좌표 오프셋, y 좌표 오프셋이 그것이다.

예를 들어 만약 여러분이 Toast가 왼쪽 상단 모서리에 나타나야 한다고 결정한다면, 여러분은 아래와 같이 그라비티를 설정할 수 있다.

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

만약 여러분이 그 위치를 오른쪽으로 조금씩 이동시키기를 원한다면, 두 번째 파라미터의 값을 증가시켜라. 그리고 그것을 아래로 조금씩 움직이길 원한다면, 마지막 파라미터의 값을 증가시켜라.

커스텀 토스트 뷰 생성하기(Creating a Custom Toast View)



만약 하나의 간단한 텍스트 메시지가 충분치 않다면, 여러분의 Toast 노티피케이션을 위한 커스터마이즈된 레이아웃을 생성할 수 있다. 커스텀 레이아웃을 생성하기 위해서는, XML 파일 또는 여러분의 애플리케이션 코드에서 뷰 레이아웃을 정의하고, 그리고 setView(View) 메쏘드에 루트root 뷰 오브젝트를 파라미터로 전달하라.

예를 들어 여러분은 다음에 나오는 XML(toast_layout.xml로 저장됨)을 사용해서 오른쪽에 있는 스크린샷에 보이는 Toast에 대한 레이아웃을 생성할 수 있다.

```
<LinearLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
               android:id="@+id/toast_layout_root"
               android:orientation="horizontal"
               android:layout_width="fill_parent"
               android:layout_height="fill_parent"
               android:padding="10dp"
               android:background="#DAAA" >
    <ImageView android:id="@+id/image"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_marginRight="10dp"
                />
    <TextView android:id="@+id/text"
               android:layout_width="wrap_content"
               android:layout_height="fill_parent"
               android:textColor="#FFF"
               />
</LinearLayout>
```

LinearLayout의 ID가 "toast_layout" 이라는 것에 주의하라. 여러분은 아래에서 보여지듯이 XML로부터 그 레이아웃을 인플레이트하기 위해 이 ID를 사용해야 한다.

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_layout,
       (ViewGroup) findViewById(R.id.toast_layout_root));

ImageView image = (ImageView) layout.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Hello! This is a custom toast!");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

먼저 getLayoutInflator()(또는 getSystemService())를 사용해서 LayoutInflator를 얻어라. 그런 다음에 inflate(int,ViewGroup)을 사용해서 XML로부터 그 레이아웃을 인플레이트inflate하라. inflate() 메쏘드의 첫 번째 파라미터는 레이아웃 리소스 ID이다. 그리고 그 두 번째는 루트root 뷰이다. 여러분은 그 레이아웃에서 더 많은 뷰 오브젝트들을 찾기 위해 이 인플레이트된 레이아웃을 사용할 수 있으며, 그렇게 한 후 이미지뷰ImageView와 텍스트뷰TextView 엘리먼트에 대한 컨텐트를 이제 캡쳐하고 정의하라. 최종적으로 Toast(Context)를 사용해서 새로운 Toast를 생성하라. 그리고 gravity와 duration과 같은 Toast의 약간의 속성들을 설정하라. 그런 다음 setView(View)를 호출하여 그 메쏘드에 인플레이트된 레이아웃을 파라미터로 전달하라. 여러분은 이제 show()를 호출함으로써 여러분의 커스텀 레이아웃을 사용한 Toast를 표시할 수 있다.

만약 여러분이 setView(View)를 사용해서 레이아웃을 정의할 것이 아니라면, Toast에 대한 퍼블릭(public) 생성자를 사용하지 마라. 만약 여러분이 사용할 커스텀 레이아웃을 가지고 있지 않다면, 여러분은 Toast를 생성하기 위해 makeText(Context, int, int)를 사용해야 한다.

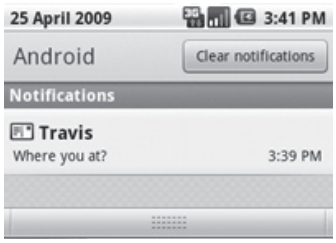
상태바(Status Bar) 노티피케이션 생성

상태바Status Bar 노티피케이션은 시스템의 상태바(선택 사항으로 조회 텍스트(ticker-text) 메시지를 가짐)에 아이콘을, 그리고 "노티피케이션notifications" 윈도우에는 확장 메시지를 추가한다. 그 사용자가 확장 메시지를 선택할 때, 안드로이드는 노티피케이션에 정의된 인텐트를 발생시킨다(보통 액티비티를 런치하기 위해). 여러분은 또한 디바이스 상의 사운드sound, 진동vibration, 그리고 발광 빛flashing light을 사용해서 사용자에게 경보alert하도록 노티피케이션을 설정할 수 있다.

상태바 노티피케이션은 백그라운드 서비스Service가 사용자에게 응답을 요구하는 이벤트를 경보alert 하는 것이 필요한 경우에 사용되어야 한다. 백그라운드 서비스는 사용자와의 상호작용을 받아들이기 위해 그것 자신의 액티비티를 결코 런치해서는 안된다. 서비스는 대신 사용자에 의해 선택될 때, 액티비티를 런치할 상태바 노티피케이션을 생성해야 한다.

아래의 스크린샷은 왼쪽 사이드에 노티피케이션 아이콘을 가진 상태바를 보여준다.

아래의 스크린샷은 "노티피케이션Notifications" 윈도우에 있는 노티피케이션의 확장 메시지를 보여준다. 사용자는 상태바를 아래로 내림(또는 홈 옵션 메뉴에서 노티피케이션을 선택함)으로써 노티피케이션 윈도우를 나타나게 할 수 있다.



기초(The Basics)

액티비티 또는 서비스는 상태바 노티피케이션을 시작시킬 수 있다. 액티비티는 그것이 활성화되어 있고 포커스되어 있는 동안에만 액션을 수행할 수 있기 때문에, 여러분은 서비스로부터 여러분의 상태바 노티피케이션을 생성해야 한다. 이 방식에서, 노티피케이션은 사용자가 다른 애플리케이션을 사용하거나 또는 그 디바이스가 절전상태에 있는 동안 백그라운드로부터 생성될 수 있다. 노티피케이션을 생성하기 위해서, 여러분은 두 가지 클래스를 사용해야 한다: Notification과 NotifactionManager가 그것이다.

상태바status bar 아이콘, 확장 메시지expanded message, 그리고 재생할 사운드와 같은 별도의 설정과 같이, 여러분의 상태 바 노티피케이션의 속성들을 정의하기 위해서는 노티피케이션Notification 클래스의 인스턴스를 사용하라. 노티피케이션 매니저NotificationManager는 모든 노티피케이션을 실행하고 관리하는 안드로이드 시스템 서비스이다. 여러분은 노티피케이션 매니저를 인스턴스화하지는 않는다. 여러분의 노티피케이션을 노티피케이션 매니저에게 건네주기 위해, 여러분은 getSystemService()를 사용하여 노티피케이션 매니저에 대한 레퍼런스를 얻어와야 한다. 그런 다음에 여러분이 사용자에게 통보하길 원할 때, notify()를 사용해서 노티피케이션 매니저에게 여러분의 노티피케이션 오브젝트를 전달하라.

상태바 노티피케이션을 생성하기 위해서는 다음과 같이 하라.

1. 노티피케이션 매니저에 대한 레퍼런스를 얻어라.

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(ns);
```

2. 노티피케이션을 인스턴스화하라.

```
int icon = R.drawable.notification_icon;
CharSequence tickerText = "Hello";
long when = System.currentTimeMillis();
Notification notification = new Notification(icon, tickerText, when);
```

3. 노티피케이션의 확장 메시지와 인텐트를 정의하라.

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent
   = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(context, contentTitle,
                                    contentText, contentIntent);
```

4. 노티피케이션 매니저에게 노티피케이션을 전달하라.

```
private static final int HELLO_ID = 1;
mNotificationManager.notify(HELLO_ID, notification);
```

이것이 끝이다. 여러분의 사용자는 이제 노티피케이션을 통보받았다notified.

노티피케이션 관리하기(Managing your Notifications)

노티피케이션 매니저NotificationManager는 모든 노티피케이션을 관리하는 시스템 서비스이다. 여러분은 getSystemService()를 사용해서 노티피케이션 매니저에 대한 레퍼런스를 얻어와야 한다. 예를 들어 다음과 같다.

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager
        = (NotificationManager) getSystemService(ns);
```

여러분이 상태바 노티피케이션을 전송하고자 할 때, 노티피케이션 매니저에게 notify(int,Notification)를 사용해서 노티피케이션 오브젝트를 파라미터로 전달하라. 첫 번째 파라미터는 노티피케이션에 대한 고유한 ID이며, 두 번째는 노티피케이션 오브젝트이다. 그 ID는 여러분의 애플리케이션으로부터 그 노티피케이션을 고유하게 식별한다. 이 ID는 여러분이 노티피케이션을 업데이트하는 것이 필요하거나 또는 (여러분의 애플리케이션이 다른 종류의 노티피케이션들을 관리한다면) 사용자가 노티피케이션에 정의된 인텐트를 통해 여러분의 애플리케이션으로 돌아올 때 적절한 액션을 선택하는 것이 필요할 때 필수적이다.

사용자가 노티피케이션 윈도우에서 상태바 노티피케이션을 선택할 때 그것을 없애기 위해서는, 여러분의 노티피케이션 오브젝트에 "FLAG_AUTO_CANCEL" 플래그를 추가하라. 여러분은 또한 cancel(int)를 사용해서 상태바 노티피케이션에 노티피케이션 ID를 전달함으로써 수동으로 그것을 없앨 수 있으며, 또한 cancelAll()를 사용해서 여러분의 모든 노티피케이션을 없앨 수 있다.

노티피케이션 생성(Creating a Notification)

노티피케이션Notificaton 오브젝트는 상태바에 표시되는 노티피케이션 메시지의 상세 사항과 "노티피케이션Notifcations" 윈도우, 그리고 사운드와 깜박이는 발광 빛과 같은 그 밖의 경보 설정을 정의한다.

상태바 노티피케이션은 다음과 같은 모든 것을 요구한다.

- 상태 바에 대한 아이콘.
- 확장 뷰에 대한 타이틀과 확장 메시지(만약 여러분이 커스텀 확장 뷰를 정의하지 않는다면).
- 노티피케이션이 선택될 때 일어나도록 하기 위한 PendingIntent.

상태바 노티피케이션에 대한 선택적인 설정에는 다음과 같은 것이 포함된다.

- 상태바에 대한 조회 텍스트(ticker-text) 메시지

- 경보 사운드
- 진동 설정
- 발광 LED 설정

신규 노티피케이션에 대한 시작 도구는 Notification(int,CharSequence,long) 생성자와 setLatestEventInfo(Context,CharSequence,CharSequence, PendingIntent) 메쏘드를 포함한다. 이것들은 노티피케이션에 대해 요구되는 모든 설정을 정의한다. 아래의 단편snippet은 기본적인 노티피케이션 셋업을 보여준다.

```
int icon = R.drawable.notification_icon;        // icon from resources
CharSequence tickerText = "Hello";               // ticker-text
long when = System.currentTimeMillis();          // notification time
Context context = getApplicationContext();       // application Context
CharSequence contentTitle = "My notification";
                                        // expanded message title
CharSequence contentText = "Hello World!";
                                        // expanded message text
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent
       = PendingIntent.getActivity(this, 0, notificationIntent, 0);

// the next two lines initialize the Notification,
// using the configurations above
Notification notification = new Notification(icon, tickerText, when);
notification.setLatestEventInfo(context, contentTitle,
                                contentText, contentIntent);
```

노티피케이션 업데이트하기(Updating the notification)

이벤트들이 여러분의 애플리케이션에서 계속 발생함에 따라 여러분은 상태바 노티피케이션에 있는 정보를 업데이트할 수 있다. 예를 들어 이전 메시지가 읽혀지기 전에 신규 SMS 텍스트 메시지가 도착하면, 그 메시징 애플리케이션은 수신된 신규 메시지의 전체 숫자를 보여주기 위해 이미 존재하는 노티피케이션을 업데이트한다. 이미 존재하는 노티피케이션을 업데이트하는 이 방식은, 노티피케이션 윈도우에서의 혼란clutter을 피하기 때문에 노티피케이션 매니저에 신규 노티피케이션을 추가하는 것보다 훨씬 더 좋다.

각각의 노티피케이션은 정수integer ID를 가지고 노티피케이션 매니저에 의해 고유하게 식별되기 때문에, 여러분은 신규 값을 가지고 setLatestEventInfo()를 호출함으로써 노티피케이션을 수정하고, 그 노티피케이션의 일부 필드 값들을 변경하고, 그리고 나서 다시 notify()를 호출할 수 있다.

여러분은 (컨텍스트와 확장 메시지 타이틀과 텍스트를 제외하고) 오브젝트 멤버 필드를 사용해서 각각의 속성을 수정할 수 있다. contentTitle과 contentText에 대한 신규 값을 가지고 setLatestEventInfo()를 호출함으로써 여러분이 그 노티피케이션을 업데이트할 때, 여러분은 항상 그 텍스트 메시지를 변경revise해야 한다. 그런 다음 그 노티피케이션을 업데이트 하기 위해 notify()를 호출하라(물론, 만약 여러분이 커스텀 확장 뷰를 생성했다면, 이러한 타이틀과 텍스트 값을 업데이트하는 것은 아무런 영향을 주지 않는다).

사운드 추가하기(Adding a sound)

여러분은 디폴트 노티피케이션 사운드(이것은 사용자에 의해 정의된다) 또는 여러분의 애플리케이션에 의해 지정된 사운드를 사용해서 사용자에게 경보alert할 수 있다.

사용자의 디폴트 사운드를 사용하기 위해서는, 디폴트 필드에 "DEFAULT_SOUND"를 추가하라.

```
notification.defaults |= Notification.DEFAULT_SOUND;
```

여러분의 노티피케이션에서 다른 사운드를 사용하기 위해서는, 사운드 필드에 Uri 레퍼런스를 전달하라. 다음의 예제는 디바이스 SD 카드에 저장된 알고 있는 오디오 파일을 사용한다.

```
notification.sound = Uri.parse("file:///sdcard/notification/ringer.mp3");
```

다음의 예제에서는, 그 오디오 파일이 내부 MediaStore 컨텐트 프로바이더에서 선택된다.

```
notification.sound
        = Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");
```

이 경우에, 미디어 파일 ("6")에 대한 정확한 ID를 알고 있으며, 그리고 컨텐트 Uri의 끝에 덧붙여진다. 만약 여러분이 정확한 ID를 모른다면, 여러분은 컨텐트

리졸버ContentResolver를 사용해서 미디어스토어MediaStore에서 사용가능한 모든 미디어를 쿼리해야 한다. 컨텐트 리졸버ContentResolver를 사용하는 것에 대한 더 자세한 정보에 대해서는 7장. "컨텐트 프로바이더"를 보라.

만약 여러분이 사용자가 노티피케이션에 응답할 때까지 또는 노티피케이션이 취소될 때까지 사운드가 계속해서 반복되길 원한다면, 그 플래그 필드에 "FLAG_INSISTENT"를 추가하라.

주의 만약 그 디폴트 필드가 "DEFAULT_SOUND" 를 포함한다면, 그러면 그 디폴트 사운드는 사운드 필드에 의해 정의된 임의의 사운드를 오버라이드 한다.

진동 추가하기(Adding vibration)

여러분은 디폴트 진동 패턴 또는 여러분의 애플리케이션에 의해 정의된 진동 패턴을 사용해서 사용자에게 경보alert할 수 있다.

디폴트 패턴을 사용하기 위해서는, 디폴트 필드에 "DEFAULT_VIBRATE"를 추가하라.

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
```

여러분 자신의 진동 패턴을 정의하기 위해서는 vibrate 필드에 long 값 배열을 전달하라.

```
long[] vibrate = {0,100,200,300};
notification.vibrate = vibrate;
```

long 배열은 진동이 꺼지고 켜지는 길이에 대한 번갈아 있는 패턴을 (1/1000초 단위로) 정의한다. 첫 번째 값은 시작하기 전에 얼마나 오래 기다릴 것인가(꺼진 상태)이고, 두 번째 값은 첫 번째 진동의 길이이고, 세 번째는 그 다음의 꺼져 있는 길이, 그리고 기타 등등이다. 그 패턴은 여러분이 좋아하는 만큼 길 수 있으나 반복되도록 설정될 수 없다.

주의 만약 디폴트 필드가 "DEFAULT_VIBRATE"를 포함한다면, 그러면 그 디폴트 진동은 vibrate 필드에 의해 정의된 임의의 진동을 오버라이드 한다.

발광 빛 추가하기(Adding flashing lights)

발광 LED 빛을 사용해서 사용자에게 경보alert하기 위해서, 여러분은 (만약 사용할 수 있다면) 디폴트 빛 패턴을 구현할 수 있거나 또는 여러분 자신의 컬러와 그 빛에 대한 패턴을 정의할 수 있다.

디폴트 빛을 설정하기 위해서는 그 디폴트 필드에 "DEFAULT_LIGHTS"를 추가하라.

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
```

여러분 자신의 컬러와 패턴을 정의하기 위해서는 (컬러에 대해서는) ledARGB 필드, ledOffMS 필드(1/1000초 단위의 빛이 꺼져 유지되는 시간의 길이), ledOnMS(1/1000초 단위의 빛이 켜져 유지되는 시간의 길이) 값을 정의하고, 그리고 또한 그 플래그 필드에 "FLAG_SHOW_LIGHTS"를 추가하라.

```
notification.ledARGB = 0xff00ff00;
notification.ledOnMS = 300;
notification.ledOffMS = 1000;
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

이 예제에서, 초록색 빛이 반복적으로 300 밀리세컨드(1/1000초) 동안 켜지고 1초 동안 꺼진다. 스펙트럼 내의 모든 컬러가 디바이스 LED에 의해 지원되는 것은 아니다. 그리고 모든 디바이스가 동일 컬러를 지원하지 않는다. 그러므로 그 하드웨어는 자신의 능력의 최선을 산정한다. 초록색은 가장 일반적인 노티피케이션 컬러이다.

더 많은 기능(More features)

여러분은 노티피케이션 필드와 플래그를 사용해서 여러분의 노티피케이션에 더 많은 몇가지 기능들을 추가할 수 있다. 약간의 유용한 기능들이 아래에 포함되어 있다.

"FLAG_AUTO_CANCEL" 플래그

노티피케이션 윈도우에서 노티피케이션이 선택된 이후, 그 노티피케이션이 자동으로 취소되게끔 하기 위해서는 이 플래그를 flags 필드에 추가하라.

"FLAG_INSISTENT" 플래그

사용자가 응답할 때까지 오디오를 반복시키기 위해서는 이 플래그를 flags 필드에 추가하라.

"FLAG_ONGOING_EVENT" 플래그

노티피케이션 윈도우의 "Ongoing" 타이틀 아래의 노티피케이션을 그룹화하기 위해서는 이것을 그 플래그에 추가하라. 이것은 애플리케이션이 진행중on-going이라는 것을 가리킨다 – (뮤직이나 전화 호출과 같이) 심지어 애플리케이션이 보이지 않을 때 조차도, 그 애플리케이션 프로세스는 여전히 백그라운드에서 실행중이다.

"FLAG_NO_CLEAR" 플래그

노티피케이션이 "Clear" 노티피케이션 버튼에 의해서 없어져서는 안된다는 것을 가리키기 위해서는 이 플래그를 flags 필드에 추가하라. 만약 여러분의 노티피케이션이 진행중이라면 이것은 특별히 유용하다.

number 필드

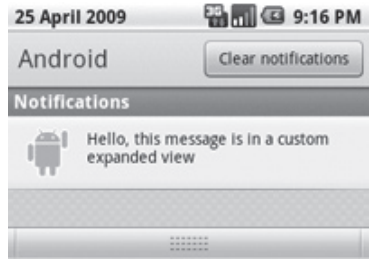
이 값은 노티피케이션에 의해 보여지는 현재의 이벤트 갯수를 가리킨다. 그에 알맞는 숫자가 상태바 아이콘 상단에 오버레이된다. 만약 여러분이 이 필드를 사용하고자 한다면, 여러분은 노티피케이션이 처음 생성될 때 "1"을 가지고 시작해야 한다(만약 여러분이 그 값을 업데이트 동안에 0에서 더 큰 어떤 수로 바꾼다면, 그 숫자는 보여지지 않는다).

iconLevel 필드

이 값은 노티피케이션 아이콘을 위해 사용되는 LevelListDrawable의 현재 레벨을 가리킨다. 여러분은 LevelListDrawable에 정의된 드로어블에 대응하는 이 값을 변경함으로써 상태바에서 아이콘을 애니메이션 할 수 있다. 더 많은 정보에 대해서는 LevelListDrawable 레퍼런스를 보라.

여러분의 애플리케이션을 커스터마이징할 수 있는 추가적인 기능들에 대한 더 자세한 정보에 대해서는 노티피케이션Notification 클래스 레퍼런스를 보라.

커스텀 확장 뷰 생성하기(Creating a Custom Expanded View)



디폴트로, "노티피케이션Notifications" 윈도우에서 사용되는 확장 뷰는 기본적인 타이틀과 텍스트 메시지를 포함한다. 이것들은 setLatestEventInfo() 메쏘드의 contentTitle과 contentText 파라미터에 의해 정의된다. 하지만 여러분은 또한 리모트뷰RemoteViews를 사용해서 확장 뷰에 대한 커스텀 레이아웃을 정의할 수 있다. 위에 보이는 스크린샷은 LinearLayout에 이미지 뷰와 텍스트 뷰를 사용한 커스텀 확장 뷰의 예제이다.

확장 메시지에 대한 여러분 자신의 레이아웃을 정의하기 위해서는, 리모트뷰RemoteViews 오브젝트를 인스턴스화하고, 여러분의 노티피케이션의 컨텐트뷰contentView 필드에 리모트뷰RemoteView를 전달하라. contentIntent 필드에는 PendingIntent를 전달하라.

커스텀 확장 뷰를 생성하는 것은 예제를 통해 가장 잘 이해된다.

1. 확장 뷰에 대한 XML 레이아웃을 생성하라. 예를 들어 custom_notification_layout.xml 이라 불리는 레이아웃 파일을 생성하고, 그것을 다음과 같이 만들어라.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="3dp"
            >
    <ImageView android:id="@+id/image"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_marginRight="10dp" />
    <TextView android:id="@+id/text"
            android:layout_width="wrap_content"
```

```
                android:layout_height="fill_parent"
                android:textColor="#000"
                />
</LinearLayout>
```

이 레이아웃은 확장 뷰를 위해 사용된다. 그러나 이미지 뷰와 텍스트 뷰의 컨텐트는 여전히 애플리케이션에 의해 정의될 필요가 있다. 리모트뷰RemoteViews는 이 컨텐트를 여러분으로 하여금 정의할 수 있도록 하는 약간의 편리한 메쏘드들을 제공한다.

2. 애플리케이션 코드에서, 이미지와 텍스트를 정의하기 위해서는 리모트뷰RemoteViews 메쏘드를 사용하라. 그런 다음에 아래의 예제에서 보여지듯이 리모트뷰RemoteViews 오브젝트를 노티피케이션의 컨텐트뷰contentView 필드에 전달하라.

```
RemoteViews contentView
    = new RemoteViews(getPackageName(),
                              R.layout.custom_notification_layout);
contentView.setImageViewResource(R.id.image,
            R.drawable.notification_image);
contentView.setTextViewText(R.id.text,
            "Hello, this message is in a custom expanded view");

notification.contentView = contentView;
```

여기에서 보여지듯이, 애플리케이션의 패키지 이름과 레이아웃 리소스 ID를 리모트뷰RemoteViews 생성자constructor에게 전달하라. 그런 다음에, setImageViewResource()와 setTextViewText()를 사용해서 이미지 뷰와 텍스트 뷰에 대한 내용을 정의하라. 각각의 경우에서, 해당 뷰에 대한 값과 함께 여러분이 설정하고자 하는 적절한 뷰 오브젝트의 레퍼런스 ID를 전달하라. 최종적으로 그 리모트뷰RemoteViews 오브젝트는 컨텐트뷰contentView 필드로 노티피케이션에게 전달된다.

3. 여러분이 커스텀 뷰를 사용할 때 setLatestEventInfo() 메쏘드를 필요로 하지 않기 때문에, 여러분은 이 예제에서처럼 contentIntent 필드를 사용해서 노티피케이션에 대한 인텐트를 정의해야 한다.

```
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notifica-
tionIntent, 0);
notification.contentIntent = contentIntent;
```

4. 노티피케이션은 이제 보통의 경우처럼 전송될 수 있다.

```
mNotificationManager.notify(CUSTOM_VIEW_ID, notification);
```

리모트뷰RemoteViews 클래스는 또한 여러분의 노티피케이션의 확장 뷰에 측정기Chronometer 또는 진행상태바ProgressBar를 쉽게 추가하기 위해 사용할 수 있는 메쏘드들을 포함한다. 리모트뷰RemoteViews 를 사용해서 커스텀 레이아웃을 생성하는 것에 대한 더 자세한 정보에 대해서는 리모트뷰RemoteViews 클래스 레퍼런스를 참고하라.

주의 커스텀 확장 뷰를 생성할 때 여러분은 특별히 여러분의 커스텀 레이아웃이 다른 디바이스 오리엔테이션과 해상도에서 적절하게 기능하는 지를 보장하기 위해 주의해야 한다. 이 조언은 안드로이드 상에 생성된 모든 뷰 레이아웃에 적용되지만, 그것은 특별히 이 경우에 중요하다. 왜냐하면 여러분의 레이아웃 실소유물real estate이 매우 제한되기 때문이다. 그러므로 여러분의 커스텀 레이아웃을 너무 복잡하게 만들지 마라. 그리고 다양한 설정에서 그것을 확실히 테스트하라.

스타일과 테마 적용하기

애플리케이션을 디자인할 때, 여러분은 그것의 각종 스크린 및 UI 엘리먼트에 단일한 포맷을 적용하기 위해 스타일과 테마를 사용할 수 있다.

- 스타일은 여러분의 레이아웃 XML 파일에 있는 하나의 엘리먼트 단위로 적용할 수 있는 하나 또는 그 이상의 포맷팅 애트리뷰트의 집합이다. 예를 들어 여러분은 특정 텍스트 크기와 그리고 컬러를 지정하는 스타일을 정의할 수 있고, 그런 다음 그것을 특정 유형의 뷰 엘리먼트의 인스턴스에 적용할 수 있다.
- 테마는 여러분이 애플리케이션 내의 모든 액티비티 또는 단지 하나의 액티비티 단위로 적용할 수 있는 하나 또는 그 이상의 포맷팅 애트리뷰트의 집합이다. 예

를 들어 여러분은 윈도우 프레임과 패널 포어그라운드foreground와 백그라운드background에 대해 특정 색깔을 지정하고, 메뉴에 대해 텍스트 크기와 색을 지정하는 테마를 정의할 수 있으며, 그런 다음 여러분의 애플리케이션의 액티비티들에 그것을 적용할 수 있다.

스타일과 테마는 리소스들이다. 안드로이드는 여러분이 사용할 수 있는 약간의 디폴트 스타일과 테마 리소스들을 제공한다. 또한 여러분은 자신의 커스텀 스타일과 테마 리소스들을 정의할 수 있다.

커스텀 스타일과 테마를 만들기 위해서는,

1. 여러분의 애플리케이션의 res/values 디렉토리에 styles.xml 라는 이름의 파일을 생성하라. 그리고 파일 안에 〈resources〉 루트root 노드를 추가하라.
2. 각 스타일 또는 테마에 대해, 고유한 name 과 선택 사항으로 parent 애트리뷰트를 가지는 〈style〉 엘리먼트를 추가하라. name은 나중에 이 스타일을 레퍼런스하기 위해 사용되며, parent는 스타일 리소스가 상속받을 것을 가리킨다.
3. 〈style〉 엘리먼트 내부에, 하나 또는 그 이상의 〈item〉 엘리먼트에 포맷 값들을 정의하라. 각각의 〈item〉은 name 애트리뷰트를 가지고 그것의 스타일 속성을 구분하며, 해당 엘리먼트 안에 그것의 값을 정의한다.
4. 여러분은 그런 다음에 다른 XML 리소스, 여러분의 매니페스트 또는 애플리케이션 코드로부터 커스텀 리소스를 레퍼런스할 수 있다.

스타일

여기에 스타일 선언declaration 예제가 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="SpecialText" parent="@style/Text">
        <item name="android:textSize">18sp</item>
        <item name="android:textColor">#008</item>
    </style>
</resources>
```

위에서 보여지듯이, 여러분은 스타일에 대한 특정 포맷팅 값들을 지정하기 위해 〈item〉 엘리먼트를 사용할 수 있다. item의 name 애트리뷰트는 표준 문자열, 16진수 컬러 값, 또는 임의의 다른 리소스 타입에 대한 레퍼런스를 참조할 수 있다.

〈style〉 엘리먼트에 있는 parent 애트리뷰트에 주의하라. 이 애트리뷰트는 어떤 리소스로부터 현재 스타일이 값들을 상속할 것인가를 여러분이 지정할 수 있게 한다. 스타일은 여러분이 원하는 스타일을 포함하는 어떤 유형의 리소스로 부터도 상속할 수 있다. 일반적으로 여러분의 스타일은 (직접 또는 간접적으로) 항상 표준 안드로이드 스타일 리소스로 부터 상속해야 한다. 이런 방법으로 여러분은 단지 바꾸고자 하는 값들을 정의해야 한다.

여기에 여러분이 XML 레이아웃에서 커스텀 스타일을 레퍼런스하는 방법이 있다. 이 경우에는, EditText 엘리먼트에 대한 것이다.

```
<EditText id="@+id/text1"  style="@style/SpecialText"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="Hello, World!" />
```

이제 이 EditText 위젯은, 위의 XML 예제에 의해 정의되는 것으로 스타일될 것이다.

테마(Theme)

마치 스타일과 같이 테마 또한 XML 〈style〉 엘리먼트로 선언되며, 그리고 같은 방식으로 레퍼런스된다. 그 차이는 여러분이 테마를 안드로이드 매니페스트 내에 있는 〈application〉과 〈activity〉 엘리먼트를 통해, 전체 애플리케이션 또는 액티비티에 추가한다는 것이다 – 테마는 개별적인 뷰에는 적용될 수 없다.

여기에 테마 선언declaration 예제가 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomTheme">
    <item name="android:windowNoTitle">true</item>
```

```
    <item name="windowFrame">@drawable/screen_frame</item>
    <item name="windowBackground">@drawable/screen_background_white
      </item>
    <item name="panelForegroundColor">#FF000000</item>
    <item name="panelTextColor">?panelForegroundColor</item>
    <item name="panelTextSize">14</item>
    <item name="menuItemTextColor">?panelTextColor</item>
    <item name="menuItemTextSize">?panelTextSize</item>
  </style>
</resources>
```

리소스를 레퍼런스하기 위한 @심볼at-symbol과 ?마크question-mark의 사용에 주목하라. @심볼은 이전에 다른 곳(이것은 현재 프로젝트이거나 또는 안드로이드 프레임워크일 수 있음)에서 정의된 리소스를 우리가 레퍼런스한다는 것을 가리킨다. ?마크는 현재 로드load된 테마에서의 리소스 값을 우리가 레퍼런스한다는 것을 가리킨다. 이것은 그것의 name 값에 의해 특정 〈item〉을 레퍼런스함으로써 수행된다(예를 들어 panelTextColor는 앞서 정의된 panelForegroundColor에 지정된 동일한 색깔을 사용한다). 이 기법은 XML 리소스에서만 사용될 수 있다.

매니페스트에서의 테마 설정(Set the theme in the manifest)

여러분 애플리케이션의 모든 액티비티에 대해 이 테마를 설정하기 위해서는, AndroidManifest.xml 파일을 오픈해서 〈application〉 태그를 편집해서 테마 이름을 가지는 android:theme 애트리뷰트를 포함시켜라.

```
<application android:theme="@style/CustomTheme">
```

만약 여러분의 애플리케이션에 있는 오직 하나의 액티비티에만 테마를 적용하고자 한다면, 대신 〈activity〉 태그에 그 테마 애트리뷰트를 추가하라.

안드로이드는 다른 내장된built-in 리소스를 제공하기 때문에, 여러분이 직접 그것을 만들지 않고 추가할 수 있는 몇 가지 테마가 있다. 예를 들어 여러분의 액티비티를 다이얼로그 박스처럼 만들기 위해 Dialog 테마를 사용할 수 있다. 매니페스트 내에서 다음과 같이 안드로이드 테마를 레퍼런스하라.

```
<activity android:theme="@android:style/Theme.Dialog">
```

만약 여러분이 어떤 테마를 좋아하지만 그것을 가볍게 고치고 싶다면, 여러분

의 커스텀 테마의 parent에 해당 테마를 추가하라. 예를 들어 우리는 Theme.Dialog 테마를 수정할 것이다. 그렇게 하기 위해서는 parent로 Theme.Dialog를 가지는 스타일을 만들어라.

```
<style name="CustomDialogTheme" parent="@android:style/Theme.Dialog">
```

바로 그것이다. 우리가 좋아하는 대로 해당 스타일을 고칠 수 있도록 안드로이드 Dialog 테마를 상속했다. 그러므로 우리가 변경하고자 하는 Dialog 테마에 있는 각각의 item에 대해, 우리는 여기서 그 값을 새로 정의하고 안드로이드 매니페스트 내에서 Thema.Dialog 대신 CustomDialogTheme를 사용한다.

애플리케이션에서의 테마 설정(Set the theme from the application)

여러분은 또한 만약 필요하다면 프로그램적으로 액티비티를 위한 테마를 로드할 수 있다. 그렇게 하기 위해서 setTheme() 메쏘드를 사용하라. 그렇게 할 때, 여러분은 컨텍스트 내에서 임의의 뷰를 인스턴스화하기 전에 테마를 설정해야 한다는 것에 주의하라. 예를 들어 setContentView(View)를 호출하기 전 또는 inflate(int,ViewGroup) 전에 설정해야 한다. 이것은 시스템이 여러분의 모든 UI 스크린에 대해 동일한 테마를 적용하는 것을 보장한다. 여기에 예제가 있다.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);    ...
    setTheme(android.R.style.Theme_Light);
    setContentView(R.layout.linear_layout_3);
}
```

만약 애플리케이션의 메인 스크린을 위해 프로그램적으로 임의의 테마를 로드하는 것을 고려한다면, 시스템이 액티비티를 시작하기 위해 사용할 어떤 애니메이션에는 테마가 적용되지 않는다는 것에 주의하라. 그것은 여러분의 애플리케이션이 오픈되기 전에 일어난다. 대부분의 경우, 만약 메인 스크린에 테마를 적용하고자 한다면, XML에서 그렇게 하는 것이 더 좋은 접근법이다.

커스텀 스타일과 테마, 그리고 그것을 여러분의 애플리케이션에서 레퍼런스하는 것에 대한 자세한 정보는 4장. "리소스와 에셋"의 "사용가능한 리소스 타입"중 "스타일과 테마"를 보라. 사용가능한 디폴트 테마와 스타일에 대한 정보에 대해서는 R.style을 보라.

커스텀 컴포넌트 제작

안드로이드는 기초적인 레이아웃 클래스인 뷰View와 뷰그룹ViewGroup에 기반해서 여러분의 UI 제작을 위한 정교하고 강력한 컴포넌트화된 모델을 제공한다. 우선 안드로이드 플랫폼은 여러분의 UI를 만드는 데 사용할 수 있는 다양한 미리 만들어진 뷰와 뷰그룹 서브클래스 – 각각은 위젯과 레이아웃이라 불린다 – 를 포함하고 있다.

사용가능한 위젯의 일부 리스트에는 Button, TextView, EditText, ListView, CheckBox, RadioButton, Gallery, Spinner, 그리고 더 특수한 목적의 AutoCompleteTextView,ImageSwitcher, 그리고 TextSwitcher가 포함된다.

사용가능한 레이아웃에 속하는 것에는 LinearLayout, FrameLayout, RelativeLayout, 그리고 그 밖의 것들이 있다. 더 많은 예제들에 대해서는 "일반적인 레이아웃 오브젝트"를 보라..

만약 미리 만들어진 위젯 또는 레이아웃 중 어떤 것도 여러분의 필요를 충족시키지 못한다면, 여러분 자신의 뷰 서브클래스를 만들 수 있다. 만약 여러분이 기존의 위젯이나 레이아웃을 조금 개선할 필요가 있다면 여러분은 위젯 또는 레이아웃의 서브클래스를 간단히 만들어서 그것의 메쏘드를 오버라이드할 수 있다.

여러분 자신의 뷰 서브클래스를 만드는 것은, 스크린 엘리먼트의 모양과 기능에 대한 정확한 제어를 여러분에게 가능하게 한다. 커스텀 뷰를 사용해서 여러분이 얻을 제어에 대한 하나의 아이디어를 주기 위해, 여기에 여러분이 그것을 사용해서 할 수 있는 약간의 예제가 있다.

- 여러분은 완전하게 직접 렌더링 된custom-rendered 뷰 타입, 예를 들어 아날로그 전자 장치와 비슷한 2D 그래픽을 사용해 만들어진 "볼륨 제어volume control" 손잡이를 만들 수 있다.
- 여러분은 아마 콤보박스ComboBox(팝업 리스트와 프리 텍스트 입력 필드의 조합), 듀얼-페인 실렉터 컨트롤dual-pane selector control(왼쪽과 우른쪽 판 각각에 리스트를 가지고, 그 리스트의 아이템을 다른 곳에 재지정할 수 있는 것) 등과 같은 어떤 것을 만들기 위해, 새로운 하나

의 컴포넌트 안으로 뷰 컴포넌트의 그룹을 조합할 수 있다.

- 여러분은 EditText 컴포넌트의 스크린 렌더링 방식을 오버라이드 할 수 있다(노트패드(Notepad) 튜토리얼은 페이지에 멋진 효과와 밑줄을 주기 위해 이것을 사용한다).
- 여러분은 키 입력과 같은 다른 이벤트를 캡쳐할 수 있으며, 그것을 약간의 커스텀 방식으로 제어할 수 있다(게임에서 처럼).

아래의 섹션은 커스텀 뷰를 만드는 방법과 여러분의 애플리케이션에서 그것을 사용하는 방법을 설명한다. 더 자세한 레퍼런스 정보에 대해서는, 뷰View 클래스를 보라.

기본적 접근법(The Basic Approach)

여기에 있는 것은 여러분 자신의 뷰 컴포넌트를 생성하기 위해 알아야 할 필요가 있는 것에 대한 상위 레벨에서의 개요이다.

1. 여러분 자신의 클래스를 사용해서 기존의 뷰View 클래스 또는 서브클래스를 확장extend하라.
2. 슈퍼클래스의 몇몇 메쏘드를 오버라이드 하라. 오버라이드 할 슈퍼클래스 메쏘드들은 'on'으로 시작한다. 예를 들어 onDraw(), onMeasure(), 그리고 onKeyDown()이 그것이다. 이것은 여러분이 액티비티Activity 또는 리스트 액티비티ListActivity에서 생명주기를 오버라이드해서 다른 기능들을 추가하는 on...events와 유사하다.
3. 여러분의 새로운 확장 클래스를 사용하라. 일단 완성되면, 여러분의 새로운 확장 클래스는 그것의 기반이 되었던 뷰를 대신해서 사용될 수 있다.

> **Tip**
> 확장 클래스는 액티비티 내에 내부inner 클래스로 정의될 수 있다. 이것은 그것에 대한 접근을 통제하기 때문에 유용하다. 하지만 필수사항은 아니다(여러분은 아마 여러분 애플리케이션 내에서 더 넓게 사용할 새로운 public 뷰를 만들고자 할 것이다).

완전하게 커스터마이즈된 컴포넌트

완전하게 커스터마이즈된 컴포넌트는, 여러분이 원하는 것이 어떤 것이든 보여줄 수 있는 그래픽컬한 컴포넌트를 제작하기 위해 사용될 수 있다. 여러분이 오래된 아날로그 게이지gauge처럼 보이는 그래픽한 VU 미터를 원하거나, 또는 여러분이 가라오케 머신을 가지고 노래할 수 있도록 단어를 따라 움직이는 바운싱bouncing 볼이 있는 sing-a-long 텍스트 뷰를 원한다고 가정하자. 어떤 쪽이든, 여러분이 원하는 것은 내장된built-in 컴포넌트를 어떻게 결합combine하더라도 할 수 없는 것이다.

다행히도, 여러분은 원하는 방식으로 보여지고 동작하는 컴포넌트를 쉽게 만들 수 있다. 아마도 그것은 여러분의 상상력과, 스크린 크기, 그리고 사용가능한 프로세싱 파워에 의해서만 제한될 뿐이다(결국에는 여러분의 애플리케이션이 여러분의 데스크탑 워크스테이션보다는 현격하게 작은 파워를 가진 어떤 것 위에서 실행될지도 모른다는 것을 기억하라).

완전하게 커스터마이즈된 컴포넌트를 생성하기 위해서:

1. 여러분이 확장할 수 있는 가장 일반적인 뷰는 놀랍지 않게도 뷰View이다. 그러므로 여러분의 새로운 슈퍼 컴포넌트를 만들기 위해, 여러분은 보통 이것을 확장하는 것에서 시작할 것이다.
2. 여러분은 XML 내에 애트리뷰트와 파라미터를 가질 수 있는 생성자를 제공할 수 있다. 그리고 여러분 자신의 그러한 애트리뷰트와 파라미터를 또한 사용할 수 있다(아마도 VU 미터의 컬러과 범위, 또는 바늘의 너비와 민감함 등이 이에 속할 것이다).
3. 여러분은 아마도 여러분 자신의 이벤트 리스너listener, 속성 접근자accessor 및 수정자modifier, 그리고 추가적으로 가능하다면 여러분의 컴포넌트 클래스 안에 보다 더 정교한 동작을 생성하고자 할 것이다.
4. 만약 여러분이 어떤 것을 보여주기 위한 컴포넌트를 원한다면, 여러분은 거의 확실히 onMeasure()를 오버라이드하길 원할 것이며, 또한 onDraw()를 오버라이드할 것이다. 둘 다 모두 디폴트 작동구조를 가지고 있지만, 디폴트 onDraw()는 아무 것도 하지 않을 것이며, 디폴트 onMeasure()는 항상 100x100 크기 - 아마 여러분이 원하는 것이 아닌 것 - 를 설정할 것이다.
5. 다른 on...메쏘드들 또한 필요에 따라 오버라이드 될 수 있다.

onDraw() 와 onMeasure() 확장하기

onDraw() 메쏘드는 여러분에게 캔버스Convas를 파라미터로 전달하며, 그 캔버스 위에 여러분이 원하는 어떤 것, 즉 2D 그래픽, 다른 표준 또는 커스텀 컴포넌트, 스타일드styled 텍스트, 또는 그 밖의 여러분이 생각할 수 있는 어떤 것이든 구현할 수 있다.

> NOTE
> 이것은 3D 그래픽에는 적용되지 않는다. 만약 여러분이 3D 그래픽을 이용하고 싶다면, 여러분은 뷰 대신에 서피스뷰(SurfaceView)를 확장해야만 하고, 별도의 쓰레드에서 그리기를 해야 한다. 보다 자세한 것에 대해서는 GLSurfaceViewActivity 예제를 보라.

onMeasure()는 약간 더 복잡하다. onMeasure()는 여러분의 컴포넌트와 그것의 컨테이너 사이의 렌더링 협약의 중요한 부분이다. onMeasure()는 그것이 포함하고 있는 영역을 효과적이고 정확하게 측정해서 레포트하도록 오버라이드 되어야 한다. 이것은 (onMeasure() 메쏘드에 전달된 파라미터 안에 있는) 부모parent의 요구조건과 일단 계산되어졌던 측정된 너비measured width와 높이measured height를 가지고 setMeasuredDimension() 메쏘드를 호출하는 요구조건에 의해, 약간 더 복잡하게 만들어진다. 만약 여러분이 오버라이드한 onMeasure() 메쏘드에서 이 메쏘드를 호출하지 않는다면, 그 결과는 측정 시점에 예외exception가 될 것이다.

상위 레벨에서 볼 때At a high level, onMeasure()를 구현하는 것은 아래와 같이 보인다.

1. 오버라이드된 onMeasure() 메쏘드는 너비width와 높이height에 대한 측정명세Measure Specification를 파라미터로 가지고 호출된다(측정 명세는 widthMeasureSpec와 heightMeasureSpec 으로 구성되며, 두가지 모두 크기를 나타내는 정수(integer) 코드이다). 측정명세는 여러분이 산출해야 하는 너비와 높이 치수measurement에 대한 제약 요건으로 취급되어야 한다. 이러한 명세가 요구할 수 있는 제약의 종류에 대한 전체 레퍼런스는 View.onMeasure(int, int) 아래의 레퍼런스 문서에서 찾을 수 있다(이 레퍼런스 문서는 또한 전체적인 치수 오퍼레이션을 설명하는 매우 좋은 일을 한다).
2. 여러분 컴포넌트의 onMeasure() 메쏘드는 컴포넌트를 렌더링하기 위해 요구되는 너비와 높이 치수measurement를 계산해야 한다. 그것은 비록 onMeasure()

에 전달된 그 명세의 범위를 초과exceed할 수 있음에도 불구하고, 그 명세 범위 내에 머무르도록 노력해야 한다(명세를 넘어서는 경우, 그 부모(parent)는 잘라내기, 스크롤링, 예외(exception) 발생, 또는 아마 다른 측정 명세를 사용해서 다시 시도하기 위한 onMeasure() 호출을 포함하는 어떤 것을 선택할 수 있다).

3. 일단 너비와 높이가 계산되면 setMeasuredDimension(int width, int height) 메쏘드가 계산된 치수를 가지고 호출되어야 한다. 이것을 하지 않으면, 예외가 발생할 것이다.

여기에, 뷰에서 프레임워크가 호출하는 일부 다른 표준 메쏘드들의 개요가 있다.

카테고리	메쏘드	설명
Creation	Constructors	이것에는 코드에서 뷰가 생성될 때 호출되는 생성자의 형태와 레이아웃 파일에서 뷰가 인플레이트(inflate)될 때 호출되는 형태가 있다. 두 번째 형태는 레이아웃 파일에 정의된 임의의 애트리뷰트를 분석하고 적용해야 한다.
	onFinishInflate()	뷰와 그것의 모든 자식들(children)이 XML에서 인플레이트(inflalate)된 이후 호출된다.
Layout	onMeasure(int, int)	뷰와 그것의 모든 자식들(children)에 대한 크기 요건를 결정하기 위해 호출된다.
	onLayout(boolean, int, int, int, int)	뷰가 그것의 모든 자식들(children)에게 크기와 위치를 할당해야 할 때 호출된다.
	onSizeChanged(int, int, int, int)	뷰의 크기가 바뀌었을 때 호출된다.
Drawing	onDraw(Canvas)	뷰가 그것의 컨텐트를 렌더링해야 할 때 호출된다.
Event processing	onKeyDown(int, KeyEvent)	새로운 키 이벤트가 발생할 때 호출된다.
	onKeyUp(int, KeyEvent)	키 업(key up) 이벤트가 발생할 때 호출된다.
	onTrackballEvent (MotionEvent)	트랙볼 모션 이벤트가 발생할 때 호출된다.
	onTouchEvent (MotionEvent)	터치 스크린 모션 이벤트가 발생할 때 호출된다.
Focus	onFocusChanged(boolean, int, Rect)	뷰가 포커스를 얻거나 잃을 때 호출된다
	onWindowFocusChanged (boolean)	뷰를 포함하고 있는 윈도우가 포커스를 얻거나 잃을 때 호출된다.

Attaching	onAttachedToWindow()	뷰가 윈도우에 첨부될 때 호출된다.
	onDetachedFromWindow()	뷰가 윈도우에서 분리될 때 호출된다.
	onWindowVisibilityChanged(int)	뷰를 포함하고 있는 윈도우의 가시성(visibility) 이 변경될 때 호출된다

커스텀 뷰 예제(A Custom View Example)

API Demos 안에 있는 CustomView 샘플은 커스터마이징된 뷰의 예제를 제공한다. 커스텀 뷰는 LabelView 클래스에서 정의된다.

LabelView 샘플은 커스텀 컴포넌트의 다양한 다른 측면을 보여준다.

- 완전한 커스텀 컴포넌트를 위한 뷰 클래스를 확장하기.
- 파라미터화된 생성자는 뷰 인플레이션inflation 파라미터(XML에서 정의된 파라미터)를 가진다. 이것들 중 일부는 뷰 슈퍼클래스에게 전달된다. 그러나 더 중요하게도, LabelView를 위해 정의된 몇 가지 커스텀 애트리뷰트가 있다.
- 라벨label 컴포넌트에 대해 여러분이 보고자 할 타입에 대한 표준 퍼블릭public 메쏘드. 예를 들어 setText(), setTextSize(), setTextColor() 등이 있다.
- 컴포넌트의 렌더링 크기를 결정 및 설정하기 위해 오버라이드된 onMeasure() 메쏘드(LabelView에서 private measureWidth() 메쏘드에 의해 실제 작업이 이루어진다는 것에 주의하라).
- 제공된 캔버스Canvas위에 라벨label을 그리기 위해 오버라이드된 onDraw() 메쏘드.

여러분은 그 샘플의 custom_view_1.xml 안에서 LabelView 커스텀 뷰의 몇 가지 샘플 사용법을 볼 수 있다. 특히, 여러분은 android: 네임스페이스 파라미터와 커스텀 app: 네임스페이스 파라미터 두 가지의 혼합을 볼 수 있다. 이러한 app: 파라미터는 LabelView가 인식하고 그것을 가지고 작업하는 커스텀 파라미터이며, 샘플에 대한 R 리소스 정의 클래스의 styleable 내부inner 클래스에서 정의된다.

합성(Compound) 컨트롤

만약 여러분이 완벽하게 커스터마이즈된 컴포넌트를 생성하는 것을 원하지 않고, 대신 기존에 있는 컨트롤 그룹을 구성하는 재사용가능한 컴포넌트를 함께 배치하고자 한다면, 합성 컴포넌트(혹은 합성 컨트롤)를 만드는 것이 여러분의 기대를 충족시킬 수 있다. 간단하게 말해서, 이것은 하나로 취급될 수 있는 아이템들의 논리적 그룹안에 다수의 더 원자적인atomic 컨트롤(또는 뷰)를 합쳐 놓는다. 예를 들어 콤보박스는 단일 라인 EditText 필드에 PopupList가 붙은 인접한 버튼의 조합으로 생각될 수 있다. 만약 여러분이 버튼을 누르거나 리스트로부터 무언가를 선택한다면, 그것은 EditText 필드에 존재할 것이다. 하지만 사용자들은 또한 원하다면 EditText 안에 직접적으로 어떤 것을 입력할 수 있다.

안드로이드에는, 이것을 하기 위해 이미 사용가능한 두 개의 다른 뷰가 실제로 존재한다. Spinner와 AutoCompleteTextView가 그것이다. 하지만 이와 상관없이 콤보박스 개념은 이해하기 쉬운 예제를 제공한다.

합성 컴포넌트를 생성하기 위해서:

1. 일반적인 시작점은 어떤 종류의 레이아웃이다. 그러므로 하나의 레이아웃을 확장하는 클래스를 생성하라. 아마도 콤보박스의 경우에, 우리는 수평horizontal 오리엔테이션을 가진 LinearLayout를 사용할 수 있을 것이다. 그 합성 컴포넌트가 임의로 복잡하게 구조화될 수 있도록, 다른 레이아웃이 그 안에 포함될nested 수 있다는 것에 유념하라. 마치 액티비티에서와 같이, 여러분은 포함된 컴포넌트를 생성하기 위해 (XML 기반의)선언 방식을 사용하거나, 또는 여러분의 코드에서 그것을 프로그램적으로 포함할nest 수 있다는 것에 주목하라.

2. 새로운 클래스에 대한 생성자에서, 슈퍼클래스가 기대하는 파라미터가 무엇이든 그것을 얻은 후, 먼저 슈퍼클래스 생성자에게 그것을 전달하라. 그런 다음, 여러분의 새로운 컴포넌트 안에서 사용할 다른 뷰를 구성할 수 있다; 이곳이 여러분의 EditText 필드와 PopupList를 생성할 장소이다. 여러분은 또한 XML안에 여러분 자신의 애트리뷰트와 파라미터를 도입할 수 있으며, 그것을 가져와서 여러분의 생성자가 사용할 수 있다는 것에 주목하라.

3. 여러분은 또한 여러분이 보유한 뷰가 발생시킬 지도 모르는 이벤트에 대한 리

스너listener를 생성할 수 있다. 예를 들어 만약 리스트가 선택되면, EditText의 내용을 업데이트하기 위한 리스트 아이템 클릭 리스너에 대한 리스너 메쏘드를 생성할 수 있다.

4. 여러분은 또한 접근자accessor와 수정자modifier를 사용해서 여러분 자신의 속성들을 생성할 수 있다. 예를 들어 EditText 값이 컴포넌트 초기에 설정되도록 하고 필요할 때 그것의 내용물content을 요청query하도록 할 수 있다.

5. 레이아웃을 확장하는 경우에, 여러분은 onDraw()와 onMeasure() 메쏘드를 오버라이드 할 필요는 없다. 왜냐하면 그 레이아웃은 훌륭하게 작업할 수 있는 디폴트 작동구조을 가지고 있기 때문이다. 하지만 필요하다면 여러분은 여전히 그것을 오버라이드 할 수 있다.

6. 여러분은 특정 키가 눌려졌을 때, 콤보박스의 팝업 리스트에서 특정 디폴트 값을 선택하기 위해 onKeyDown()과 같은 다른 on...메쏘드를 오버라이드 할 수도 있다.

요약하면, 커스텀 컨트롤에 대한 기초로 레이아웃을 사용하는 것은 다음과 같은 다양한 장점을 가진다.

- 여러분은 액티비티 스크린에서와 같이 XML 선언 파일을 사용하여 레이아웃을 지정할 수 있으며, 또한 뷰를 프로그램적으로 생성해서 여러분 코드의 레이아웃에 그것을 포함할 수 있다.
- onDraw()와 onMeasure() 메쏘드(더불어 다른 모든 on...메쏘드)는 여러분이 그것을 오버라이드 할 필요가 없도록 적절하게 동작할 것이다.
- 끝으로, 여러분은 아주 빠르고 임의로 복잡한 합성 뷰를 생성할 수 있고, 마치 그것들이 단일 컴포넌트인 것처럼 그것을 재사용할 수 있다.

합성 컨트롤 예제(Examples of Compound Controls)

SDK에 포함되어 있는 API Demos 프로젝트에서는 두 개의 리스트 예제가 있다 – Views/Lists 아래에 있는 Example4와 Example6은 SpeechView를 보여준다. 그것은 연설문을 보여주는 컴포넌트를 만들기 위해 LinearLayout를 확장하고 있다. 샘플 코드 안의 각각에 대응되는 클래스는 List4.java와 List6.java이다.

이미 존재하는 뷰 타입 수정하기

어떤 상황에 유용한 커스텀 뷰를 생성하기 위한 더 쉬운 방법도 있다. 만약 여러분이 원하는 것과 매우 비슷한 컴포넌트가 이미 존재한다면, 여러분은 그 컴포넌트를 간단하게 확장extend해서 여러분이 바꾸고 싶어하는 것을 오버라이드 할 수 있다. 여러분은 완전하게 커스터마이즈된 컴포넌트를 사용해서 여러분이 원하는 모든 것을 할 수 있다. 그러나 뷰 계층구조에 더 전문화된 클래스를 사용함으로써, 여러분이 원하는 것을 아마도 정확하게 수행하는 더 많은 동작을 무상으로 또한 얻을 수 있다.

예를 들어 SDK 샘플 안에는 노트패드NotePad 애플리케이션이 포함되어 있다. 이것은 안드로이드 플랫폼을 사용하는 것의 다양한 측면을 보여준다. 그것들 중에는 밑줄이 있는 노트패드를 만들기 위해 EditText 뷰를 확장한 것이 있다. 이것은 완벽한 예제는 아니다. 그리고 이것에 대한 그 API는 이전 버전의 형태와 다르게 변경될 수도 있다. 하지만 그것은 그 원리를 설명한다.

여러분이 아직 사용해 보지 않았다면, 노트패드 샘플을 이클립스 안으로 가져와라(또는 제공된 링크를 사용해 소스를 보라). 특별히 NoteEditor.java 파일에 있는 MyEditText(역주: 현재는 LineEditText로 변경되어 있음)의 정의를 살펴보라

주목할 몇 가지가 여기에 있다.

1. 정의The Definition

   그 클래스는 아래의 줄로 정의된다.

```
public static class MyEditText extends EditText
```

- 이것은 NoteEditor 액티비티 내의 내부inner 클래스로 정의된다. 그러나 만약 필요하다면 NoteEditor 클래스의 외부에서 NoteEditor.MyEditText를 사용해서 접근될 수 있도록 퍼블릭public 으로 선언되어 있다.
- 이것이 정적static으로 선언되었다는 것은, 그것이 부모parent 클래스로부터의 데이터에 대한 접근을 가능하게 하는 "통합적synthetic 메쏘드"라 불리는 것을 만들어내지 않음을 의미한다. 이것은 NoteEditor와 강하게 연결된 클래스라

기 보다는 오히려 분리된 클래스로써 실제로 동작한다는 것을 의미한다. 이것은 메쏘드들이 외부 클래스로부터의 접근을 필요로 하지 않는다면 내부inner 클래스들을 생성하기 위한 더 깨끗한 방법이고, 생성된 클래스를 작게 유지하며 그 클래스를 다른 클래스로부터 쉽게 사용되도록 허용한다.

- 이것은 EditText를 확장한다. 이 경우에, 이것은 우리가 커스터마이즈하기 위해 선택한 뷰가 된다. 우리가 끝냈을 때, 새로운 클래스는 보통의 EditText 뷰를 대체하는 것이 가능하게 될 것이다.

2. 클래스 초기화Class Initialization

   항상 그렇듯이 super가 먼저 호출된다. 더 나아가, 이것은 디폴트 생성자가 아니라 파라미터화된 생성자이다. EditText는 그것이 XML 레이아웃 파일로부터 인플레이트inflate 될 때 이러한 파라미터들을 가지고 생성된다. 따라서, 우리의 생성자는 파라미터들을 얻는 것과 더불어 그것들을 슈퍼클래스 생성자에게 전달하는 것, 둘 다를 필요로 한다.

3. 오버라이드된 메쏘드Overridden Methods

   이 예제에는, 오버라이드된 하나의 메쏘드, 즉 onDraw()만이 있을 뿐이다 – 하지만 여러분 자신의 커스텀 컴포넌트를 만들 때 필요한 다른 것들이 쉽게 있을 수도 있다.

   노트패드 샘플에 대해, onDraw() 메쏘드를 오버라이드하는 것은 우리가 EditText 뷰 캔버스Canvas 위에 파란색 라인을 그리는 것을 가능하게 한다(캔버스는 오버라이드된 onDraw() 메쏘드에 전달된다). super.onDraw() 메쏘드는 그 메쏘드가 끝나기 전에 불려져야 한다. 그 슈퍼클래스 메쏘드는 불려져야 한다. 하지만 이런 경우에 우리는 우리가 포함하고 싶은 선을 그린 후 끝에서 그것을 한다.

4. 커스텀 컴포넌트 사용하기Use the Custom Component

   우리는 이제 우리의 커스텀 컴포넌트를 가지고 있다. 그러나 우리가 그것을 어떻게 사용할 수 있을까? 노트패드Nodepad 예제 안에서 커스텀 컴포넌트는 레이아웃 선언에서 직접 사용된다. res/layout 폴더에 있는 note_editor.xml를 살펴보라.

```
<view
  class="com.android.notepad.NoteEditor$MyEditText"
  id="@+id/note"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="@android:drawable/empty"
  android:padding="10dip"
  android:scrollbars="vertical"
android:fadingEdge="vertical" />
```

- 커스텀 컴포넌트는 XML에 있는 일반적인 뷰로써 생성된다. 그리고 해당 클래스는 전체 패키지를 사용해서 지정된다. 또한 우리가 정의한 내부inner 클래스는 자바 프로그램 언어에서 내부inner 클래스를 레퍼런스하는 표준 방법인 NoteEditor$MyEditText 표기법을 사용하여 레퍼런스된다는 것에 유의하라.

  만약 여러분의 커스텀 뷰 컴포넌트가 내부inner 클래스로 정의되지 않는다면, 여러분은 대신에 XML 엘리먼트 이름을 가진 뷰 컴포넌트를 선언할 수 있으며, class 애트리뷰트를 제외시킬 수 있다. 예를 들어 다음과 같다.

```
<com.android.notepad.MyEditText
  id="@+id/note"
... />
```

  MyEditText 클래스는 이제 분리된 클래스 파일이라는 것을 주의하라. 클래스가 NoteEditor 클래스에서 포함될 경우, 이 기법technique은 동작하지 않을 것이다.

- 그 정의 안에 있는 다른 애트리뷰트와 파라미터는 커스텀 컴포넌트 생성자에게 전달되는 것이며, 그리고 난 다음 EditText 생성자에게 전달된다. 그러므로 그것들은 여러분이 EditText 뷰에서 사용하는 것과 동일한 파라미터들이다. 마찬가지로 여러분 자신의 파라미터를 추가하는 것이 가능하다. 그리고, 우리는 다음에 이것을 다시 다룰 것이다.

그리고 이것이 그곳에 있는 전부이다. 틀림없이 이것은 간단한 사례이다. 하지만 그것이 핵심이다 – 커스텀 컴포넌트를 만드는 것은 여러분이 그것을 필요로 하는

만큼만 복잡해 진다.

더 정교한 컴포넌트는 심지어 더 많은 on... 메쏘드들을 오버라이드 할 지도 모르며, 실제로 그것의 속성과 동작을 커스터마이징하여 그것 자신의 보조 메쏘드들 일부를 도입할 수 있다. 유일한 한계는 여러분의 상상력과 여러분이 그 컴포넌트로 하여금 무엇을 하도록 하느냐이다.

어댑터뷰를 사용한 데이터 바인딩

어댑터뷰AdapterView는 뷰그룹ViewGroup의 서브클래스이다. 그것의 자식child 뷰들은 몇 가지 타입의 데이터를 바인드bind하는 어댑터Adapter에 의해 결정된다. 어댑터뷰는 여러분의 레이아웃안에 저장된 데이터(리소스 문자열들이나 드로어블(dwables)에 반대되는 개념)를 표시하고자 할 때 유용하다.

갤러리Gallery, 리스트뷰ListView, 그리고 스피너Spinner는 여러분이 특정 타입의 데이터를 바인드해서 그것을 어떤 방식으로 표시하기 위해 사용할 수 있는 어댑터뷰AdapterView의 서브클래스의 예제이다.

어댑터뷰AdapterView 오브젝트는 두 가지의 주요 책무를 가고 있다.

- 레이아웃을 데이터로 채우는 것
- 사용자가 선택한 것을 처리하는 것

데이터를 사용하여 레이아웃 채우기

레이아웃에 데이터를 삽입하는 것은 일반적으로 어댑터뷰AdapterView 클래스를 어댑터Adapter에 바인딩하는 것에 의해 일반적으로 이루어진다. 어댑터는 코드에서 제공하는 리스트 또는 디바이스의 데이터베이스에 대한 쿼리 결과와 같은 외부 소스로부터 데이터를 가져온다.

다음에 나오는 코드 샘플은 다음과 같은 것을 수행한다.

1. 이미 존재하는 뷰를 사용해서 스피너Spinner를 생성하고, 로컬 리소스의 컬러 배열을 읽어들이는 새로운 어레이어댑터ArrayAdapter에 해당 스퍼너를 바인드한다.

2. 또 다른 스피너Spinner 오브젝트를 뷰에서 생성하고, 디바이스의 컨택(Contacts. People를 보라)에서 사람들의 이름을 읽어들이는 새로운 심플커서어댑터 SimpleCursorAdapter에 해당 스피너를 바인드한다.

```
// Get a Spinner and bind it to an ArrayAdapter that
// references a String array.
Spinner s1 = (Spinner) findViewById(R.id.spinner1);
ArrayAdapter adapter = ArrayAdapter.createFromResource(
    this, R.array.colors, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(
   android.R.layout.simple_spinner_dropdown_item);
s1.setAdapter(adapter);

// Load a Spinner and bind it to a data query.
private static String[] PROJECTION = new String[] {
        People._ID, People.NAME
    };

Spinner s2 = (Spinner) findViewById(R.id.spinner2);
Cursor cur
   = managedQuery(People.CONTENT_URI, PROJECTION, null, null);

SimpleCursorAdapter adapter2 = new SimpleCursorAdapter(this,
    android.R.layout.simple_spinner_item, // Use a template
                                          // that displays a
                                          // text view
    cur, // Give the cursor to the list adatper
    new String[] {People.NAME}, // Map the NAME column in the
                                // people database to...
    new int[] {android.R.id.text1}); // The "text1" view defined in
                                     // the XML template
adapter2.setDropDownViewResource(
            android.R.layout.simple_spinner_dropdown_item);
s2.setAdapter(adapter2);
```

커서어댑터CursorAdapter를 함께 사용되는 프로젝션projection에서 People._ID 컬럼을 가지는 것이 필요하다는 것에 주목하라. 그렇지 않으면, 여러분은 예외exception을 얻을 것이다.

만약 여러분의 애플리케이션이 존속하는 동안 여러분의 어댑터Adapter에 의해 읽

혀지게 되는 기본적인 데이터를 여러분이 바꾼다면, 여러분은 notifyDataSet Changed()를 호출해야 한다. 이것은 첨부된 어댑터뷰AdapterView에게 데이터가 변경되었고, 그래서 스스로를 갱신해야 한다는 것을 통보할 것이다.

사용자가 선택한 것을 처리하기

여러분은 그 클래스의 AdapterView.OnItemClickListener 멤버를 리스너listener로 설정하고 사용자가 선택한 것에 대한 변화를 감지catching함으로써, 사용자가 선택한 것을 처리한다.

```
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClick-
Listener() {
    public void onItemClick(AdapterView parent, View v, int position,
long id)
    {
        // Display a messagebox.
        Toast.makeText(mContext,"You've got an event",
                       Toast.LENGTH_SHORT).show();
    }
};

// Now hook into our object and set its onItemClickListener member
// to our class handler object.
mHistoryView = (ListView)findViewById(R.id.history);
mHistoryView.setOnItemClickListener(mMessageClickedHandler);
```

다른 형태의 어댑터뷰AdapterViews를 만드는 방법에 대한 더 많은 논의에 대해서, 뒤에 나오는 Hello Spinner, Hello ListView, 그리고 Hello GridView와 같은 튜토리얼을 읽어라

일반적인 레이아웃 오브젝트

이 섹션은 여러분의 애플리케이션에서 사용할 수 있는 더 많은 일반적 유형의 레이아웃 오브젝트 중 몇 가지를 설명한다. 모든 레이아웃처럼, 그것들도 뷰그룹ViewGroup의 서브클래스이다. 더 많은 안드로이드 뷰 레이아웃을 사용하는 것에 대한 몇 가지 안내에 대해서 Hello Views 튜토리얼을 보라.

FrameLayout

FrameLayout은 가장 간단한 레이아웃 오브젝트 유형이다. 그것은 기본적으로 여러분의 스크린 상의 빈 공간이다. 여러분은 그곳에 하나의 오브젝트를 넣을 수 있다 – 예를 들어 여러분은 그 안에 사진을 넣거나 뺄 수 있다. FrameLayout의 모든 자식child 엘리먼트는 스크린의 상단 좌측 가장자리에 고정적으로 배치된다. 여러분은 자식child 뷰를 다른 위치에 지정할 수 없다. 그 다음의 자식child 뷰들은 간단하게 이전의 것 위에 부분적으로 또는 전체적으로 그것을 가리는 형태로 표시될 것이다(새로운 오브젝트가 투명하지 않는 한 그렇다).

LinearLayout

LinearLayout는 모든 자식들children을 하나의 방향으로 정렬한다 – 그 방향은, 여러분이 orientation 애트리뷰트를 어떻게 정의했느냐에 따라 수직 또는 수평이 된다. 모든 자식들children은 순차적으로 쌓여진다stacked. 그러므로 수직적 리스트는 그들이 넓이가 얼마든 상관없이 각 행은 하나의 자식child만 가질 것이다. 그리고 수평적 리스트는 오직 한 개의 행 높이를 가질 것이다(그 행 높이는, 가장 큰 자식(child)의 높이에 패딩(padding)을 더한 것이 된다). LinearLayout는 자식들 간의 마진margin과 각 자식의 그라비티gravity(right, center, 또는 left 정렬 값을 가짐)를 고려한다.

LinearLayout는 또한 개별적인 자식들children에게 가중치weight를 지정하는 것을 지원한다. 이 애트리뷰트는 뷰에 "중요도importance" 값을 지정하고, 부모parent 뷰에 있는 잔여 공간을 채울 수 있도록 그것을 확대expand하는 것을 가능하게 한다. 자식child 뷰는 정수integer 가중치weight 값을 지정할 수 있으며, 그러면 뷰그룹에 있는 임의의 잔여 공간은 그것들이 선언한 가중치weight 비율대로 자식들children에게 할당된다. 디폴트 가중치weight는 0이다. 예를 들어 3개의 텍스트 박스가 있고 그것들 중 2개가 1이라는 가중치weight를 선언하고 나머지가 가중치weight를 가지지 않는다면, 가중치weight를 가지지 않은 세 번째 텍스트 박스는 커지지 않을 것이고 그것의 내용물에 의해 요구되는 영역만을 점유할 것이다. 나머지 두 개는 모든 세 개의 박스가 측정된 이후 남는 공간을 동일하게 채울 수 있도록 확대될 것이다. 만약 세 번째 박스에 0 대신 2라는 가중치weight가 주어진다면, 그것은 이제 나머지 두 개 보다 "더 중요한more important" 것으로 선언된 것이며,

그것은 전체 잔여 공간의 절반을 갖게 되는 반면에 처음 두개는 나머지를 동등하게 공유한다.

> **Tip** 스크린에 잘 맞는 크기의 레이아웃을 생성하기 위해서는 layout_width와 layout_height 애트리뷰트가 fill_parent로 설정된 컨테이너 뷰그룹 오브젝트를 생성하라. 자식들children의 높이와 너비를 0(zero)으로 할당하라. 그런 후, 각각이 스크린 상에서 가져야 하는 비율에 의존해서 각각의 자식 child에게 상대적인 weight 값을 지정하라.

다음에 나오는 두개의 폼은 하나의 버튼, 몇 개의 라벨들, 그리고 텍스트 박스들과 같은 엘리먼트의 집합을 가진 LinearLayout을 보여준다. 텍스트 박스들은 fill_parent로 지정된 그것들의 너비를 가지고 있으며, 나머지 엘리먼트들은 wrap_content로 지정되어 있다. 디폴트로 gravity는 left이다. 두 개의 폼 버전간의 차이는, 오른쪽에 있는 폼이 가중치weight 1로 지정된 comments 텍스트 박스를 가진 반면에, 왼쪽에 있는 폼은 가중치weight 값을 가지고 있지 않다는 것(디폴트로 0임)이다. 만약 Name 텍스트 박스 또한 1로 지정되었다면, Name과 Comments 텍스트 박스는 동일한 높이를 가지게 될 것이다.

수평적인 LinearLayout 안에서 아이템들은 그것들의 텍스트 베이스 라인base line의 위치에 의해 정렬된다(최상위 또는 가장 왼쪽인, 첫 번째 리스트 엘리먼트의 첫 번째 라인이 레퍼런스 라인으로 간주된다). 이것은 하나의 형식으로 되어 있는 엘리먼트들을 살펴보고

있는 사람이, 인접하는neighboring 엘리먼트들에 있는 엘리먼트 텍스트를 읽기 위해 위와 아래로 움직이지 않도록 하기 위한 것이다. 이것은 레이아웃 XML 내의 android:baselineAligned="false" 설정에 의해 해제될 수 있다.

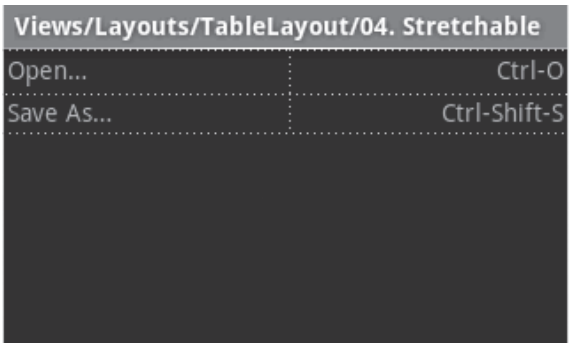
다른 샘플 코드를 보려면 Hello LinearLayout 튜토리얼을 보아라.

TableLayout

TableLayout는 행과 열 안에 그것의 자식들children을 배치한다. TableLayout 컨테이너는 그것들의 행, 열, 또는 셀에 대한 경계선을 표시하지 않는다. 테이블은 가장 많은 셀을 가진 행 만큼 많은 열을 가질 것이다. 테이블은 셀을 빈 공간으로 남겨둘 수 있다. 하지만 셀은 HTML에서 할 수 있는 것처럼, 행을 병합할 수는 없다. (역주: 원문에는 cells cannot span columns, 즉 셀은 열을 병합할 수 없다고 되어 있으나 잘못된 듯하다. layout_span 애트리뷰트가 열을 병합하는 데 사용된다. 아마도 행row을 병합할 수 없다로 정정되어야 할 듯함.)

TableRow 오브젝트는 TableLayout의 자식child 뷰이다(각 TableRow는 테이블 안의 하나의 행을 정의한다). 각 행은 0 또는 그 이상의 셀을 가지며, 그것들 각각은 임의의 종류의 다른 뷰에 의해 정의된다. 그러므로 행의 셀은 ImageView 또는 TextView 오브젝트처럼 다양한 뷰 오브젝트로 구성될 수도 있다. 셀은 또한 뷰 그룹ViewGroup 오브젝트일 수도 있다(예를 들어 여러분은 셀에 또 다른 TableLayout을 포함할 수 있다).

다음에 오는 샘플 레이아웃은 각각 2개의 행과 각각의 행에 2개의 셀을 가지고 있다. 아래에 있는 스크린샷은 그 결과를 보여주며, 점선(시각적 효과를 위해 추가된 것임)으로 셀 경계선을 표시하고 있다.



```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:text="@string/table_layout_4_open"
            android:padding="3dip"  />
        <TextView
            android:text="@string/table_layout_4_open_shortcut"
            android:gravity="right" android:padding="3dip" />
    </TableRow>

    <TableRow>
        <TextView
            android:text="@string/table_layout_4_save"
            android:padding="3dip" />
        <TextView
            android:text="@string/table_layout_4_save_shortcut"
            android:gravity="right" android:padding="3dip" />
    </TableRow>
</TableLayout>
```

열은 숨겨질 수도 있고, 확대stretch되도록 마크되어 사용가용한 스크린 공간을 채울 수도 있으며, 또는 테이블이 스크린에 맞춰질 때까지 행이 축소shrink되도록 하기 위해 축소가능shrinkable으로 마크될 수도 있다. 보다 자세한 것에 대해서 TableLayout 레퍼런스 문서를 보아라.

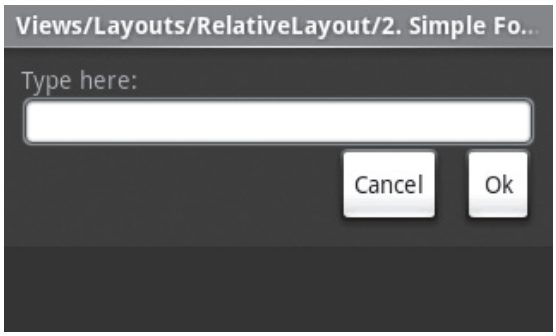
샘플 코드를 보기 위해서는 Hello TableLayout 튜토리얼을 보아라.

RelativeLayout

RelativeLayout는 자식child 뷰들의 위치를 부모parent 뷰 또는 상호간(ID에 의해 지정됨)에 상대적으로 지정하게 한다. 그러므로 여러분은 오른쪽 경계선에 두 개의 엘리먼트를 정렬할 수 있고, 또한 하나를 다른 것 아래에, 스크린의 중앙에, 왼쪽 중간 등에 배치할 수 있다. 엘리먼트는 주어진 순서대로 렌더링된다. 그러므로 만

약 첫 번째 엘리먼트가 스크린의 중앙에 있고, 이전 엘리먼트에 대해 그것들 스스로를 정렬하는 다른 엘리먼트들은 스크린 중앙에 대해 상대적으로 정렬될 것이다. 또한 이러한 순서 때문에 만약 이런 레이아웃을 지정하기 위해 XML을 사용한다면, 여러분이 (다른 뷰 오브젝트들을 배치하기 위해) 레퍼런스할 엘리먼트는 여러분이 다른 뷰에서 그것의 레퍼런스 ID로 그것을 참조하기 전에 XML 파일에서 나열되어야 한다.

아래의 예제는 XML 파일과 UI의 스크린 결과를 보여준다. (예를 들어 layout_toLeft 같이) 관련된 엘리먼트를 가리키는 애트리뷰트들은 관련된 리소스 구문(@id/id)을 사용해서 ID를 참조한다는 것에 주목하라.




```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:background="@drawable/blue"
                android:padding="10px" >
    <TextView android:id="@+id/label"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Type here:" />
    <EditText android:id="@+id/entry"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="@android:drawable/editbox_background"
            android:layout_below="@id/label" />
    <Button android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/entry"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10px"
```

```
            android:text="OK" />
    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@id/ok"
            android:layout_alignTop="@id/ok"
            android:text="Cancel" />
</RelativeLayout>
```

이런 속성들 중 일부는 엘리먼트에 의해 직접적으로 지원된다. 그리고 또 다른 일부는 그것의 LayoutParams 멤버(이 스크린에 있는 모든 엘리먼트는 RelativeLayout의 서브클래스이다. 왜냐하면 모든 엘리먼트들은 RelativeLayout 부모(parent) 오브젝트의 자식들(children)이기 때문이다)에 의해 지원된다. 정의되어 있는 RelativeLayout 파라미터들은 width, height, below, alignTop, toLeft, padding[Bottom|Left|Right|Top], 그리고 margin[Bottom|Left|Right|Top]이다. 이들 파라미터들 중 몇 가지는 명확하게 상대적인 레이아웃 위치들을 지원한다는 것에 유념하라 – 그것들의 값은 여러분이 이 뷰가 상대적으로 놓이길 원했던 엘리먼트의 ID이어야 한다. 예를 들어 TextView에 파라미터 toLeft="@id/my_button" 을 지정하는 것은 그 TextView를 my_button(이것은 XML 안에서 TextView 앞에 작성되어 있어야 한다)이라는 ID를 가진 뷰의 왼쪽에 놓고자 한다는 것이다.

이 샘플 코드를 보려면 Hello RelativeLayout 튜토리얼을 보아라.

주요(Important) 뷰 그룹 정리

이들 오브젝트들 모두는 자식child UI 엘리먼트를 보유하고 있다. 일부는 그들 자신의 시각적인 UI 형태를 제공한다. 반면에 나머지는 오직 그들의 자식child 뷰들의 레이아웃을 관리만하는 시각적이지 않은 구조들이다.

클래스	설 명
FrameLayout	단 하나의 오브젝트를 표시하기 위한 뷰 프레임으로서 역할을 하는 레이아웃
Gallery	하나로 묶여진 리스트에서, 이미지들의 수평적 스크롤링을 표시한다.
GridView	m 열과 n 행의 스크롤링 그리드(grid)를 표시한다.
LinearLayout	하나의 수평적 또는 수직적 행에 그것의 자식들(children)을 배치하는 레이아웃. 윈도우의 길이가 화면의 길이를 초과한다면 그것은 스크롤 바를 생성한다.

클래스	설 명
ListView	하나의 열로된 스크롤링 리스트를 표시한다.
RelativeLayout	여러분이 자식(child) 오브젝트의 위치를 서로간(자식(child) A는 자식(child) B의 왼쪽)에 또는 부모(parent)에 상대적(부모(parent)의 최상단에 정렬)으로 지정하는 것을 가능하게 한다.
ScrollView	엘리먼트의 열을 수직적으로 스크롤링 한다.
Spinner	한 줄짜리 텍스트 박스 내에, 하나로 묶여진 리스트에서 한번에 하나의 아이템을 보여준다. 수평 또는 수직으로 스크롤 할 수 있는 한 줄짜리 리스트 박스와 다소 비슷하다.
SurfaceView	그리기 전용 공간에 직접적인 접근을 제공한다. 그것은 서피스(surface) 위에 중층적인(layered) 자식(child) 뷰들을 보유할 수 있다. 그러나 그것은 위젯을 사용하는 것보다는 픽셀을 그릴 필요가 있는 애플리케이션을 위해 의도한 것이다.
TabHost	클릭을 모니터하는 탭 선택 리스트를 제공하며, 탭을 클릭할 때마다 애플리케이션이 스크린을 변경하는 것을 가능하게 한다.
TableLayout	임의의 개수의 행과 열을 가진 표 형식의 레이아웃, 각각의 셀은 여러분이 선택한 위젯을 보유한다. 행의 크기는 가장 큰 열에 맞게 재구성된다. 셀 경계선은 보이지 않는다.
ViewFlipper	한 줄짜리 텍스트박스에서 한번에 한 개의 아이템을 보여주는 리스트. 그것은 슬라이드 쇼처럼 시간 간격으로 아이템을 교체하도록 설정될 수 있다.
ViewSwitcher	ViewFlipper와 같다.

안드로이드가 뷰를 그리는 방법

액티비티가 포커스를 얻을 때, 그것은 자신의 레이아웃을 그리도록 요청될 것이다. 안드로이드 프레임워크는 그리기를 위한 절차를 처리할 것이다. 하지만 액티비티는 그것의 레이아웃 계층구조의 루트root 노드를 제공해야 한다.

그리기는 레이아웃의 루트root 노드를 가지고 시작된다. 그것은 레이아웃 트리를 측정하고 그리도록 요청된다. 그리기는 트리를 이동해 가면서 무효화invalid 영역과 만나는 각각의 뷰를 렌더링하는 것에 의해 처리된다. 순서대로, 각각의 뷰그룹은 그것의 자식들children 각각이 (draw() 메쏘드를 가지고) 그려지도록 요청하는 것에 대해 책임을 진다. 그리고 각각의 뷰는 그것 스스로를 그리는 것에 책임을 진다. 트리가 순서대로in-order 경유되기traversed 때문에, 이것이 의미하는 것은 부모parent가 그것의 자식들children 이전에 그려질 것이고, 형제sibling들은 트리에서 나타나는 순서대로 그려진다는 것이다.

프레임워크는 무효화 영역에 있지 않은 뷰는 그리지 않을 것이다. 그리고 또한 여러분을 위한 뷰 백그라운드(background)를 그리는 것을 처리할 것이다. 여러분은 invalidate()를 호출함으로써 뷰가 그려지도록 강제할 수 있다.

레이아웃을 그리는 것은, 측정 패스measure pass와 레이아웃 패스layout pass라는 두 번의 통과 과정pass process이다. 측정 패스는 measure(int,int)에서 구현되며, 뷰 트리를 위에서 아래로 경유traversal해 가면서 진행된다. 각 뷰는 재귀recursion되는 동안 크기 명세dimension specifications을 트리 아래로 보낸다. 측정 패스가 끝나면, 모든 뷰는 그것의 치수measurement(역주 : 뷰에서 필요한 크기)에 대한 저장을 마치게 된다. 두 번째 패스는 layout(int,int, int,int)에서 일어나며, 역시 위에서 아래로 진행된다. 이 패스 동안 각각의 부모parent는 측정 패스에서 계산된 크기를 사용해 그것의 자식들children 모두의 위치를 정하는 책임을 진다.

레이아웃을 인플레이트(inflate)하기 위해서는 requestLayout()을 호출하라. 일반적으로 이 메쏘드는 뷰가 더 이상 현재 경계 영역 내에 맞지 않다고 믿을 때 뷰 자신에 의해 스스로 호출된다.

뷰의 measure() 메쏘드가 리턴될 때, 그것의 getMeasuredWidth()와 getMeasuredHeight() 값은 그 뷰의 모든 자손descendant들의 값들과 함께 설정되어져야 한다. 뷰의 측정된 너비measured width와 측정된 높이measured height 값은 뷰의 부모parent에 의해 부과된 제약을 존중해야 한다. 측정 패스measure pass가 끝나면, 모든 부모parent는 그들 자식들children의 크기measurement 전부를 수용한다는 것을 보장한다. 부모parent 뷰는 그들의 자식들children에 대해서 한 번 이상 measure()를 호출할 수도 있다. 예를 들어 부모parent는 자식들children 얼마나 커질 수 있는지 찾기 위해 크기를 지정하지 않고(역주 : MeasureSpecs에 UNSPECIFIED 값을 설정함.) 각각의 자식child을 일단 측정할 수도 있다. 그런 다음, 만약 모든 자식들children의 자발적인 크기의 합이 너무 크거나 또는 너무 작다면 실질적인 숫자를 사용해서 그들에 대한 measure()를 다시 호출한다(즉, 만약 자식들(children)이 그들 각자가 얻을 공간이 얼마나 큰 지에 대해 그들끼리 동의하지 않는다면, 부모(parent)가 개입할 것이고 두 번째 패스에서 규칙을 정할 것이다).

측정 패스measure pass는 크기를 전달하기 위해 두 개의 클래스를 사용한다.

View.MeasureSpec 클래스는 뷰가 어떻게 측정되고 배치하길 원하는 지 그들의 부모에게 알리기 위해 사용된다. 베이스 LayoutParams 클래스는 단지 뷰가 너비와 높이 모두에 대해 얼마나 크게 되고 싶은 지를 기술할 뿐이다. 각각의 크기에 대해, 그것은 다음 중 하나를 지정할 수 있다.

- 정확한 숫자
- 뷰가 그 부모(패딩 제외)만큼 크고 싶어하는 것을 의미하는 FILL_PARENT
- 뷰가 단지 그것의 내용(패딩 추가)를 감쌀 만큼 충분히 크고 싶다는 것을 의미하는 WRAP_CONTENT

뷰그룹의 다른 서브클래스에 대한 LayoutParams의 서브클래스가 있다. 예를 들어 RelativeLayout은 그것 자신의 LayoutParams의 서브클래스를 가진다. 그것은 자식child 뷰를 수평 및 수직으로 가운데에 놓는 능력을 포함한다.

MeasureSpecs는 부모에서 자식child으로, 트리 아래로 요구조건을 보내기 위해 사용된다. MeasureSpec은 세 가지 모드 중 하나가 될 수 있다:

- UNSPECIFIED : 이것은 자식child 뷰가 원하는 크기를 결정하기 위해 부모parent에 의해 사용된다. 예를 들어 LinearLayout은 240 픽셀의 너비가 주어진 자식child 뷰가 원하는 높이가 얼마인지 알기 위해, UNSPECIFIED로 설정된 높이와 EXACTLY 240으로 설정된 너비를 사용해서 그것의 자식child 뷰에 대한 measure()를 호출할 수 있다.
- EXACTLY : 이것은 자식child에게 정확한 크기를 부과하기 위하여 부모parent에 의해서 사용된다. 자식child은 이 크기를 사용해야 하며, 그것의 모든 자손들decendants은 이 크기 안에서 맞춰져야 하는 것을 보장해야 한다.
- AT_MOST: 이것은 자식child에게 최대 크기를 부과하기 위해 부모parent에 의해서 이용된다. 자식child은 그것과 그것의 자손들descendants 모두가 이 크기 안에서 맞춰질 것이라는 것을 보장해야 한다.

Section | 4

리소스와 에셋(Asset)

4.1. 리소스와 국제화

4.2. 사용가능한 리소스 타입

4. 리소스와 에셋(Asset)

리소스는 안드로이드 애플리케이션의 필수불가결한 부분이다. 일반적으로. 이것들은 여러분의 애플리케이션에서 포함시켜서 레퍼런스하고자 하는 이미지, 오디오, 비디오, 텍스트 문자열, 레이아웃, 테마 등과 같은 외부 요소들이다. 모든 안드로이드 애플리케이션은 리소스를 위한 디렉토리(res/)와 에셋을 위한 디렉토리(assets/)를 가진다. 에셋은 그것을 가지는 애플리케이션이 매우 적기 때문에 드물게 사용된다. 여러분이 원시 바이트bytes를 읽을 필요가 있을 때, 여러분은 에셋으로써 데이터를 저장할 필요가 있을 뿐이다. 리소스와 에셋에 대한 디렉토리 모두 여러분의 프로젝트 디렉토리의 최상위에, 여러분의 소스 코드 디렉토리와 나란히 존재한다.

"리소스resources"와 "에셋assets" 의 차이는 외관 상 많지 않다. 그러나 일반적으로 여러분은 외부 컨텐트를 저장하기 위해 에셋을 사용하기 보다는 리소스를 더 자주 사용할 것이다. 실제 차이는, 리소스 디렉토리에 놓인 어떤 것은 안드로이드에 의해 컴파일된 R 클래스를 통해, 애플리케이션에 의해 쉽게 접근 가능하다는 것이다. 에셋 디렉토리 안에 있는 모든 것은 원시 파일 포맷으로 유지될 것이며, 여러분은 그것을 읽기 위해 바이트 스트림으로 파일을 읽기 위한 에셋매니저AssetManager를 사용해야 한다. 그러므로 파일과 데이터를 리소스(res/)에 보관하는 것은 그것에 대한 접근을 용이하게 만든다.

이 토픽의 문서에서, 여러분은 안드로이드 애플리케이션에서 일반적으로 사용되는 표준 리소스의 종류와 여러분의 코드에서 그것들을 레퍼런스하는 방법에 관한 정보를 찾을 수 있을 것이다. "리소스와 국제화Internationalization"는, 안드로이드가 프로젝트 리소스를 이용하는 방법에 대해 더 많이 공부하기 위한 시작점이다. 그런 다음에 "사용가능한 리소스 타입" 문서는 다양한 리소스 타입의 개요와 그것들의 스펙에 대한 레퍼런스를 제공한다.

리소스와 국제화

리소스는 외부 파일(즉, 비코드(non-code) 파일)이다. 그것은 여러분의 코드에서 사용되며, 빌드 시점에 여러분의 애플리케이션안으로 컴파일된다. 안드로이드는 XML, PNG, 그리고 JPEG 파일들을 포함하는 많은 다른 종류의 리소스 파일을 지원한다. XML 파일들은 그것들이 설명하고자 하는 것에 따라 매우 다른 포맷들을 가진다. 이 문서는 지원되는 파일의 종류가 무엇인지, 그리고 각각의 구문과 포맷을 설명한다.

리소스는 소스 코드 외부에 존재하며, 그리고 그 안의 XML 파일은 효율성을 위해 빠른 로딩loading 포맷인 바이너리로 컴파일된다. 문자열 역시 마찬가지로 효율적인 저장 형태로 압축된다. 이것이 우리가 안드로이드 플랫폼 내에 이와 같이 다른 타입의 리소스를 가지는 이유이다.

이것은 상당히 기술적으로 빽빽한 문서이며, "사용가능한 리소스 타입 문서"와 함께 리소스에 대한 많은 정보를 다룬다. 안드로이드를 사용하기 위해 이 문서를 암기해서 알고 있을 필요는 없으나, 여러분이 그것을 필요로 할 때 그 정보가 여기에 있다는 것을 알아야 한다.

소개(Introduction)

이 토픽은 리소스와 관련된 용어 리스트와, 그리고 코드에서 리소스를 사용하는 일련의 예제들을 포함한다. 지원되는 안드로이드 리소스 타입에 대한 완전한 가이드를 위해서는, 이후에 나오는 사용가능한 리소스를 보라.

안드로이드 리소스 시스템은 애플리케이션과 관련된 모든 비코드non-code 에셋asset 정보를 유지한다. 여러분의 애플리케이션의 리소스에 접근하기 위해, 여러분은 리소스Resources 클래스를 사용한다. 여러분의 애플리케이션과 관련된 리소스 인스턴스는 Context.getResources()를 통해 일반적으로 찾아내어 진다.

애플리케이션의 리소스는 빌드 시점에 빌드 시스템에 의해 여러분을 위해 애플리케이션 바이너리로 컴파일된다. 여러분은 리소스를 사용하기 위해 소스 트리에 정확하게 그것을 설치하고 애플리케이션을 빌드해야 한다. 빌드 과정의 일부로써 여

러분이 소스코드에서 사용할 수 있는 각각의 리소스에 대한 심볼들이 만들어진다. 이것은 여러분의 애플리케이션 코드가 여러분이 정의한 리소스와 일치하는 것을 컴파일러로 하여금 검증하는verify 것을 가능하게 한다.

이 섹션의 나머지는 애플리케이션에서 리소스를 사용하는 방법에 관한 튜토리얼로 구성되어 진다.

리소스 생성(Creating Resources)

안드로이드는 문자열, 비트맵 그리고 많은 다른 타입의 리소스를 지원한다. 각각의 구문과 포맷, 그리고 그것들이 저장되어져 있는 곳은 오브젝트의 타입에 의존한다. 그럼에도 불구하고, 일반적으로 여러분은 세 가지 타입의 파일로부터 리소스를 만든다. XML 파일(비트맵과 원시 파일이 아닌 모든 것), 비트맵 파일(이미지를 위한 것), 그리고 원시raw 파일(그 밖의 것들로, 예를 들어 사운드 파일과 기타)이 그것이다. 실제로는 두 가지의 다른 타입의 XML이 또한 존재한다. 현재 그대로 패키지로 컴파일되는 것과, 그리고 aapt에 의해 리소스를 생성하기 위해 사용되는 것이 있다. 여기 각 리소스 타입의 리스트, 파일의 포맷, 파일의 설명, 그리고 임의의 XML 파일에 대한 상세 사항이 있다.

여러분은 프로젝트에 있는 res/ 디렉토리 아래에 있는 적절한 하위 디렉토리 아래에 여러분의 리소스 파일을 만들고 저장할 것이다. 안드로이드는 리소스들이 위치하는 하위 폴더와 파일의 포맷에 따라 그것들을 컴파일하는 리소스 컴파일러(aapt)를 가지고 있다. 테이블 1은 각각의 리소스에 대한 파일 타입 리스트를 보여준다. 각각의 오브젝트 타입, 구문, 그리고 포맷 또는 포함하고 있는 파일의 구문에 대한 설명에 대해서는 이후에 나오는 "사용가능한 리소스 타입"을 보라.

리소스는 최종적인 APK 파일내로 컴파일된다. 안드로이드는 여러분이 코드에서 이 리소스들을 레퍼런스하기 위해 사용할 수 있는 R이라 불리는 래퍼wrapper 클래스를 생성한다. R은 소스 파일의 경로와 이름에 따라 이름지어진 서브클래스를 포함한다.

테이블 1.

디렉토리	리소스 타입들
res/anim/	프레임 애니메이션의 프레임 또는 트윈드(tweened) 애니메이션 오브젝트로 컴파일되는 XML 파일들.
res/drawable/	다음과 같이 드로어블(Drawable) 리소스 서브타입으로 컴파일되는 .png, .9.png, jpg 파일들 : • 비트맵 (bitmap) 파일 • 9-패치(patches) (크기 변경가능한 비트맵) 이 타입의 리소스를 얻기 위해서는, mContext.Resource.getDrawable(R.drawable.imageId)를 사용하라. 주의 : 여기에 위치한 이미지 리소스는 aapt 도구에 의해 무손실 이미지 압축을 사용해서 자동으로 최적화된다. 예를 들어 256 컬러 이상이 요구되지 않는 트루컬러 PNG는 컬러 팔레트를 가진 8-비트(bit) PNG로 변환된다. 이것은 동일 품질의 이미지를 만들어낼 것이지만, 더 적은 메모리를 요구한다. 그러므로 이 디렉토리에 위치하는 이미지 바이너리는 빌드하는 동안 변경될 수 있다는 것에 주의하라. 만약 여러분이 비트맵으로 변환하기 위해 비트(bit) 스트림으로 그 이미지를 읽어들일 계획이라면, 대신에 여러분의 이미지를 res/raw/ 폴더에 넣어라. 그 폴더에 있는 이미지들은 최적화되지 않을 것이다.
res/layout/	스크린 레이아웃 (또는 스크린의 일부)로 컴파일되는 XML 파일들. 3장. 사용자 인터페이스의 레이아웃 선언하기를 보라.
res/values/	많은 종류의 리소스로 컴파일되는 XML 파일들. 주의: 다른 res/ 폴더들과 다르게, 이것은 리소스 그 자체보다는 생성할 리소스에 대한 설명을 포함하는 어떤 갯수의 파일들도 보유할 수 있다. XML 엘리먼트의 타입들은, 이 리소스의 R 클래스 내에서의 위치를 제어한다. 파일들은 무엇이든 이름 붙여질 수 있지만, 이 폴더에는 전형적인 파일들이 있다(그 규약은 이후 그 폴더 안에 정의될 엘리먼트 타입의 이름을 따서 그 파일의 이름을 명명한다). • array.xml 배열을 정의함. • colors.xml 컬러 드로어블(drawable)과 컬러 문자열 값을 정의함. 이 리소스를 얻기 위해서는 각각에 대응되는 Resources.getDrawable()와 Resources.getColor()를 사용하라. • dimens.xml 크기(dimension) 값을 정의함. 이 리소스을 얻기 위해 Resources.getDimension()을 사용하라. • strings.xml 문자열 값을 정의함(이 리소스를 얻기 위해서는 Resources.getString() 또는 보다 좋은 Resources.getText()를 사용하라. getText()는 UI 문자열을 위해 통상 바람직한 임의의 리치(rich) 텍스트 스타일을 보유할 것이다). • styles.xml 스타일 오브젝트를 정의함.
res/xml/	컴파일되어서, Resources.getXML()을 호출함으로써 런타임에서 읽어들여 질 수 있는 임의의 XML 파일들.
res/raw/	디바이스에 직접적으로 복사되는 임의의 파일들. 그것은 여러분의 애플리케이션 빌드과정이 만들어내는 압축파일로 컴파일되지 않고 추가된다. 여러분의 애플리케이션에서 이 리소스를 사용하기 위해서는 R.raw.somefilename 형태의 리소스 ID를 사용하여 Resources.openRawResource()를 호출하라.

글로벌 리소스 노트(Global Resource Notes)

- 몇몇 리소스들은 여러분으로 하여금 컬러를 정의할 수 있게 한다. 안드로이드는 다양한 웹 스타일 포맷들로 작성되는 컬러 값을 수용한다 – 다음과 같은 임의의 형식의 16진수 상수: #RGB, #ARGB, #RRGGBB, #AARRGGBB가 그것이다.
- 모든 컬러 값은 알파채널 값 설정을 지원하며, 컬러 값의 첫번째 2개의 16진수는 투명도를 나타낸다. 알파 채널이 0인 것은 투명을 의미한다. 디폴트 값은 불투명이다.

리소스 사용(Using Resources)

이 섹션은 여러분이 만들었던 리소스를 사용하는 방법을 설명한다. 그것은 다음과 같은 토픽을 포함한다:

- 코드 안에서 리소스 사용하기 – 여러분의 코드에서 리소스를 인스턴스화하기 위해 그것을 호출하는 방법.
- 다른 리소스에서 리소스 레퍼런스 하기 – 여러분은 다른 리소스에서 리소스를 레퍼런스할 수 있다. 이것은 여러분으로 하여금 리소스에서 일반적인 리소스 값을 재사용하는 것을 가능하게 한다.
- 택일적alternate 설정을 위한 택일적 리소스 지원하기 – 여러분은 호스트 하드웨어의 언어 또는 디스플레이 설정에 의존해서 다른 리소스가 로드load되도록 지정할 수 있다.

컴파일 시점에, 안드로이드는 여러분의 프로그램에 있는 모든 리소스에 대한 리소스 식별자identifier를 포함하는 R 이라는 이름의 클래스를 생성한다. 이 클래스는 안드로이드가 지원하는 리소스의 각 타입에 따른 하나씩의 서브클래스를 포함하며, 여러분은 각 서브클래스를 위해 하나의 리소스 파일을 제공했었다. 각각의 클래스는 컴파일된 리소스에 대해 하나 또는 그 이상의 식별자identifier를 포함하며, 이것은 여러분이 코드에 그 리소스를 로드하기 위해 사용된다. 아래에 문자열, 레이아웃(스크린 혹은 스크린의 부분), 그리고 이미지 리소스를 포함하는 작은 리소스 파일이 있다.

NOTE R 클래스는 자동으로 생성되는 파일이며, 손으로 편집하기 위해 설계된 것이 아니다. 그것은 리소스가 업데이트됐을 때, 필요에 따라서 자동으로 새로 만들어질 것이다.

```
package com.android.samples;
public final class R {
    public static final class string {
        public static final int greeting=0x0204000e;
        public static final int start_button_text=0x02040001;
        public static final int submit_button_text=0x02040008;
        public static final int main_screen_title=0x0204000a;
    };
    public static final class layout {
        public static final int start_screen=0x02070000;
        public static final int new_user_pane=0x02070001;
        public static final int select_user_list=0x02070002;

    };
    public static final class drawable {
        public static final int company_logo=0x02020005;
        public static final int smiling_cat=0x02020006;
        public static final int yellow_fade_background=0x02020007;
        public static final int stretch_button_1=0x02020008;

    };
};
```

코드에서 리소스 사용(Using Resources in Code)

코드에서 리소스를 사용하는 것은, 단지 정확한 리소스 ID와 여러분의 리소스가 컴파일된 오브젝트의 타입을 아는 문제이다. 여기에 리소스를 레퍼런스하는 것에 대한 구문이 있다.

```
R.resource_type.resource_name
```

또는

```
android.R.resource_type.resource_name
```

위의 구문에서 resource_type은 리소스의 특정 타입을 보유하는 R의 서브클래스이다. resource_name는 XML 파일에서 정의된 리소스에 대한 name 애트리뷰트이거나, 또는 다른 파일 타입에 의해 정의된 리소스에 대한 파일 이름(확장

자 없이) 이다. 리소스의 각 타입은 리소스의 타입에 따라서 R의 특정 서브클래스에 추가될 것이다. R의 어떤 서브클래스가 여러분의 컴파일된 리소스 타입을 보유하는가를 배우기 위해서는, 사용가능한 리소스 문서를 참조하라. 여러분 자신의 애플리케이션에 의해 컴파일된 리소스는 패키지 이름 없이(단순히 R.resource_type.resource_name으로) 레퍼런스될 수 있다. 안드로이드는 스크린 스타일과 버튼 백그라운드와 같은, 다수의 표준 리소스를 포함한다. 코드에서 이것을 레퍼런스하기 위해서, 여러분은 android.R.drawable.button_background에서와 같이 그것이 android를 가지도록 제약해야 한다.

여기에 코드에서 컴파일된 리소스를 사용하는 몇 가지의 좋고 나쁜 예제가 있다.

```
// Load a background for the current screen
// from a drawable resource.this.getWindow().
// setBackgroundDrawableResource(R.drawable.my_background_image);
// WRONG Sending a string resource reference into a
// method that expects a string.
this.getWindow().setTitle(R.string.main_title);

// RIGHT Need to get the title from the Resources wrapper.
this.getWindow().setTitle(Resources.getText(R.string.main_title));
// Load a custom layout for the current screen.
setContentView(R.layout.main_screen);

// Set a slide in animation for a ViewFlipper object.
mFlipper.setInAnimation(AnimationUtils.loadAnimation(this,
        R.anim.hyperspace_in));

// Set the text on a TextView object.
TextView msgTextView = (TextView)findViewById(R.id.msg);
msgTextView.setText(R.string.hello_message);
```

리소스에 레퍼런스 하기(References to Resources)

애트리뷰트 (또는 리소스)로 제공되는 값 또한 리소스로 레퍼런스될 수 있다. 어떤 레퍼런스는 컬러와 정수integer를 포함하는 어떤 리소스 타입이 될 수도 있지만, 이것은 흔히 레이아웃 파일 안에 문자열(문자열이 로컬라이즈 될 수 있도록 하기 위해)과 (다른 파일에 존재하는) 이미지를 공급하기 위해 사용된다.

예를 들어 만약 우리가 컬러 리소스를 가지고 있다면, 우리는 레이아웃 파일의 텍

스트 컬러가 그러한 리소스 중 하나의 값으로 설정되도록 작성할 수 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<EditText id="text"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="Hello, World!" />
```

여기에서 리소스 레퍼런스를 나타내기 위해 "@" 접두어를 사용하는 것에 주의하라 – 그 뒤에 붙어있는 텍스트는 @[package:]type/name의 형식의 리소스 이름이다. 위의 경우에서 우리는 패키지를 지정할 필요가 없다. 왜냐하면 리소스가 우리 자신의 패키지에 있는 리소스를 레퍼런스하기 때문이다. 시스템 리소스를 레퍼런스하기 위해서는 패키지 이름을 써야 할 필요가 있을 것이다.

```
<EditText id="text"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@android:color/opaque_red"
    android:text="Hello, World!" />
```

다른 예제로, 여러분은 레이아웃 파일에 문자열을 공급할 때 그 문자열이 로컬라이즈될localized 수 있도록 항상 리소스 레퍼런스를 사용해야 한다.

```
<?xml version="1.0" encoding="utf-8"?>
<EditText id="text"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@android:color/opaque_red"
    android:text="@string/hello_world" />
```

이 설비facility는 또한 리소스들간의 레퍼런스를 생성하기 위해 이용될 수도 있다. 예를 들어 우리는 이미 존재하는 이미지에 대한 앨리어스alias인 새로운 드로어블drawable 리소스를 생성할 수 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <drawable id="my_background">@android:drawable/theme2_background
      </drawable>
</resources>
```

테마 애트리뷰트 레퍼런스 하기(References to Theme Attribute)

또 다른 종류의 리소스 값은 여러분으로 하여금 현재 테마의 애트리뷰트의 값을 레퍼런스하는 것을 허용한다. 이 애트리뷰트 레퍼런스는 스타일 리소스와 XML 애트리뷰트에서만 사용될 수 있다. 그것은 여러분이 구체적인 값을 제공하는 대신, 현재 테마에 의해 공급되는 표준 변이variation들로 그것을 변경함으로써 UI 엘리먼트의 모습를 커스터마이즈하는 것을 허용한다.

하나의 예제로, 우리의 레이아웃 안에 기본적인 시스템 테마에서 정의된 표준 컬러중 하나로 텍스트 컬러를 지정하기 위해, 우리는 테마 애트리뷰트를 사용할 수 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<EditText id="text"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="?android:textDisabledColor"
    android:text="@string/hello_world" />
```

이것은 '@' 대신 "?" 접두어를 우리가 사용한다는 것을 제외하고는 리소스 레퍼런스와 매우 유사하는 것에 주의하라. 여러분이 이 마크업markup을 사용할 때, 여러분은 테마에서 찾을 애트리뷰트 리소스의 name을 공급한다 – 왜냐하면 리소스 도구가 예상되는 애트리뷰트 리소스를 알기 때문에, 여러분은 명시적으로 그 타입(?android:attr/android:textDisabledColor가 되도록)을 기술할 필요는 없다.

자체raw 리소스 대신, 테마에서 값을 찾기 위해 이 리소스 식별자Identifier를 사용한다는 것이외에, 그 name 구문은 "@" 포맷과 동일하다: – 여기에서 ?[namespace:]type/name이 type을 가지는 것은 선택 사항이다.

시스템 리소스 사용하기(Using System Resources)

시스템에 포함된 많은 리소스는 애플리케이션에서도 사용가능하다. 그런 리소스 전부는 "android.R" 클래스 안에 정의되어 있다. 예를 들어 여러분은 아래의 코드를 사용해서 스크린에 표준 애플리케이션 아이콘을 표시할 수 있다.

(역주: 아래의 코드 중 requestScreenFeatures()와 setBadgeResource()는 현재 사용되는 않는 메쏘드임.)

```
 // feature was removed from M5
public class MyActivity extends Activity
{
    public void onStart()
    {
        requestScreenFeatures(FEATURE_BADGE_IMAGE);
        super.onStart();
        setBadgeResource(android.R.drawable.sym_def_app_icon);
    }
}
```

유사한 방법으로, 아래의 코드는 시스템에 의해 정의된 표준 "초록색(green) 백그라운드" 시각적 효과를 여러분의 스크린에 적용할 것이다.

(역주: 아래의 테마는 green 테마가 아니라 black 테마임)

```
public class MyActivity extends Activity
{
    public void onStart()
    {
        super.onStart();
        setTheme(android.R.style.Theme_Black);
    }
}
```

택일적(Alternate) 리소스 택일하기

여러분은 디바이스 상의 UI 언어 또는 하드웨어 설정에 의존해서, 여러분의 애플리케이션이 사용할 서로 다른 리소스를 제공할 수 있다. 여러분이 다른 문자열들, 레이아웃들 그리고 그 밖의 리소스들을 포함할 수 있을지라도, SDK는 어떤 택일적alternate 리소스가 로드되어야 하는가를 지정하는 메쏘드를 여러분에게 제시하지 않는다는 것에 주의하라. 안드로이드는 그 하드웨어와 위치location에 알맞은 집

합을 찾아내어 적절하게 그것을 로드한다. 사용자는 디바이스의 설정setting 패널을 사용하여 택일적alternate 언어 설정을 선택할 수 있다.

택일적alternate 리소스를 포함하기 위해서는, 폴더 이름에 그것에 적용되는 설정들(언어, 스크린 오리엔테이션 등)을 가리키는 수식어qualifiers가 덧붙여지도록 병렬적 리소스 폴더를 생성하라. 예를 들어 여기에 영어를 위한 한 개의 문자열string 리소스 파일과 프랑스어를 위한 또 다른 한 개의 문자열 리소스 파일을 가지는 프로젝트가 있다.

```
MyApp/
    res/
        values-en/
            strings.xml
        values-fr/
            strings.xml
```

안드로이드는 각각에 대해 다양한 값을 가지는 여러 타입의 수식어qualifiers를 지원한다. 리소스 폴더 이름의 끝에, 이것들을 대쉬(–)로 구분되어 지게 덧붙여라. 여러분은 각각의 폴더 이름에 다양한 수식어qualifiers를 추가할 수 있으나, 여기에 나열된 순서대로 나타내야 한다. 예를 들어 모든 설정을 명시하는 드로어블drawable 리소스를 포함하는 폴더는 이렇게 보일 것이다.

```
MyApp/
    res/
        drawable-en-rUS-large-long-port-mdpi-finger
        -keysexposed-qwerty-dpad-480x320/
```

보다 일반적으로, 여러분은 몇몇 특정 설정 옵션만을 지정할 것이다. 여러분은 전체 리스트에서 임의의 값들을 누락시킬 수 있다. 단, 사용된remaining 값들이 여전히 동일한 순서를 있을 때에만 가능하다.

```
MyApp/
    res/
        drawable-en-rUS-finger/
        drawable-port/
        drawable-port-mdpi/
        drawable-qwerty/
```

테이블 2는 유효한 폴더 이름 수식어qualifier를 서열 순서로 나열한다. 안드로이드가 가장 일치하는 디렉토리를 찾는 방법에서 설명되듯이, 테이블에서 더 위쪽에 리스트된 수식어는 더 아래쪽에 리스트된 것보다 앞쪽에 놓인다.

테이블 2

수식어(Qualifier)	값
MCC 와 MNC	디바이스의 SIM에 있는 모바일 국가 코드(MCC, Mobile Country Code)와 모바일 네트워크 코드(MNC, Mobile Network Code). 예를 들어 mcc3130-mnc004 (미국, 버라이존 브랜드); mcc208-mnc00 (프랑스, 오렌지 브랜드); mcc234-mnc00 (영국, BT 브랜드); 만약 그 디바이스가 라디오 커넥션 (GSM 폰)을 사용한다면, 그 MCC는 SIM에서 올 것이고, 그 MNC는 디바이스가 속한 네트워크에서 올 것이다. 여러분은 예를 들어 여러분의 애플리케이션에 국가에 한정된 법적 리소스를 포함하기 위해 가끔 MCC만을 사용할 지도 모른다. 만약 여러분의 애플리케이션이 MCC/MNC 조합에 대한 리소스들을 지정한다면, 그 리소스들은 MCC와 MNC 모두가 일치하는 경우에만 사용될 수 있을 뿐이다.
언어와 지역	ISO 639-1 언어 코드 두자와 ISO 3166-1-alpha-2 지역 코드 두자(앞에 소문자 "r"이 있음). 예를 들어 en-rUS, fr-rFR, es-rES. 코드는 대소문자 구분이 있다: 언어 코드는 소문자이며, 국가 코드는 대문자이다. 여러분은 지역을 단독으로 지정할 수 없으나, 언어는 단독으로 지정할 수 있다. 예를 들어 en, fr, es..
스크린 크기	small(작음), normal(보통), large(큼) 해당 리소스가 특별한 종류의 스크린을 위한 것임을 지정한다. 이것들이 의미하는 것은 다음과 같다. • **Normal 스크린**은 전통적인 안드로이드 HVGA 중간 밀도 스크린에 기반한다. 만약 스크린이 적어도 이 크기(밀도 독립적)이고 큰 것(large)이 아니라면, normal인 것으로 간주된다. 그러한 스크린의 예로는, WQVGA 낮은 밀도, HVGA 중간 밀도, WVGA 높은 밀도가 있다. • **Small 스크린**은 QVGA 낮은 밀도 스크린상의 가용 공간에 기반한다. 세로 방향(portrait) HVGA 디스플레이를 고려할 때, 그것은 동일한 가용 폭을 갖지만 높이는 더 작다 -- HVAG의 가로 세로 비율이 2:3인 것 대비 이것은 3:4이다. 예로는 QVGA 낮은 밀도와 VGA 높은 밀도가 있다. • **Large 스크린**은 VGA 중간 밀도 스크린상의 가용 공간에 기반한다. 이런 스크린은 HVGA 디스플레이보다 너비와 높이 모든 면에서 현저하게 가용 공간을 더 가진다. 예로는 VGA와 WVGA 중간 밀도 스크린이 있다.
더 넓은/더 긴 스크린	long, notlong 해당 리소스가 전통적인 스크린 보다 더 길고/더 넓은 것을 위한 것임을 지정한다. 이것은 순수하게 스크린의 외관비율에 기반한다: QVGA, HVGA, 그리고 VGA는 notlong이며, WQVGA, WVGA, FWVGA는 long이다. long이 현재의 방향(orientation)에 의존해서 넓거나 긴 것을 의미한다는 것에 주의하라.
스크린 오리엔테이션	port(세로), land(가로),square(정사각형) 해당 리소스가 세로 또는 가로 스크린을 위한 것임을 지정한다; square는 현재 사용되지 않는다.

스크린 픽셀 밀도	ldpi, mdpi, hdpi, nodpi 해당 리소스가 목적으로 하는 스크린 밀도를 지정한다. 전통적인 HVGA 스크린의 중간 밀도(mdpi)는 대략 160dpi로 정의된다; 낮은 밀도(ldpi)는 120이고, 높은 밀도(hdpi)는 240이다. 따라서 각각의 밀도간의 크기계수(scaling factor)는 4:3이다. 그러므로 ldpi에서 9x9 비트맵은 mdpi에서는 12x12가 될 것이고, hdpi에서는 16x16이 될 것이다. 특별히 nodpi 밀도는 비트맵 리소스들이 로드 시점에 디바이스의 밀도에 일치하도록 크기가 조정되는 것을 방지하기 위해 사용될 수 있다. 안드로이드가 사용할 리소스를 선택할 때, 안드로이드는 스크린 밀도를 그 밖의 수식어들과는 다르게 취급한다. (이후에 나오는) "안드로이드가 가장 일치되는 디렉토리를 찾는 방법"의 단계 1에서, 스크린 밀도는 항상 일치하는 것으로 간주된다. 단계 4에서 고려되는 해당 수식어가 만약 스크린 밀도라면, 안드로이드는 그 시점에서 단계 5로 이동하지 않고 최종적으로 가장 일치되는 것을 선택할 것이다. 여러분은 또한 92dpi 또는 108dpi와 같이 명시적인 밀도들을 지정할 수 있다. 그러나 이것은 시스템에서 완전하게 지원되지 않기 때문에 사용되어서는 안된다.
터치스크린 타입	notouch(터치 없음), stylus(스타일러스), finger(손가락)
키보드가 사용자에게 제공되는지 여부	keysexposed(키보드 사용가능), keyshidden(키보드 없음), keyssoft(소프트 키보드 사용) 만약 여러분의 애플리케이션이 오직 소프트 키보드를 가지고 사용되어야 하는 특정 리소스를 가진다면, keyssoft 값을 사용하라. 만약 사용가능한 keyssoft 리소스가 없고 (오직 keysexposed와 keyshidden 만 있음) 그 디바이스가 소프트 키보드를 보여준다면, 그 시스템은 keysexposed 리소스를 사용할 것이다.
기본적인 텍스트 입력방법	nokeys(키 없음), qwerty, 12key
네비게이션 키가 사용자에게 제공되는지 여부	navexposed(네비게이션 키 사용가능), navhidden(네비게이션 키 없음) 만약 사용자가 하드웨어 네비게이션 키를 현재 사용할 수 있다면, navexposed 리소스가 사용될 것이다; 만약 (덮개이 닫힌 것처럼) 그것을 사용할 수 없다면, navhidden이 사용될 것이다.
기본적인 터치스크린 이외의 네비게이션 방법	nonav(네비케이션 없음), dpad(방향키), trackball(트랙볼),이 wheel(휠)
스크린 해상도	320x240, 640x480, 기타. 더 큰 수치값이 앞쪽에 지정되어야 한다. 이 설정은 경시되고(deprecated) 있다. 그러므로 사용되어서는 안된다; 대신에 위에서 언급된 스크린 밀도, 더 넓고/더 긴 스크린, 그리고 스크린 방향을 사용하라.
SDK 버전	디바이스에 의해 지원되는 SDK 버전, 예를 들어 v3. 안드로이드 1.0 SDK는 v1, 1.1 SDK는 v2, 그리고 1.5 SDK는 v3이다.
(마이너,Minor 버전)	(여러분은 현재 마이너 버전을 지정할 수 없다. 그것은 항상 0으로 설정된다.)

이 리스트는 통신사, 브랜딩, 디바이스/하드웨어, 또는 제조사 같은 디바이스 제한적 파라미터를 포함하지 않는다. 애플리케이션이 자신이 실행되는 디바이스에 대해 알아야 할 필요가 있는 모든 것은, 위의 테이블에 있는 리소스 수식어qualifiers를 통해 인코드encode된다.

수식어가 붙거나 붙지 않은 모든 리소스 디렉토리는 res/ 폴더 아래에 있다. 여기에 수식어가 붙은 리소스 디렉토리 이름에 대한 약간의 가이드라인이 있다.

- 여러분은 대쉬(-)에 의해 분리되는 여러 개의 수식어를 지정할 수 있다. 예를 들어 drawable-en-rUS-land는 가로방향 오리엔테이션의 미국(US)-영어(English) 디바이스에 적용될 것이다.
- 수식어는 위의 테이블 2에서 나열된 순서로 되어야 한다. 예를 들어 다음과 같다.
  - 옳음 : values-mcc460-nokeys/
  - 틀림 : values-nokeys-mcc460/
- 값들은 대소문자를 구분하지 않는다. 리소스 컴파일러는 대소문자를 구분하는 파일 시스템에서의 문제를 피하기 위해서 포로세싱 전에에 폴더 이름을 소문자로 변환한다. 대소문자를 구분하는 파일 시스템에서 여러분은 모든 이름을 소문자로 유지하거나, 나중에 리소스 파일을 찾고자 할 때 여러분이 합리적으로 할 수 있도록 적어도 일관된 방식을 사용해야 한다.
- 각각의 수식어 타입에 대해서는 오직 하나의 값만이 지원된다. 예를 들어 만약 여러분이 스페인과 프랑스을 위해 정확하게 동일한 드로어블drawable 파일을 사용하고자 한다면, 여러분은 똑같은 파일을 포함하고 있는 drawable-rES/ 와 drawable-rFR/ 과 같이 두 개의 리소스 디렉토리를 필요로 할 것이다. 여러분은 drawable-rES-rFR/ 로 명명된 디렉토리를 가질 수 없다.
- 수식어 디렉토리는 다른 수식어 디렉토리 안에 있을 수 없다cannot be nested. 예를 들어 여러분은 res/drawable/drawable-en 을 가질 수 없다.

코드에서 리소스가 레퍼런스되는 방법

모든 리소스는 코드 또는 리소스 레퍼런스 구문에서 간단하고, 꾸밈없는undecorated 이름으로 레퍼런스될 것이다. 그러므로 만약 리소스가 다음과 같이 명명되었다면,

```
MyApp/res/drawable-port-mdpi/myimage.png
```

그것은 아래와 같이 레퍼런스될 것이다.

```
R.drawable.myimage (code)
@drawable/myimage (XML)
```

만약 여러 개의 드로어블drawable 디렉토리가 있다면 (위에서 언급되었듯이) 안드로이드는 그것 중 하나를 선택할 것이고, 그것에서 myimage.png를 로드할 것이다.

안드로이드가 가장 일치되는 디렉토리를 찾는 방법

안드로이드는 디바이스의 현재 설정에 의존해서 런타임에서 사용되어야 하는 리소스를 아래에 놓인 다양한 리소스 파일 중에서 골라낼 것이다. 여기에서 사용되는 예제는 다음과 같은 디바이스 설정을 가정한다.

```
Locale = en-GB
Screen orientation = port
Screen pixel density = mdpi
Touchscreen type = notouch
Primary text input method = 12key
```

여기에 안드로이드가 선택하는 방법이 있다.

1. 현재 디바이스 설정에 일치하지 않는 임의의 리소스를 제거한다. 예를 들어 드로어블drawable에 대한 다음과 같은 리소스 디렉토리가 있다고 가정하자. drawable-fr-rCA/ 디렉토리는 제거될 것이다. 왜냐하면 그것은 디바이스의 로케일locale과 일치하지 않기 때문이다.

   MyApp/res/drawable/
   MyApp/res/drawable-en/
   ~~MyApp/res/drawable-fr-rCA/~~
   MyApp/res/drawable-en-port/
   MyApp/res/drawable-en-notouch-12key/
   MyApp/res/drawable-port-ldpi/
   MyApp/res/drawable-port-notouch-12key

   예외 : 스크린 픽셀 밀도는 파일을 제거하기 위해 사용되지 않는 유일한 수식어이다. 디바이스의 스크린 픽셀 밀도가 중간medium dpi임에도 불구하고, drawable-port-ldpi/ 는 그 리스트에서 제거되지 않는다. 왜냐하면 모든 스크린 밀도가 이 시점에서 일치하는 것으로 간주되기 때문이다.

2. 테이블 2로부터, 그 리스트에 있는 가장 높은 순위의 수식어를 골라낸다(MCC로 시작해서, 그 리스트의 아래로 이동한다).

3. 사용가능한 리소스 디렉토리 중 어떤 것이 이 수식어를 포함하는가?

   - 만약 "아니오"라면, 단계 2로 돌아가서 테이블 2에 열거된 다음 수식어를 찾는다. 우리들의 예제에서 우리가 언어에 도달할 때까지 그 답은 "아니오"이다.
   - 만약 "예"라면, 단계 4로 이동한다.

4. 이 수식어를 포함하지 않는 리소스 디렉토리를 제거한다. 우리들의 예제에서, 우리는 언어 수식어를 포함하지 않는 모든 디렉토리를 제거한다.

   ~~MyApp/res/drawable/~~
   MyApp/res/drawable-en/
   MyApp/res/drawable-en-port/
   MyApp/res/drawable-en-notouch-12key/
   ~~MyApp/res/drawable-port-ldpi/~~
   ~~MyApp/res/drawable-port-notouch-12key~~

   예외 : 만약 문제되는 수식어가 스크린 픽셀 밀도라면, 안드로이드는 디바이스에 가장 근접하게 일치하는 옵션을 선택할 것이다. 그리고 그 선택 절차는 완료될 것이다. 일반적으로, 안드로이드는 더 작은 원본 이미지를 키우는 것보다 더 큰 원본 이미지를 줄이는 것을 선호한다.

5. 뒤로 돌아가서, 오직 하나의 선택이 남을 때까지 단계 2,3, 그리고 4를 반복한다. 예제에서는 스크린 오리엔테이션이 테이블에서 우리가 일치하는 다음 수식어이다. 스크린 오리엔테이션을 지정하지 않은 리소스를 제거한다.

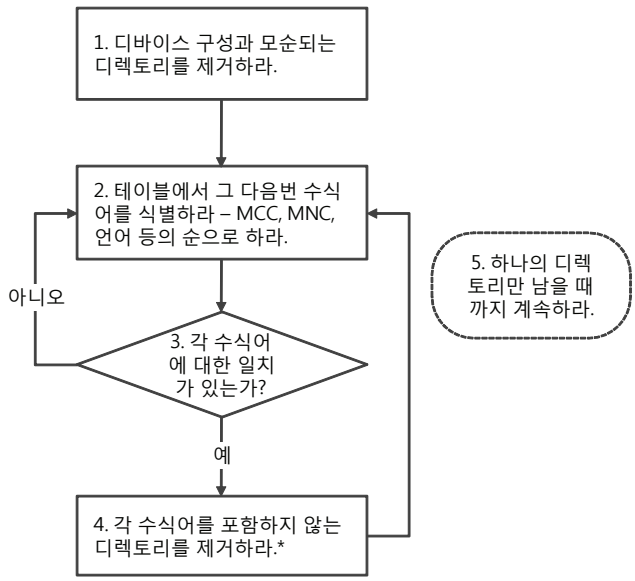
   ~~MyApp/res/drawable-en/~~
   MyApp/res/drawable-en-port/
   ~~MyApp/res/drawable-en-notouch-12key/~~

   오직 하나의 선택이 남는다. 그래서 그것으로 끝난다. 이 예제 애플리케이션에서 드로어블drawable이 호출될 때, 안드로이드 시스템은 MyApp/res/drawable-en-port/ 디렉토리로부터 리소스를 로드할 것이다.

   또한, 만약에 로드load된 리소스가 비트맵이라면, 제공되었던 저밀도low density 비트맵이 디바이스의 중간 밀도medium density에 맞도록 크기가 커진다.

> **Tip** 디바이스와 정확하게 일치하는 수식어의 갯수보다 수식어의 순서가 더 중요하다. 예를 들어 위의 단계 4에서, drawable-en은 오직 한 개의 파라미터만 일치하는 (언어) 반면, 그 리스트에서 마지막 선택은 디바이스와 정확하게 일치하는 세 개의 수식어(오리엔테이션, 터치 스크린, 그리고 입력 방법)를 포함한다. 하지만 언어가 더 높은 서열을 가지며, 그러므로 drawable-port-notouch-12key는 제거된다

다음에 나오는 순서도는 안드로이드가 로드할 리소스 디렉토리를 선택하는 방법을 정리한다.



* 또는 수식어가 스크린 밀도라면, 안드로이드는 이 시점에서 가장 일치되는 디렉토리를 선택하고 과정은 종료된다.

용 어(Teminology)

리소스 시스템은 최종적인 완전한 리소스 기능을 만들기 위해 다수의 다른 조각들을 함께 가져온다. 전체 시스템 이해를 돕기 위해, 여기에 여러분이 그것을 사용하면서 접하게 될 핵심 개념과 컴포넌트에 대한 약간의 간단한 정의가 있다.

Asset(애셋): 애플리케이션과 연관된 하나의 데이터 블러브blob. 이것은 java 소

스 코드, (PNG 이미지와 같은) 그래픽, XML 파일 등으로부터 컴파일된 오브젝트 파일들을 포함한다. 하나의 디렉토리 계층구조로 구성된 이 파일들은, 애플리케이션의 최종 패키징 과정에서 하나의 ZIP 파일로 함께 묶여진다.

aapt: 안드로이드 에셋 패키징 도구(Android Asset Packaging Tool). 애플리케이션 에셋의 최종 ZIP 파일을 생성하는 도구. 그것은 원시raw 에셋을 함께 모으는 것에 추가해서, 리소스 정의를 바이너리 에셋 데이터로 또한 분석한다parse.

리소스 테이블: 여러분을 위해 aapt가 생성하는 특별한 에셋으로, application/package 안에 포함된 모든 리소스를 설명한다. 여러분은 이 파일을 리소스 클래스를 통해 접근할 수 있다. 그것은 애플리케이션에 의해 직접적으로 터치되지 않는다.

리소스: 하나의 지칭된 값을 설명하는 리소스 테이블 내의 엔트리entry. 대체로 그들은 두 가지 타입의 리소스가 있다: 프리미티브primitives와 백bags이 그것이다.

리소스 식별자(Identifier): 리소스 테이블에 있는 모든 리소스는 고유한 정수integer number에 의해 식별된다. 소스 코드(리소스 설명들, XML 파일들, Java 소스 코드)에서 여러분은 실제 리소스 식별자 정수integer에 대한 상수constant인 심볼릭 이름을 사용할 수 있다.

프리미티브(Primitive) 리소스: 모든 프리미티브 리소스는 리소스 시스템에 포함된 다양한 프리미티브 타입(정수(integer), 컬러(color), 문자열(string), 다른 리소스에 대한 레퍼런스 등)을 기술하는 포맷팅을 사용해서 간단한 문자열로 작성될 수 있다. 비트맵과 XML 표현과 같은 복잡한 리소스은 프리미티브 문자열 리소스로 저장되며, 그 문자열 리소스의 값은 그것의 실제 데이터를 보유하는 에셋의 경로이다.

백(Bag) 리소스: 간단한 문자열 대신, 임의의 이름/값 쌍name/value pairs 리스트를 보유하는 특별한 종류의 리소스 엔트리. 각각의 이름은 그 자체로 리소스 식별자이며, 그리고 각각의 값은 보통의 리소스와 같은 종류의 문자열로 포맷된 데이터를 보유할 수 있다. 백Bags은 또한 상속을 지원한다. 백bag은 다른 백bag으로부터 값들을 상속할 수 있으며, 그것 자신의 컨텐츠들을 생성하기 위해 그것들을 선택적으로 대체하거나 또는 확장할 수 있다.

종류(kind): 리소스 종류는 다양한 목적을 위해 리소스 식별자를 구성하는 방법

이다. 예를 들어 드로어블drawable 리소스는 드로어블Drawable 오브젝트를 인스턴스화하기 위해 이용된다. 그래서 그들의 데이터는 비트맵 또는 XML 에셋에 대한 컬러 상수나 문자열 경로를 포함하는 프리미티브 리소스이다. 그 밖의 보편적인 리소스 종류에는 문자열(로컬라이즈된 문자열 프리미티브), 컬러(컬러 프리미티브), 레이아웃(뷰 레이아웃을 기술하는 XML 에셋에 대한 문자열 경로), 그리고 스타일(사용자 인터페이스 애트리뷰트를 기술하는 백(bag) 리소스)이 있다. 또한 표준 "attr" 리소스 종류가 있다. 이것은 백bag 아이템과 XML 애트리뷰트를 네이밍naming하기 위해 사용되는 리소스 식별자를 정의한다.

스타일: 사용자 인터페이스 애트리뷰트에 대한 하나의 집합을 제공하기 위해 사용되는 백bag을 포함하는 리소스 종류의 이름. 예를 들어 TextView 클래스에는 텍스트 크기, 컬러, 그리고 정렬을 정의하는 스타일 리소스가 주워질 수도 있다. 여러분은 레이아웃 XML 파일에 "style" 애트리뷰트를 사용하는 백bag을 가지고 하나의 스타일을 결합시킬 수 있다. 그 style의 값은 스타일 리소스의 이름이다.

스타일 클래스: 관련된 애트리뷰트 리소스의 집합을 명시한다. 이 데이터는 리소스 테이블 자체에 위치하지 않는다. 하지만 소스 코드에서 상수를 생성하기 위해 사용되며, 그것은 여러분으로 하여금 스타일 리소스 및/또는 XML 태그의 애트리뷰트로부터 값을 얻어오는 것을 더 용이하게 한다. 예를 들어 안드로이드 플랫폼은 모든 표준 뷰 애트리뷰트들(패딩(padding), visibility(비저블리티), background(백그라운드) 등)을 포함하는 "View" 스타일 클래스를 정의한다.

뷰가 인플레이트inflate될 때, 뷰는 XML 파일로부터 스타일 리소스 관련 값을 가져오기 위해 이 스타일 클래스를 사용하며(이 시점에 스타일과 테마 정보가 적절하게 적용된다), 스타일 리소스 값들을 뷰의 인스턴스 안으로 로드한다.

설정(Configuration): 어떤 특별한 리소스 식별자에 대해, 현재의 설정에 의존하는 복수의 다른 유효한 값들이 있을 수 있다. 설정은 로케일(언어와 국가), 스크린 오리엔테이션, 스크린 밀도 등을 포함한다. 현재 설정은 리소스 테이블이 로드될 때 어떤 리소스 값을 실제로 선택하기 위해 사용된다.

테마(Theme): 특별한 컨텍스트에 대한 글로벌 애트리뷰트 값을 제공하는 표준 스타일 리소스. 예를 들어 액티비티를 만들 때, 애플리케이션 개발자는 Theme.

White 또는 Theme.Black 스타일과 같은 사용할 표준 테마를 선택할 수 있다. 이 스타일은 스크린 백그라운드 이미지/컬러, 디폴트 텍스트 컬러, 버튼 스타일, 텍스트 에디터 스타일, 텍스트 크기 등과 같은 정보를 제공한다. 레이아웃 리소스를 인플레이트inflate할 때, 만약 위젯에 대한 대부분의 값(텍스트 컬러, 실렉터(selector), 백그라운드)이 명시적으로 설정되지 않는다면 현재 테마로부터 가져올 것이다. 레이아웃에 제공된 스타일과 애트리뷰트 값 또한 필요하다면 테마 애트리뷰트에서 명시적으로 명명된 값으로 그것들의 값을 지정할 수 있다.

오버레이(Overlay): 리소스의 새로운 집합을 정의하지 않는 리소스 테이블은, 대신에 다른 리소스 테이블에 있는 리소스 값을 대체한다. 설정과 같이 이것은 로드 시점에 리소스 데이터에 적용된다. 그것은 새로운 설정 값(예를 들어 새로운 로케일 안의 문자열)을 추가할 수 있고, 기존에 있는 값(예를 들어 "Hello Kitty" 백그라운드 이미지로 표준 화이트 백그라운드를 변경)을 대체하고, 리소스 백bag(예를 들어 18 pt 폰트 크기로 Theme.White 스타일의 폰트 사이즈를 변경)을 수정할 수 있다. 이것은 사용자로 하여금 그들의 디바이스의 전체적인 외관appearance을 선택하는 것을 가능하게 하며, 새로운 외관을 사용하는 파일을 다운로드하는 것을 가능하게 하는 설비이다.

리소스 레퍼런스(Resource Reference)

"사용가능한 리소스" 문서는 리소스의 다양한 타입에 대한 상세한 리스트와 Java 소스 코드, 또는 다른 레퍼런스에서 그것들을 사용하는 방법을 제공한다.

국제화 및 지역화(Internationalization and Localization)

보강 예정 : 국제화Internationalization와 지역화Localization는 중요하다. 하지만 또한 현재 SDK에서는 아직 완전히 준비되어 있지 않다. SDK가 성숙해지면서, 이 섹션은 안드로이드 플랫폼의 국제화와 지역화 특징들에 대한 정보를 포함할 것이다. 그 동안에는, 모든 문자열을 외부화시켜서, 리소스를 생성하고 사용하기 좋은 구조를 유지하고 있는 것이 좋다. (역주 : 최근 지역화와 관련된 구글 문서가 추가되고 있음, http://developer.android.com/guide/topics/resources/localization.html)

사용가능한 리소스 타입

이 페이지는 여러분의 애플리케이션의 코드와 패키지로부터 외부화할 수 있는 다

른 타입들의 리소스를 설명한다. 여러분의 애플리케이션에서 리소스를 사용하는 방법에 대한 보다 상세한 것에 대해서 앞서 언급된 "리소스와 국제화"를 보라.

간단한 값(Simple Values)

모든 간단한 리소스 값들은 하나의 문자열로 표시될 수 있다. 그것은 생성된 리소스의 타입을 명백하게 가리키기 위해 다양한 포맷을 사용하고 있다. 이런 이유로, 이들 값은 스타일과 테마의 매핑, 그리고 레이아웃과 같은 XML 파일의 애트리뷰트에 대해 제공된 직접적인 값 뿐만 아니라, 표준 리소스(res/values/ 아래에 있음) 두 가지 모두로 정의될 수 있다.

컬러 값(Color Values)

컬러 값은 알파 채널을 가지는 RGB 값을 지정한다. 이것은 드로어블Drawable을 위한 솔리드 컬러solid color를 지정하거나 텍스트를 위해 사용할 컬러를 지정하는 것과 같이 다양한 장소에서 사용될 수 있다. 컬러 값은 항상 파운드(#) 문자로 시작하며, 그리고 나서 알파(Alpha)-빨강(Red)-초록(Green)-파랑(Blue) 정보가 이어진다. 이것은 다음과 같은 포맷 중 하나이다.

- #RGB
- #ARGB
- #RRGGBB
- #AARRGGBB

만약 여러분이 리소스 ID에 대응되는 컬러를 얻고자 한다면, 여러분은 Resources.getColor() 메쏘드를 호출할 수 있다.

소스 파일 포맷: XML 파일은 〈?xml version="1.0" encoding="utf-8"?〉 선언을 요구하며, 그리고 루트 〈resources〉 엘리먼트는 하나 또는 그 이상의 〈color〉 태그를 포함한다.

리소스 소스 파일 위치: res/values/colors.xml (파일 이름은 상관없음)

컴파일된 리소스 데이터 타입: Java int 에 대한 리소스 포인터

리소스 레퍼런스 이름:

- Java: R.color.some_name
- XML: @[package:]color/some_name (some_name은 특정 컬러명임).

구문

```
<color name=color_name>#color_value</color>
```

〈color〉

값은 위에서 설명한 것처럼 웹 스타일 구문을 사용하는 컬러이며 오직 하나의 애트리뷰트만을 가진다.

- name – 이 컬러를 가리키는데 사용되는 이름.

예제 XML 선언(Example XML Declaration)

다음의 코드는 두 개의 컬러를 선언한다. 첫 번째는 완전 불투명이고, 두 번째는 반투명이다.

```
<resources>
   <color name="opaque_red">#f00</color>
   <color name="translucent_red">#80ff0000</color>
</resources>
```

예제 코드 사용(Example Code Use)

예제 Java 코드:

```
// Retrieve a color value.
int color = getResources.getColor(R.color.opaque_red);
```

예제 XML 코드:

```
<TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textAlign="center"
          android:textColor="@color/translucent_red"
          android:text="Some Text"/>
```

(역주: 위의 XML 예제 코드에서 사용된 textAlign 애트리뷰트는 더이상 사용되지 않음.)

문자열과 스타일 텍스트(Strings and Styled Text)

선택사항으로 간단한 포맷팅을 가지는 문자열은 리소스로써 저장되고 얻어질 수 있다. 여러분은 세 개의 표준 HTML 태그(〈b〉, 〈i〉, 그리고 〈u〉)를 사용해서 여러분의 문자열에 포맷팅을 추가할 수 있다. 스타일이 적용되지 않은 문자열(원시 텍스트)만을 얻는 것을 보증하기 위해서는, 얻어 온 CharSequence 오브젝트의 toString() 메쏘드를 호출하라. 문자열 리소스를 수용하는 메쏘드는 이러한 스타일링 태그를

처리할 수 있어야 한다.
만약 여러분이 리소스 ID에 대응되는 문자열을 얻고자 한다면, 여러분은 Context.getString() 메쏘드를 호출할 수 있다.

노트: 여러분이 만약 여러분의 문자열에서 어퍼스트로피(') 또는 따옴표를 사용한다면, 여러분은 그것을 이스케이프escape하거나 또는 다른 종류의 따옴표로 전체 문자열을 감싸야 한다. (역주: 만약 문자열에 작은 따옴표가 있다면 큰 따옴표로 전체 문자열을 감싸고, 큰 따옴표가 있다면 작은 따옴표로 전체 문자열을 감싸야 한다.)

```
<string name="good_example">"This'll work"</string>
<string name="good_example_2">This\'ll also work</string>
<string name="bad_example">This won't work!</string>
<string name="bad_example_2">XML encodings won&apos;t work either!
</string>
```

소스 파일 포맷: XML 파일은 〈?xml version="1.0" encoding="utf-8"?〉 선언을 요구하며, 루트 〈resources〉 엘리먼트는 하나 또는 그 이상의 〈string〉 태그를 포함한다.
리소스 소스 파일 위치: res/values/strings.xml (파일 이름은 상관없음)
컴파일된 리소스 데이터 타입: Java CharSequence 에 대한 리소스 포인터
리소스 레퍼런스 이름:

- Java: R.string.some_name
- XML: @[package:]string/some_name(some_name 은 특정 문자열의 이름임).

구문

```
<string name=string_name>string_value</string>
```

〈string〉

값은 선택사항으로 스타일링 태그를 가지는 문자열이며 오직 하나의 애트리뷰트를 가진다.

- name – 이 문자열을 가리키는데 사용되는 이름.

예제 XML 선언(Example XML Declaration)

아래의 XML 예제는 두 개의 문자열을 선언한다. 첫 번째는 포맷팅 없는 간단

한 텍스트(간단한 스트링 오브젝트인 CharSequence)이며, 두 번째는 문자열에 포맷팅 정보(복잡한 데이터 구조인 CharSequence)를 포함한다. 만약 여러분이 이클립스에서 문자열 파일에 대한 커스텀 에디터를 사용한다면, HTML 포맷팅 태그는 자동으로 사라질 것이며, 여러분은 리소스를 얻기 위해 Context.getString()과 fromHtml(String)를 사용할 필요가 있고, 그런 다음에 그것을 포맷된 텍스트로 변환해야 한다. (역주 : getString()대신 getText()를 사용하여야 함.)

```
<resources>
   <string name="simple_welcome_message">Welcome!</string>
   <string name="styled_welcome_message">We are <b><i>so</i></b>
      glad to see you.</string>
</resources>
```

예제 코드 사용(Example Code Use)

예제 Java 코드:

```
// Assign a styled string resource to a TextView
// on the current screen.
CharSequence str = getString(R.string.styled_welcome_message);
TextView tv = (TextView)findViewById(R.id.text);
tv.setText(str);
```

(역주 : 위의 예제 코드에서 getString()을 getText()로 바꾸어야 스타일 문자열을 얻을 수 있음.)

예제 XML 코드:

```
<TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textAlign="center"
          android:text="@string/simple_welcome_message"/>
```

포맷 문자열로써 스타일이 있는 텍스트 사용하기

가끔 여러분은 포맷 문자열로써도 사용될 수 있는 스타일을 가진 텍스트 리소스를 생성하길 원할 수 있다. 이것은 직접적으로 될 수는 없다. 왜냐하면 스타일 정보를 제거하지 않고 String.format()의 포맷 문자열 아큐먼트로써 스타일을 가진 텍스트를 전달하는 방법이 없기 때문이다. 차선책은 이스케이프된escaped HTML 태그로써 스타일 태그를 저장하는 것이다. 그리고 나서 포맷팅을 한 후에 이스케이프된escaped HTML 문자열을 스타일을 가진 텍스트로 변환하라.

포맷 문자열로써 스타일이 있는 텍스트를 사용하기 위해서는 다음을 수행하라.

1. 여러분의 텍스트 리소스에 있는 HTML 태그들이 마치 XML 태크인 것처럼 해석되지 않도록 하기 위해, 이스케이프된escaped 문자열로 여러분의 스타일이 있는 텍스트를 저장하라.

```
<resources>
<string name="search_results_resultsTextFormat">%1$d results
for &lt;b>&amp;&quot;%2$s&amp;&quot;&lt;/b></string>
</resources>
```

   이 예제에서 포맷 문자열은 두 개의 아규먼트를 가진다. %1$d는 숫자, %2$s는 문자열임.

2. 만약 문자열 아규먼트들이 '<' 또는 '&' 문자를 포함한다면, 그 문자열 아규먼트들이 적절하게 이스케이프escape 되도록 하라. htmlEncode(String) 메쏘드는 이것을 할 것이다.

```
String escapedTitle = TextUtils.htmlEncode(title);
```

3. HTML 텍스트를 포맷하기 위해 String.format()을 사용하라. 그리고 나서 HTML 텍스트를 스타일을 가진 텍스트로 변환하기 위해 fromHtml(String)을 사용하라.

```
String resultsTextFormat =
    getContext().getResources().getString(
            R.string.search_results_resultsTextFormat);
String resultsText
    = String.format(resultsTextFormat, count, escapedTitle);
CharSequence styledResults = Html.fromHtml(resultsText);
```

크기 값(Dimension Values)

여러분은 XML로 크기dimension 값을 정의함으로써, 다양한 스크린 엘리먼트를 위해 사용할 일반적인 크기를 생성할 수 있다. 크기dimension 리소스는 측정 단위가 뒤에 붙는 하나의 숫자이다. 예를 들어 10px, 2in, 5sp가 그것이다. 여기에 안드로이드에 의해 지원되는 측정 단위가 있다.

px

픽셀 – 스크린 상의 실제 픽셀에 대응한다.

in

인치 – 스크린의 물리적 크기에 기반한다.

mm

밀리미터 – 스크린의 물리적 크기에 기반한다.

pt

포인트 – 스크린의 물리적 크기에 기반한 인치(inch)의 1/72.

dp

밀도 독립적 픽셀Density-independent Pixels – 스크린의 물리적 밀도에 기반한 추상적 단위. 이 단위는 160 dpi의 스크린에 비례한다. 그래서 1 dp는 160 dpi 스크린 상의 1 픽셀이다. dp-to-pixel 비율은 스크린 밀도에 따라 변할 것이다. 하지만 반드시 정비례는 아니다. 주의: "dp"가 "sp"와 더 잘 맞음에도 불구하고, 컴파일러는 "dip"와 "dp" 모두를 허용한다.

sp

스케일 독립적 픽셀Scale-independent Pixels – 이것은 dp 단위와 같다. 그러나 이것은 또한 사용자의 폰트 크기 설정에 의해 스케일된다. 여러분이 폰트 크기를 지정할 때 이 단위를 사용할 것을 권장한다. 따라서 폰트 크기는 스크린 밀도와 사용자 설정 모두에 대해서 조정될 것이다.

크기dimension 값은 일반적으로 원시 리소스로서 사용되기 보다는 오히려 XML 파일의 애트리뷰트 값으로써 사용된다. 하지만 여러분은 이 데이터 타입을 포함하는 일반적인 리소스를 생성할 수 있다.

소스 파일 포맷: XML 파일은 〈?xml version="1.0" encoding="utf-8"?〉 선언을 요구하며, 루트 〈resources〉 엘리먼트는 하나 또는 그 이상의 〈dimen〉 태그를 포함한다.

리소스 소스 파일 위치: res/values/dimens.xml (파일 이름은 상관없음. 표준 사례는 치수를 다루는 하나의 파일에 모든 치수들을 넣는 것이다).

컴파일된 리소스 데이터타입: dimension에 대한 리소스 포인터.

리소스 레퍼런스 이름:

- Java: R.dimen.some_name
- XML: @[package:]dimen/some_name (some_name 은 특정 〈dimen〉 엘리먼트임)

구문

```
<dimen name=dimen_name>dimen_value</dimen>
```

〈dimen〉

유효한 치수 값.

- name – 이 치수를 가리키는데 사용되는 이름.

예제 XML 선언(Example XML Declaration)

다음의 코드는 몇가지 치수 값을 선언한다.

```
<resources>
    <dimen name="one_pixel">1px</dimen>
    <dimen name="double_density">2dp</dimen>
    <dimen name="sixteen_sp">16sp</dimen>
</resources>
```

예제 코드 사용(Example Code Use)

예제 Java 코드:

```
float dimen = Resources.getDimen(R.dimen.one_pixel);
```

예제 XML 코드:

```
<TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textSize="@dimen/sixteen_sp"/>
```

드로어블(Drawables)

드로어블Drawable은 여러분이 Resources.getDrawable()을 사용해서 얻어와서 스크린에 그리기 위해 사용하는 리소스의 한 타입이다. 생성될 수 있는 다수의 드로어블drawable 리소스들이 있다.

비트맵 파일(Bitmap Files)

안드로이드는 다소 다른 포맷 – png(선호되는), jpg(수용가능한), gif(좋지않은) – 의 비트맵 리소스 파일을 지원한다. 비트맵 파일 그 자체는 컴파일되어 확장자 없는 파일명으로 레퍼런스된다(그러므로 res/drawable/my_picture.png는 R.drawable.my_picture 로써 레퍼런스될 것이다).

소스 파일 포맷: png(선호되는), jpg(수용가능한), gif(좋지않은). 파일당 하나의 리소스.

리소스 파일 위치: res/drawable/some_file.png

컴파일된 리소스 데이터 타입: BitmapDrawable에 대한 리소스 포인터.
리소스 레퍼런스 이름:

- Java: R.drawable.some_file
- XML: @[package:]drawable/some_file

드로어블Drawable 리소스 사용에 대한 더 많은 논의와 예제는, 10장. "그래픽"의 "2차원 그래픽"을 보라.

컬러 드로어블(Color Drawables)

여러분은 선택사항으로 둥근 모서리를 가지는 직사각형의 컬러 PaintDrawable 오브젝트를 만들 수 있다. 이 엘리먼트는 res/values/ 내의 어떤 파일에서나 정의될 수 있다.

소스 파일 포맷: XML 파일은 〈?xml version="1.0" encoding="utf-8" ?〉 선언을 요구하며, 루트 〈resources〉 엘리먼트는 하나 또는 그 이상의 〈drawable〉 태그를 포함한다.
리소스 소스 파일 위치: res/values/colors.xml (파일이름은 상관없음. 표준 사례는 파일에 숫자 컬러 값과 PaintDrawable 아이템을 넣는 것이다)
컴파일된 리소스 데이터타입: PaintDrawable에 대한 리소스 포인터.
리소스 레퍼런스 이름:

- Java: R.drawable.some_name
- XML: @[package:]drawable/some_name (some_name은 특정 리소스의 이름임)

구문

```
<drawable name=color_name>color_value</drawable>
```

〈drawable〉

유효한 컬러 값.

- name – 이 드로어블drawable을 가리키는데 사용되는 이름.

예제 XML 선언(Example XML Declaration)

다음 코드는 몇가지 컬러 드로어블(drawables)을 선언한다.

```
<resources>
    <drawable name="solid_red">#f00</drawable>
    <drawable name="solid_blue">#0000ff</drawable>
    <drawable name="solid_green">#f0f0</drawable>
</resources>
```

예제 코드 사용(Example Code Use)

예제 Java 코드:

```
// Assign a PaintDrawable as the background to
// a TextView on the current screen.
Drawable redDrawable = Resources.getDrawable(R.drawable.solid_red);
TextView tv = (TextView)findViewById(R.id.text);
tv.setBackground(redDrawable);
```

예제 XML 코드:

```
<TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textAlign="center"
          android:background="@drawable/solid_red"/>
```

나인-패치 확대가능한 이미지(Nine-Patch Stretchable Image)

안드로이드는 나인패치NinePatch 그래픽이라 불리는 확대가능한stretchable 비트맵 이미지를 지원한다. 이것은 PNG 이미지이며, 여러분은 그 안에 확대가능한 stretchable 섹션을 정의하며, 안드로이드는 디스플레이 시점에 이 오브젝트를 텍스트 문자열과 같은 가변적 크기 섹션을 수용할 수 있도록 크기를 변경할 것이다. 여러분은 일반적으로 뷰의 백그라운드에 이 리소스를 할당한다. 확대가능한 Stretchable 이미지의 사용 예제는 안드로이드가 사용하는 버튼Button 백그라운드이다. 버튼은 다양한 길이의 문자열을 수용하기 위해 확대되어야 한다.

소스 파일 포맷: PNG – 파일 별로 하나의 리소스
리소스 소스 파일 위치: res/drawable/some_name.9.png (.9.png 로 끝나야 함.)
컴파일된 리소스 데이터타입: NinePatchDrawable에 대한 리소스 포인터
리소스 레퍼런스 이름:

- Java: R.drawable.some_file
- XML: @[package:]drawable.some_file

나인패치NinePatch 드로어블drawables을 사용하는 더 많은 정보와 예제는, 10장. "그래픽"의 "2차원 그래픽"을 보라.

애니메이션(Animation)

트윈 애니메이션(Tweened Animation)

안드로이드는 그래픽 또는 그래픽 시리즈에 간단한 애니메이션을 수행할 수 있다. 이것들은 회전rotations과 페이딩fading, 움직임moving, 확대stretching를 포함한다.

소스 파일 포맷: XML 파일, 파일 별로 하나의 리소스, 〈?xml〉 선언을 갖지 않는 하나의 루트 태크
리소스 파일 위치: res/anim/some_file.xml
컴파일된 리소스 데이터타입: Animation에 대한 리소스 포인터.
리소스 레퍼런스 이름:

- Java: R.anim.some_file
- XML: @[package:]anim/some_file

구 문

파일은 하나의 루트 엘리먼트를 가져야 한다. 이것은 하나의 〈alpha〉, 〈scale〉, 〈translate〉, 〈rotate〉, interpolator 엘리먼트, 또는 이러한 엘리먼트들의 그룹을 보유하는 〈set〉 엘리먼트가 될 수 있다(〈set〉은 또 다른 〈set〉을 포함할 수도 있다). 디폴트로, 모든 엘리먼트들은 동시에 적용된다. 그것들이 순차적으로 일어나게 하기 위해서는, 여러분이 startOffset 애트리뷰트를 지정해야 한다.

```
<set android:shareInterpolator=boolean> // Only required
                                        // if multiple tags are used.
   <alpha android:fromAlpha=float
          android:toAlpha=float >   |
   <scale android:fromXScale=float
          android:toXScale=float
          android:fromYScale=float
          android:toYScale=float
          android:pivotX=string
          android:pivotY=string >    |
   <translate android:fromX=string
```

```
            android:toX=string
            android:fromY=string
            android:toY=string >   |
   <rotate android:fromDegrees=float
           android:toDegrees=float
           android:pivotX=string
           android:pivotY=string > |
   <interpolator tag> <set>
</set>
```

엘리먼트와 애트리뷰트(Elements and Attributes)

〈set〉

자기 자신 또는 다른 애니메이션을 재귀적으로 보유할 수 있는 컨테이너. AnimationSet을 표현한다. 여러분이 원하는 만큼, 동일하거나 또는 다른 타입의 자식child 엘리먼트를 많이 포함할 수 있다. 다음의 애트리뷰트를 지원한다.

- shareInterpolator – 모든 인접한 자식child 엘리먼트 사이에서 같은 인터폴레이터interpolator를 공유하는지에 대한 여부.

〈alpha〉

페이딩fading 애니메이션. AlphaAnimation을 표현한다. 다음의 애트리뷰트를 지원한다.

- fromAlpha – 0.0 에서 1.0, 여기에서 0.0은 투명임.
- toAlpha – 0.0 에서 1.0, 여기에서 0.0은 투명임.

〈scale〉

크기를 변경하는 애니메이션. ScaleAnimation를 표현한다. 여러분은 pivotX와 pivotY를 지정함으로써 그것으로부터 바깥 쪽(또는 안 쪽)으로 커지는 이미지의 중심 지점(핀이 꽂힌 중심)을 지정할 수 있다. 예를 들어 만약 이것들이 0,0(왼쪽 상단 모서리)인 경우, 모든 증가는 아래와 우측으로 될 것이다. 스케일은 다음의 애트리뷰트를 지원한다.

- fromXScale – 시작하는 X 크기(여기에서 1.0은 어떤 변화도 없는 것임)
- toXScale – 끝나는 X 크기(여기에서 1.0이 어떤 변화도 없는 것임)
- fromYScale – 시작하는 Y 크기(여기에서 1.0은 어떤 변화도 없는 것임)
- toYScale – 끝나는 Y 크기(여기에서 1.0은 어떤 변화도 없는 것임)

- pivotX – 핀이 꽂힌 중심의 X 좌표.
- pivotY – 핀이 꽂힌 중심의 Y 좌표.

〈translate〉

수직/수평 움직임 애니메이션. TranslateAnimation을 표현한다. 뒤따르는 세 개의 포맷으로 된 어떤 애트리뷰트든 지원한다. "%"로 끝나는 -100에서 100까지 값은, 그것 자신에 상대적인 백분율을 가리킨다. "%p"로 끝나는 -100에서 100까지 값은, 자신의 부모parent에 상대적인 백분율을 가리킨다. 접미어가 없는 부동소수점은 절대 값을 가리킨다.

- fromXDelta – 시작하는 X 위치.
- toXDelta – 끝나는 X 위치.
- fromYDelta – 시작하는 Y 위치.
- toYDelta – 끝나는 Y 위치.

〈rotate〉

회전 애니메이션. RotateAnimation을 표현한다. 다음의 애트리뷰트를 지원한다.

- fromDegrees – 회전을 시작하는 각도.
- toDegrees – 회전을 끝내는 각도.
- pivotX – 회전의 중심에 대한 픽셀 X 좌표, 여기에서 (0,0)은 왼쪽 상단 모서리.
- pivotY – 회전의 중심에 대한 픽셀 Y 좌표, 여기에서 (0,0)은 왼쪽 상단 모서리.

〈interpolator tag〉

여러분은 또한 R.styleable에서 정의된 임의의 interpolator 서브클래스 엘리먼트를 사용할 수 있다. 예제들은 〈CycleInterpolator〉, 〈EaseInInterpolator〉, 그리고 〈EaseOutInterpolator〉를 포함한다. 이 오브젝트는 타임라인 상에서 시각적 액션이 얼마나 빠르게 일어나는가를 설명하는 속도 곡선을 정의한다(먼저 빠르게 그리고 나중에 느리게, 먼저 느리게 그리고 점차적으로 빠르게, 그리고 기타 등등).

위의 각각의 엘리먼트에 대해 정의된 애트리뷰트에 추가적으로, 엘리먼트 〈alpha〉, 〈scale〉, 〈translate〉, 〈rotate〉, 그리고 〈set〉 모두는 (Animation 클래스로부터 상속된) 다음과 같은 애트리뷰트들을 지원한다.

duration

이 효과가 대한 기간. (단위, 1/1000초)

startOffset

이 효과에 대한 오프셋 시작 시각. (단위, 1/1000초)

fillBefore

참(true)으로 설정할 때, 애니메이션이 시작되기 전에 애니메이션 변형이 적용된다.

fillAfter

참(true)으로 설정할 때, 애니메이션이 끝난 후에 애니메이션 변형이 적용된다.

repeatCount

애니메이션이 반복되어야 하는 횟수를 정의한다.

repeatMode

애니메이션이 반복 횟수repeat count가 0보다 클 때, 애니메이션이 끝나는 순간의 동작을 정의한다. 옵션은 애니메이션을 다시 시작하거나 또는 역으로 진행시키는 것이다.

zAdjustment

애니메이션을 실행할 때 사용하는 z-축 지시 모드 (normal, top, 또는 bottom).

interpolator

여러분은 선택사항으로 각각의 엘리먼트에 대해 전체 애니메이션 시간동안 그것의 효과를 얼마나 빠르게 또는 느리게 수행할 것인가를 결정하는 인터폴레이터interpolator를 지정할 수 있다. 예를 들어 EaseInInterpolator는 처음에는 느리게 그리고 끝에서 더 빠르며, 그리고 EaseOutInterpolator는 그 반대이다. Interpolators 리스트는 R.anim에서 제공된다. 이들을 지정하기 위해서는 @android:anim/interpolatorName 구문을 이용하라.

더 많은 논의와 애니메이션 코드 샘플을 위해서는 10장. "그래픽"의 "2차원 그래픽"을 보라.

메뉴(Menus)

애플리케이션 메뉴(옵션 메뉴, 컨텍스트 메뉴, 또는 서브 메뉴)는 XML 리소스로써 정의될 수 있으며, 여러분의 애플리케이션에 의해 MenuInflater를 사용해서 인플레이트 inflate 된다.

소스 파일 포맷: XML 파일, 파일 별로 하나의 리소스, 하나의 루트 태그, 〈?xml〉 선언이 요구되지 않음.
리소스 파일 위치: res/menu/some_file.xml
컴파일된 리소스 데이터타입: Menu (또는 서브 클래스) 리소스에 대한 리소스 포인터.
리소스 레퍼런스 이름:

- Java: R.menu.some_file

구 문

그 파일은 하나의 루트 엘리먼트인 〈menu〉 엘리먼트를 가져야 한다. 모두 합쳐 세 개의 유효한 엘리먼트가 있다. 〈menu〉, 〈group〉, 그리고 〈item〉이 그것이다. 〈item〉과 〈group〉 엘리먼트는 〈menu〉의 자식들children이 되어야 한다. 그러나 〈item〉 엘리먼트 또한 〈group〉의 자식들children이 될 수 있으며, (서브 메뉴를 생성하기 위해) 또 다른 〈menu〉 엘리먼트가 〈item〉의 자식child가 될 수도 있다.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/example_item
          android:title="Example Item"
          android:icon="@drawable/example_item_icon" />
    <group android:id="@+id/example_group">
        <item android:id="@+id/example_item2
              android:title="Example Item 2"
              android:icon="@drawable/example_item2_icon" />
    </group>
    <item android:id="@+id/example_submenu
          android:title="Example Sub Menu" >
        <menu>
            <item android:id="@+id/example_submenu_item
                  android:title="Example Sub Menu Item" />
        </menu>
    </item>
</menu>
```

엘리먼트와 애트리뷰트(Elements and Attributes)

모든 애트리뷰트는 android 네임스페이스(예를 들어 android:icon="@drawable/icon)를 가지고 정의되어야 한다.

〈menu〉

메뉴의 루트. 〈item〉 과 〈group〉 노드들을 포함한다. 어떠한 애트리뷰트도 가지지 않는다.

〈group〉

메뉴 그룹. 〈item〉 엘리먼트를 포함한다. 유효한 애트리뷰트는 다음과 같다.

- id – 그룹에 대한 고유한 정수integer ID.
- menuCategory – 메뉴 CATEGORY_*상수에 대응하는 값 – 그룹의 우선순위를 정의한다. 유효한 값: container, system, secondary, 그리고 alternative.
- orderInCategory – 카테고리 내에서 아이템의 디폴트 순서를 정의하는 정수integer.
- checkableBehavior – 아이템의 체크 가능 여부. 유효한 값: none, all(상호배타적인/라디오 버튼), single(배타적이지 않은/체크박스)
- visible – 그룹이 보여지는 지 여부. 참true 또는 거짓false.
- enabled – 그룹이 활성화 가능한 지 여부. 참true 또는 거짓false.

〈item〉

메뉴 아이템. 하나의 〈menu〉 엘리먼트(서브 메뉴를 위해)를 포함할 수 있다. 유효한 애트리뷰트는 다음과 같다.

- id – 아이템에 대한 고유한 리소스 ID.
- menuCategory – 메뉴 카테고리를 정의하기 위해 사용된다.
- orderInCategory – 그룹 안에서 아이템의 순서를 정의하기 위해 사용된다.
- title – 메뉴 타이틀에 대한 문자열.
- titleCondensed – 일반적인 타이틀이 너무 긴 상황을 위해 간결한 문자열 타이틀.
- icon – 드로어블drawable 아이콘에 대한 리소스 식별자.
- alphabeticShortcut – 알파벳 단축키에 대한 문자.

- numericShortcut – 숫자 단축키에 대한 숫자.
- checkable – 아이템의 체크 가능 여부. 참true 또는 거짓false.
- checked – 아이템의 디폴트 체크 여부. 참true 또는 거짓false.
- visible – 아이템이 디폴드로 보여지는 지 여부. 참true 또는 거짓false.
- enabled – 아이템이 디폴트로 활성화 가능한 지 여부. 참true 또는 거짓false.

XML로 메뉴를 만드는 방법과 그것들을 여러분의 애플리케이션에서 인플레이트inflate 하는 방법에 대한 더 많은 논의는 "메뉴 생성하기" 섹션을 읽어라.

레이아웃(Layout)

안드로이드는 여러분이 HTML 파일로 웹 페이지에 대한 스크린 레이아웃을 디자인하는 것과 유사하게 XML 파일 내의 XML 엘리먼트를 사용하여 스크린 레이아웃을 지정하는 것을 가능하게 한다. 각각의 파일은 전체 스크린 또는 스크린의 일부를 포함하며, 그리고 뷰 리소스로 컴파일된다. 이것은 Activity.setContentView()에 전달되거나 또는 그 밖의 레이아웃 리소스 엘리먼트에 의해 레퍼런스로 사용될 수 있다. 파일은 여러분 프로젝트의 res/layout 폴더에 저장되며, 안드로이드 리소스 컴파일러인 aapt에 의해 컴파일된다.

모든 레이아웃 XML 파일은 하나의 루트 엘리먼트를 가져야 한다. 먼저 우리는 그것이 XML에 적재된 대로, 안드로이드에 의해 이해되는 표준 XML 태그를 사용하는 방법을 설명할 것이고, 그리고 나서 커스텀 뷰 오브젝트를 위한 여러분 자신의 커스텀 XML 엘리먼트 정의 방법에 대한 약간의 정보를 제공할 것이다.

루트 엘리먼트는 루트 엘리먼트에 정의된 "http://schemas.android.com/apk/res/android"Android 네임스페이스를 가져야 한다.
레이아웃을 생성하는 것에 대한 완전한 논의에 대해서는 3장. "사용자 인터페이스" 를 보라.

소스 파일 포맷: XML 파일은 〈?xml version="1.0" encoding="utf-8"?〉 선언과 그리고 지원되는 XML 레이아웃 엘리먼트 중 하나가 루트 엘리먼트가 되어야 한다.
리소스 파일 위치: res/layout/some_file.xml

컴파일된 리소스 데이터타입: View (또는 서브클래스) 리소스에 대한 리소스 포인터.
리소스 레퍼런스 이름:

- Java: R.layout.some_file
- XML: @[package:]layout/some_file

구 문

```
<ViewGroupClass
       xmlns:android="http://schemas.android.com/apk/res/android"
       id="@+id/string_name" (attributes)>
   <widget or other nested ViewGroupClass>+
   <requestFocus/>(0 or 1 per layout file, assigned to any element)
</ViewGroupClass>
```

〈ViewGroupClass〉

그 파일은 하나의 루트 엘리먼트를 가져야 한다. 이것은 다른 엘리먼트를 포함하는 뷰그룹 클래스가 될 수 있으며, 또한 만약 그것이 오직 하나의 오브젝트라면 위젯 (또는 커스텀 아이템)이 될 수도 있다. 디폴트로, 여러분은 엘리먼트로 어떤 (대소문자를 구분하는) 안드로이드 위젯이나 뷰그룹 클래스 이름을 사용할 수 있다. 이러한 엘리먼트들은 그 아래에 놓인(역주: 대응대는) 클래스에 적용하는 애트리뷰트를 지원한다. 하지만 그 네이밍naming은 분명하지 않다. 어떤 태그에서 어떤 어떤 애트리뷰트가 지원되는가를 찾는 방법이 아래에 설명되었다. 여러분은 어떤 네스팅nesting이든 유효하다고 가정해서는 안된다(예를 들어 여러분은 〈ListLayout〉 안에 〈TextView〉 엘리먼트를 포함할 수 없다).

만약 클래스가 다른 클래스로부터 파생된다면, XML 엘리먼트는 그것이 파생된 그 엘리먼트로 부터 모든 애트리뷰트를 상속한다. 그러므로 예를 들어 〈EditText〉는 EditText 클래스에 대응되는 XML 엘리먼트이다. 그것은 〈TextView〉와 〈View〉에 의해 지원되는 모든 애트리뷰트 뿐만 아니라, 그것 자신의 고유한 애트리뷰트(EditText_numeric)를 제시한다. XML 안에 있는 태그의 id 애트리뷰트에 대해, 여러분은 특별한 구문("@+id/somestringvalue")을 사용해야 한다. "@+" 구문은 만약 R.id 클래스에 리소스 숫자가 존재하지 않는다면 그것을 생성하며, 만약 그것이 존재한다면 그것을 사용한다. XML 태그에 대한 ID 값를 선언할 때, 이 구문을 사용하라. 예제: 〈TextView id= "@+id/nameTextbox"/〉, 그리고 Java에서 이러한 방법, findViewById(R.id.nameTextbox)로 그것을 레퍼런스하라. 모든 엘리먼

트는 다음의 값들을 지원한다.

- id – Java에서 이 엘리먼트에 접근하기 위해 사용되는 ID 값. 만일 여러분이 id.xml에서 스스로 그것을 생성하지 않았다면, 여러분은 일반적으로 여러분을 위한 ID를 생성하기 위해 @+id/string_name 구문을 사용할 것이다. (역주 : 만약 id.xml 에서 id를 일괄적으로 생성하고자 한다면, 아래와 같은 방식으로 간단하게 id를 생성할 수 있다. 〈resources〉 〈item type="id" name="string_name" /〉〈/resources〉)
- xmlns:android="http://schemas.android.com/apk/res/android" – 루트 엘리먼트에서 요구된다.

〈requestFocus〉

뷰 오브젝트에 해당하는 임의의 엘리먼트는 이런 빈empty 엘리먼트를 포함할 수 있다. 이것은 그것의 부모parent 태그에게 스크린 상의 초기 포커스를 제공한다. 여러분은 파일 별로 이 엘리먼트를 오직 하나만 가질 수 있다.

어떤 엘리먼트에 대해 어떤 애트리뷰트가 지원되는가?

안드로이드는 런타임에서 XML 레이아웃 리소스를 로드해서 그것을 비쥬얼 엘리먼트로 변경하기 위해 LayoutInflater 클래스를 사용한다. 디폴트로, 모든 위젯 클래스 이름은 태그로 직접적으로 사용된다. 더불어 지원되는 모든 태그와 애트리뷰트 리스트는 R.styleable 레퍼런스 페이지에 리스트된다. 하지만 애트리뷰트 이름은 다소 분명하지 않다. 만약 그 이름에 밑줄(_)이 나타나면, 이것은 그것이 애트리뷰트라는 것을 가리킨다(일반적으로 밑줄 앞이 엘리먼트의 이름이다). 그러므로 예를 들어 EditText_autoText는 〈EditText〉 태그가 애트리뷰트로 autoText를 지원한다는 것을 의미한다. 여러분이 실제로 그 엘리먼트에 있는 애트리뷰트를 사용할 때, 마지막 밑줄 이후의 영역만을 사용하라. 그리고 그 애트리뷰트에 접두어 "android:"를 붙여라. 그러므로 예를 들어 만약 R.styleable이 다음과 같은 값들을 리스트한다면,

- TextView
- TextView_lines
- TextView_maxlines

여러분은 다음과 같은 하나의 엘리먼트를 생성할 수 있다.

```
<TextView android:lines="10" android:maxlines="20"/>
```

이것은 TextView 오브젝트를 만들 것이고, 그것의 lines과 maxlines 속성들

을 설정할 것이다.

애트리뷰트는 세 개의 소스로부터 온다.

- 엘리먼트에 의해 직접적으로 보여지는 애트리뷰트. 예를 들어 위에서 논의된 것과 같이 TextView는 TextView_text를 지원한다.
- 해당 엘리먼트의 모든 슈퍼클래스에 의해 보여지는 애트리뷰트. 예를 들어 TextView 클래스는 View 클래스를 확장한다. 그러므로 〈TextView〉 엘리먼트는 〈View〉 엘리먼트가 보여주는 모든 애트리뷰트 – View_paddingBottom과 View_scrollbars를 포함하는 긴 리스트 – 를 지원한다. 이것들 또한 클래스 이름없이 사용된다. 〈TextView android:paddingBottom="20" android: scrollbars="horizontal"/〉.
- 오브젝트의 ViewGroup.LayoutParams 서브클래스의 애트리뷰트. 모든 뷰 오브젝트는 LayoutParams 멤버를 지원한다(3장. "사용자 인터페이스"의 "레이아웃 선언하기"를 보라). 엘리먼트의 LayoutParams 멤버에 속성들을 설정하기 위해, 사용할 애트리뷰트는 "android:layout_layoutParamsProperty"이다. 예를 들어 〈LinearLayout〉 엘리먼트에 의해 감싸여진 오브젝트에 대한 android:layout_gravity가 그것이다. 각각의 LayoutParams 서브클래스는 또한 상속된 애트리뷰트를 지원한다는 것을 기억하라. 각각의 서브클래스에 의해 보여지는 애트리뷰트들은 someLayoutParamsSubclass_Layout_layout_someproperty 포맷으로 주어진다. 이것은 애트리뷰트 "android:layout_someproperty"를 정의한다. 여기에 안드로이드 문서가 LinearLayout.LayoutParams 클래스의 속성을 어떻게 나열하는 지에 대한 예제가 있다.
    - LinearLayout_Layout // 실제 오브젝트 – 사용되지 않음.
    - LinearLayout_Layout_layout_gravity // gravity 애트리뷰트를 보여줌.
    - LinearLayout_Layout_layout_height // height 애트리뷰트를 보여줌.
    - LinearLayout_Layout_layout_weight // weight 애트리뷰트를 보여줌.
    - LinearLayout_Layout_layout_width // width 애트리뷰트를 보여줌.

여기에 직접적인 애트리뷰트, 상속된 애트리뷰트, 그리고 LayoutParams 애트리뷰트를 포함하는 몇 개의 오브젝트에, 이러한 값들 중 몇 개를 설정하는 예제가 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/main_screen.xml -->
<LinearLayout
       xmlns:android="http://schemas.android.com/apk/res/android"
       android:orientation="vertical"
               // The object's own orientation property
       android:padding="4"           // Inherited View property
       android:gravity="center"      // The object's own property
       android:layout_width="fill_parent"
               // Parent object's LinearLayout.LayoutParams.width
       android:layout_height="fill_parent">
               // Parent object's LinearLayout.LayoutParams.height
   <TextView android:layout_width="fill_parent"
               // TextView.LayoutParams.width
       android:layout_height="wrap_content"
               // TextView.LayoutParams.height
       android:layout_weight="0"
               // TextView.LayoutParams.weight
       android:paddingBottom="4"         // TextView.paddingBottom
       android:text="@string/redirect_getter"/> // TextView.text

   <EditText id="@+id/text"
       android:layout_width="fill_parent"
               // EditText.LayoutParams.width
       android:layout_height="wrap_content"
               // EditText.LayoutParams.height
       android:layout_weight="0"  // EditText.LinearLayoutParams.weight
       android:paddingBottom="4"> // EditText.paddingBottom
        <requestFocus />
   </EditText>

   <Button id="@+id/apply"
    android:layout_width="wrap_content"  // Button.LayoutParams.width
    android:layout_height="wrap_content" // Button.LayoutParams.height
    android:text="@string/apply" />      // TextView.text
</LinearLayout>
```

예제 코드 사용(Example Code Use)

가장 일반적인 사용은 XML 파일(res/layout/main_screen.xml)을 로드하고, 선행하는 파일을 가지고 아래에 보여지듯이 현재 스크린으로 그것을 사용하

는 것이다.

```
setContentView(R.layout.main_screen);
```

하지만 레이아웃 엘리먼트는 템플릿처럼 사용되는 반복하는 엘리먼트를 또한 표현할 수 있다.

더불어, 레이아웃에 대한 더 많은 정보는 3.장 "사용자 인터페이스"를 보라.

커스텀 레이아웃 리소스(Custom Layout Resources)

여러분은 레이아웃 리소스에서 사용하기 위한 커스텀 엘리먼트를 정의할 수 있다. 그리고 나서 이러한 커스텀 엘리먼트는 임의의 안드로이드 레이아웃 엘리먼트처럼 동일하게 사용될 수 있다. 즉, 여러분은 다른 리소스에서 그것을 사용하고 그것의 애트리뷰트를 지정할 수 있다. ApiDemos 샘플 애플리케이션에는 LabelView라는 커스텀 레이아웃 XML 태그를 생성하는 예제가 있다. 커스텀 엘리먼트를 만들기 위해서, 여러분은 다음에 오는 파일들을 필요로 할 것이다.

- Java 구현 파일 – 그 구현 파일. 이 클래스는 뷰View 또는 서브클래스를 확장해야 한다. ApiDemos에 있는 LabelView.java를 보라.
- res/values/attrs.xml – 클라이언트가 자신의 레이아웃 XML 파일에서 여러분의 오브젝트를 인스턴스화 하기 위해, 그것이 지원하는 XML 엘리먼트와 애트리뷰트를 정의한다. 여러분의 엘리먼트를 〈declare-styleable id=your_java_class_name〉에 정의하라. ApiDemos에 있는 res/layout/attrs.xml을 보라.
- res/layout/your_class.xml [선택사항] – 여러분의 오브젝트의 레이아웃를 기술하기 위한 옵션사항인 XML 파일. 이것은 또한 Java 코드에서 이루어질 수도 있다. ApiDemos에 있는 custom_view_1.xml을 보라.

소스 파일 포맷: 〈?xml〉 선언이 없는 XML 파일, 그리고 〈resources〉 루트 엘리먼트는 하나 또는 그 이상의 커스텀 엘리먼트 태그를 포함한다.
리소스 파일 위치: res/values/attrs.xml (파일 이름은 상관없음)
컴파일된 리소스 데이터타입: 뷰(또는 서브클래스) 리소스에 대한 리소스 포인터.
리소스 레퍼런스 이름: R.styleable.some_file (Java)

스타일과 테마(Styles and Themes)

스타일은 하나의 엘리먼트에 적용되는 하나 또는 그 이상의 애트리뷰트들이다. (예를 들어 10 point red Arial 폰트가 TextView에 적용된) 스타일은 레이아웃 XML 파일 내의 하나의 엘리먼트에 하나의 애트리뷰트로서 적용된다.

테마는 전체 스크린에 적용되는 하나 또는 그 이상의 애트리뷰트이다 – 예를 들어 여러분은 액티비티가 스크린에 떠 있는floating 다이얼로그 박스가 되도록 디자인하기 위해, 안드로이드 Theme.dialog 테마를 스탁stock에 적용할 지 모른다. 테마는 매니페스트 파일에 있는 activity의 애트리뷰트로 지정된다.(역주: 테마매니페스트의 application의 애트리뷰트에도 지정할 수 있다.)

스타일과 테마 둘 다, 하나 또는 그 이상의 문자열 또는 숫자 값(일반적으로 컬러 값), 또는 다른 리소스들(드로어블(drawable) 등등)에 대한 레퍼런스를 포함하는 〈style〉 블락으로 정의된다. 이 엘리먼트는 상속을 지원한다. 그러므로 여러분은 MyBaseTheme, MyBaseTheme.Fancy, MyBaseTheme.Small 등등을 가질 수 있다.

스타일과 테마에 대한 완전한 논의에 대해서는, "스타일과 테마 적용하기" 섹션을 읽어라.

소스 파일 포맷: XML 파일은 〈?xml version="1.0" encoding="utf-8"?〉 선언을 요구하며, 루트 〈resources〉 엘리먼트는 하나 또는 그 이상의 〈style〉 태그를 포함한다.
리소스 소스 파일 위치: res/values/styles.xml (파일 이름은 상관없으나, 표준 사례는 styles.xml 이라 불리는 하나의 파일에 모든 스타일을 넣은 것이다.)
컴파일된 리소스 데이터타입: Java CharSequence 에 대한 리소스 포인터.
리소스 레퍼런스 이름:

- Java: 전체 스타일에 대한 R.style.styleID, 개별적인 설정에 대한 R.style.styleID.itemID
- XML: 전체 스타일에 대한 @[package:]style/styleID, 개별적 아이템에 대한 @[package:]style/styleID/itemID. 노트: 현재 적용된 테마에서 사용된 값을 레퍼런스하기 위해서는 아래의 XML에서 설명되듯이 "@" 대신에 "?"를 사용하라.

구 문

```
<style name=string [parent=string] >
   <item name=string>Hex value | string value | reference</item>+
</style>
```

〈style〉

하나 또는 그 이상의 〈item〉 엘리먼트를 보유하며, 그것들 각각은 하나의 값을 설명하고 있다. 이 값들의 하나의 묶음인 스타일은 테마로써 레퍼런스될 수도 있다.

- name – 이 테마를 가리키는데 사용되는 이름.
- parent – 선택사항인 parent 테마. 지정된 테마의 모든 값들은 이 테마로 상속될 것이다. 여러분이 지정한 동일한 이름을 갖는 임의의 값은 상속한 값을 오버라이드 할 것이다. 그 이름은 그 패키지 수식어를 가져야 한다. 그러나 여러분은 /style 지시어를 필요로 하지 않는다(예를 들어 기본적인 안드로이드 테마에 대해서는 android:Theme, 또는 여러분 패키지에 정의된 하나의 테마에 대해서는 MyTheme).

〈item〉

이 아이템에서 사용되는 값. 그것은 표준 문자열, 16진수 컬러 값, 또는 임의의 다른 리소스 타입에 대한 레퍼런스가 될 수 있다.

스타일과 테마를 선언하고, 적용하는 방법에 대한 예제에 대해서는, "스타일과 테마 적용하기" 섹션을 읽어라.

서처블(Searchable)

시스템 전체의 끊김없는 기능으로써 사용자에게 검색이 보여지도록 하기 위해, 안드로이드 프레임워크는 애플리케이션들이 검색되어지는 방법을 제어하게끔하는 API들을 제공한다. 애플리케이션은 검색이 호출되는 방법, 검색 다이얼로그가 보이는 방법, 그리고 사용자가 타이프할 때 제시되는 추천어suggestions를 포함해서 어떤 유형의 검색 결과들이 유용한지를 커스터마이징할 수 있다.

애플리케이션은 안드로이드 검색 프레임워크를 사용하기 위해 XML 리소스 형식으로 검색 설정configuration을 제공해야 한다. 이 섹션은 검색 설정configuration XML의 구문과 사용에 대해 설명한다. 여러분의 애플리케이션이 검색 기능을 사용하는 방법에 대한 더 완벽한 논의에 대해서는 SearchManager를 보라.

소스 파일 포맷: XML 파일은 〈?xml version="1.0" encoding="utf-8"?〉 선언과 루트 〈searchable〉 엘리먼트를 요구한다.

리소스 소스 파일 위치: res/xml/searchable.xml (파일 이름은 상관없으나, 표준 사례는 searchable.xml을 사용하는 것이다.)

컴파일된 리소스 데이터타입: xml 오브젝트에 대한 리소스 포인터.

리소스 레퍼런스 이름:

- Java: R.xml.filename
- XML: @[package:]xml/filename (예, @xml/searchable).

구 문

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android
      android:label="@string/search_label"  ... >
      <actionkey  android:keycode="KEYCODE_CALL"  ... >
</searchable>
```

〈searchable〉

애플리케이션내에서 수행되는 텍스트와 음성 검색들을 위한 셋팅settings을 포함하는 모든 애플리케이션 검색 설정configurations을 정의하라. 다음과 같은 애트리뷰트들이 허용된다.

- label – 필수임. 이것은 사용자에게 보여지게 될 애플리케이션에 대한 이름이다. 이것은 searchMode가 "showSearchLabelAsBadge"로 지정되었을 때에만 보여질 것이다(아래를 보라).
- hint – 이것은 검색 텍스트 필드에 어떤 텍스트도 입력되지 않았을 때 보여지는 텍스트이다. 이것은 해당 검색이 수행하게 되는 것에 대한 컨텍스트 context를 제공하기 위해 권장된다(예, "사전 검색").
- searchMode – 만약 이것이 제공되고 비어있지 않다면, 이것은 검색 화면의 제어를 위한 추가적인 모드mode를 지정한다. 다음과 같은 모드 mode 값들이 허용된다.
  - showSearchLabelAsBadge – 만약 이것이 지정되면, 해당 검색바 내에 검색 대상target(라벨label)을 보여주는 것을 활성화한다.
  - queryRewriteFromData – 만약 이것이 지정되면, 추천어 suggestion 컬럼column SUGGEST_COLUMN_INTENT_DATA가 추천어 쿼리 정정을 위한 텍스트로써 간주되도록 한다. 이것은 SUGGEST_COLUMN_INTENT_DATA에 있는 그 값들이 사용자

의 검열inspection과 편집editing에 적합할 때에만 사용되어야 한다 – 전형적으로, HTTP/HTTPS Uri.

- queryRewriteFromText – 만약 이것이 지정되면, 추천어suggestion 컬럼column SUGGEST_COLUMN_TEXT_1이 추천어 쿼리 정정을 위한 텍스트로써 간주되도록 한다. 이것은 쿼리 텍스트가 제공되지 않고 SUGGEST_COLUMN_INTENT_DATA 값들이 사용자 검열inspection과 편집editing에 적합하지 않는 경우에 추천어suggestions를 위해 사용되어야 한다.

searchMode에 대한 위의 값들 중 한 개 이상이 한번에 사용될 수 있다. 예를들어, 여러분은 아래와 같이 한번에 두개의 모드를 선언할 수 있다.

```
searchMode="queryRewriteFromData|queryRewriteFromText"
```

- inputType – 만약 이것이 지정되면, 사용자가 입력하게 될 검색 텍스트의 유형에 대한 힌트를 제공한다. 자유로운 형식의 텍스트가 기대되는 대부분의 검색에 대해서는, 이 애트리뷰트는 필요하지 않다. 이 애트리뷰트에 대한 적합한 값들의 목록에 대해서는 inputType을 보라.
- imeOptions – 만약 이것이 지정되면, 입력 방법에 대한 추가적인 옵션을 제공한다. 자유로운 형식의 텍스트가 기대되는 대부분의 검색에 대해서는, 이 애트리뷰트는 필요하지 않으며, 디폴트로 "actionSearch"일 것이다. 이 애트리뷰트에 대한 적합한 값들의 목록에 대해서는 imeOptions을 보라.

만약 여러분이 검색 추천어suggestions를 생성하기 위해 컨텐트 프로바이더를 정의했다면, 여러분은 컨텐트 프로바이더와의 커뮤니케이션을 설정하기 위해 약간 더 많은 searchable 메타데이터를 제공할 필요가 있다. 다음에 나오는 것이 검색 추천어suggestsions를 제공할 때 사용할 추가적인 〈searchable〉 애트리뷰트들이다.

- searchSuggestAuthority – 검색 추천어suggestions를 제공하기 위해 필수임. 이 값은 여러분의 매니페스트의 프로바이더 섹션에서 제공되는 authority(권위) 문자열과 일치해야 한다.
- searchSuggestPath – 만약 이것이 지정되면, 이 경로path는 추천어suggestions 쿼리 Uri에 삽입될 것이며, 그 위치는 여러분이 제공한 authority(권위) 뒤쪽이지만, 표준 추천어suggestions 경로의 앞쪽이다. 이

것은 여러분이 다른 유형의 추천어(예, 다른 데이터 유형에 대해)를 제시하는 하나의 컨텐트 프로바이드를 가질 때에만 필요하며, 여러분은 그것들이 수신될 때 추천어suggestions 쿼리들을 명확하게 하는 방법이 필요하다.

- searchSuggestSelection – 만약 이것이 지정되면, 이 값은 selection 파라미터로써 여러분이 쿼리 함수안으로 전달될 것이다. 전형적으로, 이것은 여러분의 데이터베이스에 대한 WHERE절이 될 것이며, 사용자에 의해 입력되었던 실제 쿼리 문자열을 가리키는 하나의 물음표를 포함할 것이다. 하지만, 여러분은 또한 (selection 아큐먼트를 통해) 쿼리 텍스트의 전달을 간단하게 발생시키는 임의의 공백이 아닌 값을 사용할 수도 있으며, 그런 다음에 여러분의 프로바이더에 (selection 파라미터의 실제 텍스트를 무시하는) 임의의 적절한 방식으로 쿼리 텍스트를 사용하라.
- searchSuggestIntentAction – 사용자가 검색 추천어suggestion를 클릭할 때 사용되는 디폴트 인텐트 액션. 만약 이것이 지정되고 선택된 추천어suggestion에 의해 오버라이드되지overriden 않는다면, 이 값은 사용자가 추천어suggestion를 클릭할 때 인텐트의 액션 필드에 위치하게 될 것이다.
- searchSuggestIntentData – 사용자가 검색 추천어suggestion를 클릭할 때 사용되는 디폴트 인텐트 데이터. 만약 이것이 지정되고 선택된 추천어suggestion에 의해 오버라이드되지overriden 않는다면, 이 값은 사용자가 추천어suggestion를 클릭할 때 인텐트의 데이터 필드에 위치하게 될 것이다.

여러분의 애플리케이션의 로컬 검색을 사용하는 동안 검색 추천어suggestions 제공을 넘어서, 여러분은 또한 Quick Search Box(역주: 홈 런처에 있는 검색창)에서 여러분의 검색 추천어suggestions가 유용하도록 설정할 수 있다. 이것은 사용자가 여러분의 애플리케이션 외부에서 여러분의 애플리케이션 컨텐트로부터 검색 추천어suggestions를 수신하는 것을 가능하게 할 것이다. 다음에 나오는 것이 Quick Search Box에 검색 추천어suggestions를 제공할 때 사용하는 추가적인 〈searchable〉 애트리뷰트들이다.

- includeInGlobalSearch – Quick Search Box에 검색 추천어suggestions를 제공하기 위해 필수임. 만약 참true라면, 이것은 여러분의 애플리케이션에 의해 제공되는 검색 추천어suggestions가 글로벌하게 접근가능한 Quick Search Box에 포함되어야 한다는 것을 가리킨다. 사용자는

Quick Search Box에 여러분의 추천어가 나타나도록 하기 위해 여전히 시스템 검색 셋팅에서 검색가능한 아이템으로써 여러분의 애플리케이션을 활성화해야 한다.

- searchSettingsDescription – 만약 이것이 제공되면, 이것은 여러분이 Quick Search Box에 제공하는 검색 추천어suggestions에 대한 간단한 설명을 제공하며, 그것은 여러분의 애플리케이션에 대한 검색 셋팅란에서 보여지게 될 것이다.
- queryAfterZeroResults – 과거에 빈 결과를 리턴했던 쿼리들의 슈퍼셋들supersets에 대해 소스source가 호출되어야 하는지 여부를 가리킨다. 예를들어, 만약 소스source가 "bo"에 대해 빈 결과를 리턴했다면, 그것은 "bob"에 대해서 무시될 것이다. 만약 이것이 거짓false으로 지정되면, 이 소스source는 하나의 세션에 대해서만 무시될 것이다; 다음번 검색 다이얼로그는 호출되며, 모든 소스source들은 쿼리될 것이다. 디폴트 값은 거짓false이다.
- searchSuggestThreshold – Quick Search Box로부터 소스source 검색lookup을 일으키기 위해 필요한 글자의 최소 갯수를 가리킨다. 임계치threshold보다 더 짧은 어떤 것에 대해서는 소스source가 쿼리되지 않는다는 것을 보장할 뿐이다. 디폴트 값은 0이다.

여러분의 액티비티에 대한 음성 검색을 활성화하기 위해, 여러분은 음성 검색을 활성화하고 설정하는 searchable 메타데이터에 필드들을 추가할 수 있다. 다음에 나오는 것이 음성 검색을 구현할 때 사용하는 추가적인 〈searchable〉 애트리뷰트들이다.

- voiceSearchMode – 음성 검색 능력을 제공하기 위해 필수임. 만약 이것이 제공되고 비워있지 않다면, 음성 검색을 활성화한다. (음성 검색은 디바이스에 의해 제공되지 않을 수도 있다. 이러한 경우에 이 플래그들은 아무런 영향을 주지 않을 것이다.) 다음에 나오는 모드mode 값들이 허용된다.
  - showVoiceSearchButton – 만약 이것이 지정되면, 음성 검색 버튼을 표시한다. 이것은 음성 검색이 디바이스에서 유용한 경우에만 영향을 가진다. 만약 이것이 지정되면, "launchWebSearch" 또는 "launchRecognizer" 또한 지정되어야 한다.
  - launchWebSearch – 만약 이것이 지정되면, 음성 검색 버튼은 해당

사용자를 직접적으로 내장된 음성 웹 검색 액티비티에 이르게 할 것이다. 대부분의 애플리케이션은 이 플래그를 사용하지 않을 것이다. 왜냐하면 그것은 그러한 검색이 호출되었던 액티비티로부터 사용자를 벗어나게 할 것이기 때문이다.

  - launchRecognizer – 만약 이것이 지정되면, 음성 검색 버튼은 해당 사용자를 직접적으로 내장된 음성 녹음 액티비티에 이르게 할 것이다. 이 액티비티는 사용자가 말하도록 요구할 것이며, 말했던 텍스트를 문자로 기록하고, 그리고 마치 사용자가 검색 UI에 그것을 타이프하고 검색 버튼을 클릭한 것처럼, 검색가능한 액티비티에 생성된 쿼리 텍스트를 전달할 것이다.

- voiceLauguageModel – RecognizerIntent로부터의 문자열 상수. 만약 이것이 제공되면, 이것은 음성 인식 시스템에 의해 사용되어야 하는 언어 모델을 지정한다. 더 많은 정보에 대해서는 EXTRA_LANGUAGE_MODEL을 보라. 만약 이것이 제공되지 않으면, 디폴트 값으로 LANGUAGE_MODEL_FREE_FORM이 사용될 것이다.
- voicePromptText – 문자열. 만약 이것이 제공되면, 이것은 음성 입력 동안 보여지게 될 프롬프트를 지정한다. 만약 이것이 제공되지 않으면, 디폴트 프롬프트가 보여지게 될 것이다.
- voiceLanguage – 로케일Locale로부터의 문자열 상수. 만약 이것이 제공되면, 이것은 기대되는 구어spoken language를 지정한다. 이것은 현재의 getDefault()의 값과 다른 경우에만 필요하다.
- voiceMaxResults – 만약 이것이 제공되면, SEARCH 인텐트의 기본 쿼리로써 항상 제공될 "최적의best" 결과를 포함하는 리턴 결과의 최대 개수를 강제한다. 1 또는 그것보다 더 커야 한다. 인텐트로부터 그 결과를 얻기 위해 EXTRA_RESULTS를 사용하라. 만약 이것이 제공되지 않으면, 인식기는 리턴할 많은 결과를 선택할 것이다.

〈actionkey〉

검색 액션에 대한 쇼트컷shortcut 키를 정의한다.

- keycode – 필수임. 이 애트리뷰트는 여러분이 응답하고자 하는 액션 키를 나타낸다. 모든 액션 키들이 실질적으로 이러한 메커니즘을 사용하는 것이 지원되지는 않는다는 것에 주의하라. 그것들 중 많은 것은 타이핑, 네

비게이션, 또는 시스템 기능을 위해 사용된다. 이것은 여러분의 검색가능한 액티비티에 전달되는 ACTION_SEARCH 인텐트에 추가될 것이다. 해당 키코드keycode를 보려면, getIntExtra(SearchManager.ACTION_KEY)를 사용하라. 키코드keycode와 더불어, 여러분은 또한 하나 또는 그 이상의 아래에 있는 액션 지정자specifier 애트리뷰트들을 제공해야 한다는 것에 주의하라.

- queryActionMsg – 만약 여러분이 보통의 검색 쿼리를 입력하는 동안 액션 키를 제어하고자 한다면, 여러분은 여기에 액션 문자열을 정의해야 한다. 이것은 여러분의 검색가능한 액티비티에 전달되는 ACTION_SEARCH 인텐트에 추가될 것이다. 그 문자열을 보려면, getStringExtra(SearchManager.ACTION_MSG)를 사용하라.
- suggestActionMsg – 만약 여러분이 추천어suggestion가 보여지고 선택되는 동안에 액션 키를 제어하고자 한다면, 이것을 제어하는 두 가지 방법이 있다. 만약 여러분의 모든 추천어suggestions가 액션 키를 제어할 수 있다면, 여러분은 간단하게 이 애트리뷰트를 사용하는 액션 메시지를 정의할 수 있다. 이것은 여러분의 검색가능한 액티비티에 전달되는 ACTION_SEARCH 인텐트에 추가될 것이다. 그 문자열을 보려면, getStringExtra(SearchManager.ACTION_MSG)를 사용하라.
- suggestActionMsgColumn – 만약 여러분이 추천어suggestion가 보여지고, 선택되는 동안에 액션 키를 제어하고자 하지만, 모든 추천어suggestion에 대해 이 액션 키를 활성화하길 원하지 않는다면, 여러분은 추천어들suggestion-by-suggestion의 기반에 그것을 제어하기 위해 이 애트리뷰트를 사용할 수 있다. 먼저, 여러분은 액션 문자열을 포함할 여러분의 추천어suggestion가 있는 하나의 컬럼을 정의해야 한다. 그리고 나서 여러분의 컨텐트 프로바이더에서 이 컬럼을 제공해야 한다. 그리고 필요할 때, 이 컬럼에 데이터를 제공하라. 검색 관리자는 하나의 컬럼을 선택하기 위해 여기에서 제공되는 문자열을 사용하는 여러분의 추천어suggestion 커서를 확인할 것이다. 그리고 커서로부터 하나의 문자열을 선택하기 위해 그것을 사용할 것이다. 그 문자열은 여러분의 검색가능한 액티비티에 전달되는 ACTION_SEARCH 인텐트에 추가될 것이다. 그 문자열을 보려면, getStringExtra(SearchManager.ACTION_MSG)를 사용하라. 만약 선택된 추천어suggestion에 대한 데이터가 존재하지 않는다면, 액션 키는 무시될 것이다.

Section I 5

인텐트와 인텐트 필터

5.1. 인텐트 오브젝트

5.2. 인텐트 확정

5. 인텐트와 인텐트 필터

애플리케이션의 핵심 컴포넌트 세 개 – 액티비티, 서비스, 브로드캐스트 리시버 – 는 인텐트intent라고 불리는 메시지를 통해서 활성화된다. 인텐트 메세징은 동일 애플리케이션 내에 존재하는 컴포넌트 또는 다른 애플리케이션 내에 존재하는 컴포넌트들 간의 다소 더딘late(역주: start-up 시점이 아님)런타임 바인딩을 위한 도구이다. 인텐트 자체, 즉 인텐트 오브젝트는 수행되어야 하는 오퍼레이션에 대한 추상적인 설명을 보유하고 있는 데이터 구조이다 – 또는, 브로드캐스트의 경우에는, 발생했고 그리고 공표된 어떤 것에 대한 표현이다. 각각의 컴포넌트 타입에 따라 인텐트를 전달하는 별도의 메커니즘이 존재한다.

- 인텐트 오브젝트는 액티비티를 런치하거나 또는 어떤 것을 새로 수행할 이미 존재하는 액티비티를 얻기 위해서, Context.startActivity() 또는 Activity.startActivityForResult()에 전달된다. (이것은 또한 startActivityForResult()를 호출했던 액티비티에 정보를 반환하기 위해 Activity.setResult()에 전달될 수도 있다.)
- 인텐트 오브젝트는 서비스를 초기화하거나 또는 실행중인 서비스에 새로운 지시를 전달하기 위해, Context.startService()에 전달된다. 유사하게, 인텐트는 호출하는 컴포넌트와 그것의 대상이 되는 서비스간의 커넥션을 확립하기 위해, Context.bindService()에 전달된다. 만약 서비스가 아직 실행되고 있지 않다면, 필요에 따라 서비스를 초기화 할 수 있다.
- Context.sendBroadcast(), Context.sendOrderedBroadcast(), Context.sendStickyBroadcast() 와 같은 브로드캐스트 메쏘드들 중 어떤 것에 전달되는 인텐트 오브젝트는 그것에 관심을 갖는 모든 브로드캐스트 리시버들에게 전달된다. 많은 종류의 브로드캐스트들은 시스템 코드 내에서 만들어진다.

각각의 경우에 있어서, 안드로이드 시스템은 인텐트에 응답할 적절한 액티비티, 서비스, 또는 브로드캐스트 리시버 집단을 찾으며, 필요하다면 그것들을 인스턴스화한다. 이러한 메세징 시스템 내에는 어떠한 오버랩도 존재하지 않는다; 브로드

캐스트 인텐트는 브로드캐스트 리시버에게만 전달되며, 결코 액티비티나 서비스에는 전달되지 않는다. startActivity()에 전달된 인텐트는 오직 하나의 액티비티에만 전달되며 결코 서비스나 브로드캐스트 리시버, 기타 등등에게는 전달되지 않는다.

이 문서는 인텐트 오브젝트에 대한 설명으로 시작된다. 그것은 컴포넌트에 인텐트를 맵핑mapping하기 위해 안드로이드가 사용하는 규칙, 즉 어떤 컴포넌트가 인텐트 메시지를 수신해야 하는 지를 결정하는 방법을 설명한다. 인텐트가 전달되어야 하는 대상이 되는 컴포넌트를 명시적으로 지명하지 않는 경우에, 이 처리 과정은 잠재적인 대상들과 관련되어 있는 인텐트 필터에 대한 인텐트 오브젝트 테스트를 포함하게 된다.

인텐트 오브젝트

인텐트Intent 오브젝트는 하나의 정보 묶음이다. 그것은 (수행되어야 하는 액션과 그곳에서 처리해야 하는 데이터와 같이) 인텐트를 수신하게 되는 컴포넌트가 관심을 갖는 정보와 (인텐트를 제어해야 하는 컴포넌트의 카테고리와, 대상이 되는 액티비티를 런치하는 방법에 대한 지시와 같은) 안드로이드 시스템에 대한 추가적인 정보를 포함하고 있다. 기본적으로 그것은 다음과 같은 내용을 포함할 수 있다.

컴포넌트 이름(Component name)

인텐트를 제어해야 하는 컴포넌트의 이름. 이 필드는 ComponentName 오브젝트 – 타겟 컴포넌트 (예를 들어 "com.example.project.app.FreneticActivity")의 전체 클래스 이름과 컴포넌트가 (예를 들어 "com.example.project") 위치하는 곳으로 애플리케이션의 매니페스트 파일에서 설정된 패키지 이름의 조합이다. 컴포넌트 이름의 패키지 영역과 매니페스트에 설정된 패키지 이름은 반드시 일치할 필요는 없다.

컴포넌트 이름은 선택사항이다. 만약 그것이 지정되면, 인텐트 오브젝트는 지정된 클래스의 인스턴스에 전달된다. 만약 그것이 지정되지 않는다면, 안드로이드는 적당한 타겟을 찾아내기 위해 인텐트 오브젝트 내의 다른 정보를 이용한다 – 이 문서 다음에 나오는 "인텐트 확정" 섹션을 보라

컴포넌트 이름은 setComponent(), setClass(), 또는 setClassName()에 의

해 설정되고 getComponent()에 의해 읽어진다.

액션(Action)

수행되어야 하는 액션을 지칭하는 문자열 – 또는 브로드캐스트 인텐트의 경우에는, 얻게 되고 리포트 될 액션이다. 인텐트 클래스는 아래에 리스트된 것처럼, 많은 액션 상수들을 정의하고 있다.

상수	타겟 컴포넌트	액션
ACTION_CALL	액티비티	전화를 걸어라.
ACTION_EDIT	액티비티	사용자에게 편집할 데이터를 표시하라.
ACTION_MAIN	액티비티	데이터 입력과 반환 결과 없이 태스크의 최초의 액티비티로써 액티비티를 실행하라.
ACTION_SYNC	액티비티	모바일 디바이스의 데이터와 서버의 데이터를 동기화하라.
ACTION_BATTERY_LOW	브로드캐스트 리시버	배터리가 부족하다는 경고.
ACTION_HEADSET_PLUG	브로드캐스트 리시버	헤드셋이 디바이스에 연결 또는 분리되었다는 것.
ACTION_SCREEN_ON	브로드캐스트 리시버	스크린이 켜졌다는 것.
ACTION_TIMEZONE_CHANGED	브로드캐스트 리시버	타임존 설정이 바뀌었다는 것.

일반적인 액션들에 대한 미리 정의된 상수들의 리스트에 대해서는 인텐트 클래스에 대한 설명을 보라. 다른 액션들은 안드로이드 API의 다른 곳에서 정의된다. 여러분은 또한 여러분의 애플리케이션에 있는 컴포넌트를 활성화하기 위해 자기만의 액션 문자열들을 정의할 수 있다. 여러분이 만든 이런 것들은, 예를 들어 "com.example.project.SHOW_COLOR" 와 같이 접두어로서 애플리케이션 패키지를 포함해야 한다.

액션은 메쏘드 명이 매개변수들의 집합과 리턴 값을 결정하듯이, 인텐트의 나머지 부분 – 특히, 데이터data와 엑스트라extras 필드 – 이 어떻게 구성되는지를 크기 결정한다. 이 이유 때문에, 가능한 한 상세하게 액션 이름을 사용하고, 그것들을 인텐트의 나머지 필드에 견고하게 결합하는 것이 좋다. 달리 표현해서, 독립된 형태로 액션을 정의하는 것보다 여러분의 컴포넌트가 제어할 수 있는 인텐트 오브젝트에 대한 전체 프로토콜을 정의하라.

인텐트 오브젝트의 액션은 setAction() 메쏘드에 의해 설정되고, getAction()에 의해 읽어진다.

데이터(Data)

처리되어야 하는 데이터에 대한 URI와 데이터의 MIME 타입. 서로 다른 액션들은 다른 종류의 데이터 명세와 한 조를 이룬다. 예를 들어 만일 액션 필드가 ACTION_EDIT라면, 데이터 필드는 편집을 위해 보여지게 될 문서의 URI를 포함할 것이다. 만일 액션이 ACTION_CALL 이라면, 데이터 필드는 통화해야 할 번호를 가진 tel: URI가 될 것이다, 비슷하게, 만일 액션이 ACTION_VIEW이고 데이터 필드가 http: URI라면, 그것을 수신하는 액티비티는 다운로드를 수행할 것이고, URI가 가리키는 데이터를 보여줄 것이다.

인텐트를 그 데이터를 처리할 능력이 있는 컴포넌트에 매칭할 때, 가끔 그 URI에 추가해서 그 데이터의 타입(그것의 MIME 타입)을 아는 것이 중요하다. 예를 들어 이미지 데이터를 보여줄 수 있는 컴포넌트가 오디오 파일을 재생하기 위해서 호출되어서는 안된다.

많은 경우에 있어서, 데이터 타입은 URI로부터 추측될 수 있다 – 특히 디바이스상에 존재하는 데이터를 가리키고 컨텐트 프로바이더(7장. "컨텐트 프로바이더"를 보라)에 의해 제어되는 content: URI들이 그렇다. 하지만 데이터 타입 또한 명시적으로 인텐트 오브젝트 내에 지정될 수도 있다. setData() 메쏘드는 URI만 사용해서 데이터를 지정하며, setType()은 MIME 타입으로만 그것을 지정하며, 그리고 setDataAndType()은 URI와 MIME 타입 둘 다에 의해 그것을 지정한다. URI는 getData()에 의해 읽혀지게 되고, 타입은 getType()에 의해 읽혀진다.

카테고리(Category)

인텐트를 제어해야 하는 컴포넌트의 종류에 대한 추가적인 정보를 포함하는 문자열. 인텐트 오브젝트 내에는 임의의 수의 카테고리들이 있을 수 있다. 액션에 대해서 그러했듯이, 인텐트 클래스는 다음과 같은 것을 포함하는 몇 가지의 카테고리 상수를 정의하고 있다.

상 수	의미하는 바
CATEGORY_BROWSABLE	타겟 액티비티는 링크에 의해 참조되는 데이터(예를 들어 이미지나 이메일 메시지)를 보여주기 위해 브라우저에 의해 안전하게 호출될 수 있다.
CATEGORY_GADGET	액티비티는 가젯들을 보유할 수 있는 다른 액티비티 내에 임베딩될수 있다.
CATEGORY_HOME	액티비티는 홈 화면을 보여준다. 디바이스가 켜질 때나 HOME 키가 눌렸을 때, 사용자가 보게되는 첫 번째 화면.
CATEGORY_LAUNCHER	액티비티는 하나의 태스크에서 최초의 액티비티가 될 수 있으며, 최상위 계층의 애플리케이션 런처에 리스트된다.
CATEGORY_PREFERENCE	타겟 액티비티는 설정 패널이다.

전체 카테고리 리스트에 대해서는 인텐트 클래스 설명을 보라.

addCategory() 메쏘드는 인텐트 오브젝트에 카테고리를 추가한다. remove Category()는 이전에 추가된 카테고리를 삭제하며, getCategories()는 해당 오브젝트 안에 있는 현재의 모든 카테고리들을 얻는다.

엑스트라(Extras)

추가적인 정보에 대한 키-값 쌍key-value pairs이 인텐트를 제어하는 컴포넌트에 전달되기도 한다. 약간의 액션들이 특별한 종류의 데이터 URI들과 한 조를 이루듯이, 약간의 것들은 특정한 엑스트라들과 한 조를 이룬다. 예를 들어 ACTION_TIMEZONE_CHANGED 인텐트는 새로운 시간대를 나타내는 "time-zone" 엑스트라를 가지며, ACTION_HEADSET_PLUG는 헤드셋의 타입에 대한 "name" 엑스트라뿐만 아니라, 헤드셋이 방금 연결되었는지 아니면 제거되었는지를 가리키는 "state" 엑스트라를 가진다. 만약 여러분이 SHOW_COLOR 액션을 만들었다면, 컬러 값이 엑스트라 키-값 쌍key-value pairs안에 지정될 수도 있다.

인텐트 오브젝트에는 엑스트라 데이터의 다양한 타입을 추가하는 일련의 put...() 메쏘드를 가지며, 해당 데이터를 읽어오기 위한 일련의 get...() 메쏘드를 가진다. 그러한 메쏘드들은 번들 오브젝트들에 대한 메쏘드들과 함께 한다. 실제로, 그러한 엑스트라는 putExtras()와 getExtras() 메쏘드들을 사용하여 하나의 번들로써 설치되거나 읽혀질 수도 있다.

플래그(Flags)

다양한 종류의 플래그. 많은 것들이 안드로이드 시스템에 의한 액티비티 런치 방법(예를 들어 액티비티가 어떤 태스크에 속해야 하는지)과 런치 후 액티비티를 다루는 방법(예를 들어 최근 액티비티 리스트에 그것이 속하는 지의 여부)을 지시한다. 이러한 모든 플래그들은 인텐트 클래스에서 정의된다.

안드로이드 시스템과 안드로이드 플랫폼의 애플리케이션들 모두는 시스템에서 발생된 브로드캐스트를 발송하고, 시스템이 정의한 컴포넌트를 활성화 하기 위해 인텐트 오브젝트를 사용한다. 시스템 컴포넌트를 활성화하기 위한 인텐트를 구조화하는 방법을 보기 위해서는 레퍼런스에 있는 인텐트 목록을 참고하라.

인텐트 확정(Resolution)

인텐트는 두 그룹으로 나눠질 수 있다.

- **명시적인(Explicit) 인텐트**는 타겟 컴포넌트를 그것의 이름(앞서 언급되었던 컴포넌트 이름 필드에 값이 설정되어 있다)으로 지정한다. 컴포넌트 이름은 일반적으로 다른 애플리케이션의 개발자에게 알려질 수 없기 때문에, 명시적인 인텐트는 전형적으로 (액티비티가 자신에 부수적인 서비스를 실행시키거나, 또는 동격의 액티비티를 실행시키는 것과 같이) 애플리케이션 내부 메시지들을 위해 사용된다.
- **암시적인(Implicit) 인텐트**는 타겟 이름을 지정하지 않는다(컴포넌트 이름 필드는 비어있다). 암시적인 인텐트는 다른 애플리케이션의 컴포넌트를 활성화하기 위해 자주 사용된다.

안드로이드는 지정된 타겟 클래스의 인스턴스에 명시적explicit 인텐트를 전달한다. 인텐트 오브젝트에서 그 인텐트를 얻어야 하는 컴포넌트를 결정함에 있어서, 컴포넌트 이름보다 더 중요한 것은 아무것도 없다.

하나의 다른 전략이 암시적Implicit 인텐트를 위해서 필요하다. 지정된 타겟이 없을 경우에, 안드로이드 시스템은 해당 인텐트를 처리할 최상의 컴포넌트(혹은 컴포넌트들), 즉 요청된 액션을 수행할 하나의 액티비티 혹은 서비스, 또는 브로드캐스트 통보에 응답할 보로드캐스트 리시버들의 집합을 찾아야만 한다. 이것은 인텐트를 잠재적으로 수신할 수 있는 컴포넌트들과 결합된 구조인 인텐트 필터와 인텐트 오

브젝트 내의 내용들을 비교하는 것에 의해 이루어진다. 필터들은 컴포넌트의 기능을 알리며 처리할 수 있는 인텐트의 범위를 정한다. 이것들은 해당 컴포넌트에게 자신이 제시한 유형과 연관된 암시적Implicit 인텐트를 수신할 수 있는 가능성을 열어준다. 만약 컴포넌트가 어떤 인텐트 필터도 가지고 있지 않다면, 그것은 명시적 인텐트만을 받을 수 있다. 필터를 가진 컴포넌트는 명시적거나 암시적인 인텐트 모두를 수신할 수 있다.

오브젝트가 인텐트 필터에 대한 테스트가 이루어질 때, 인텐트 오브젝트의 세 가지 측면이 검토된다.

action
data (URI와 데이터 타입 두가지 모두)
category

엑스트라와 플래그는 어떤 컴포넌트가 인텐트를 수신할 것인가를 결정하는데 있어서 아무런 역할도 하지 않는다.

인텐트 필터(Intent filters)

액티비티, 서비스, 그리고 브로드캐스트 리시버는 시스템에 그들이 처리할 수 있는 암시적implicit 인텐트들이 어떤 것인가를 알려주기 위해, 하나 이상의 인텐트 필터를 가질 수 있다. 각각의 필터는 컴포넌트의 처리능력, 즉 컴포넌트가 수신하고자 하는 인텐트들의 집합을 기술한다. 그 각 필터는 실제로 원하지 않았던 인텐트를 배제함으로써, 필요한 타입의 인텐트들만을 걸러낸다. 하지만 그들이 배제한 것은 단지 타겟 클래스 이름을 지칭하지 않은 암시적implicit 인텐트일 뿐이다. 명시적explicit 인텐트는 그것이 무엇을 포함하든 상관없이 항상 그것의 타겟에 전달된다. 이 경우 필터는 고려되지 않는다. 그러나 암시적implicit 인텐트는 컴포넌트 필터 중 한 개를 통과할 수 있다면, 해당 컴포넌트에게 전달되어진다.

컴포넌트는 그것들이 할 수 있는 각각의 작업, 즉 사용자에게 그것이 제시할 수 있는 각각의 모습 별로 별도의 필터를 소유할 수 있다. 예를 들어 샘플 노트패드NotePad 애플리케이션의 NoteEditor 액티비티는 두 개의 필터를 가진다. 사용자가 보거나 편집할 수 있는 특정 노트를 가지고 시작되는 것 하나와 사용자가 그 안에 기입하고 저장할 수 있는 신규 또는 빈 노트를 가지고 시작되는 다른 하나가 있

다. (모든 Note Pad 필터는 나중에 Note Pad 예제 섹션에서 설명된다.)

인텐트 필터는 IntentFilter 클래스에 대한 하나의 인스턴스이다. 하지만 안드로이드 시스템은 해당 컴포넌트를 실행하기 전에 컴포넌트의 기능에 대해 알아야 하기 때문에, 인텐트 필터는 일반적으로 Java 코드 내부가 아닌, 애플리케이션의 매니페스트 파일인 AndroidManifest.xml의 〈intent-filter〉 애트리뷰트를 사용해서 정의된다(브로드캐스트 리시버에 한하여, Context.registerReceiver()의 호출을 통해 동적으로 등록되는 필터가 하나의 예외이다; 그것들은 IntentFilter 오브젝트로 직접 생성된다).

필터와 보안(Filters and security)

인텐트 필터는 보안에 의존적일 수는 없다. 그것은 특정 종류의 암시적implicit 인텐트를 수신하는 컴포넌트를 오픈하지만, 명시적(explicit) 인텐트가 그 컴포넌트를 타겟팅하는 것을 차단하기 위해 아무 것도 하지 않는다. 비록 필터가 한 컴포넌트가 처리하도록 요청받는 인텐트들을 특정 액션들과 데이터 소스들로 제한하여도 누군가는 데이터 소스를 가지는 명시적 인텐트를 같이 넣을 수 있고 그 컴포넌트를 타겟으로써 지정할 수 있다.

필터는 인텐트 오브젝트의 액션, 데이터, 카테고리를 나열하는 필드들을 가진다. 암시적implicit 인텐트는 세 개의 모든 영역의 필터들에 대해 테스트된다. 그 필터를 소유하고 있는 컴포넌트에 인텐트가 전달되게 하기 위해서는, 그것은 세 개의 모든 테스트가 통과되어야 한다. 만약 그것들 중 하나라도 실패한다면, 안드로이드 시스템은 그 컴포넌트에 인텐트를 전달하지 않을 것이다. 적어도 그 필터의 기준하에서는 안된다. 그러나 컴포넌트는 복수의 인텐트 필터를 가질 수 있기 때문에, 컴포넌트의 필터들 중 하나를 통과하지 못했던 인텐트가 또 다른 필터를 통과할 수도 있다.

세 개의 테스트 각각이 아래에서 상세하게 설명된다.

액션 테스트(Action test)

매니페스트 파일의 〈intent-filter〉 엘리먼트는 〈action〉 서브엘리먼트로 액션들을 리스트한다. 예를 들어 아래와 같다.

```
<intent-filter . . . >
    <action android:name="com.example.project.SHOW_CURRENT" />
    <action android:name="com.example.project.SHOW_RECENT" />
    <action android:name="com.example.project.SHOW_PENDING" />
    . . .
</intent-filter>
```

예제에서 보이듯이, 인텐트 오브젝트는 단지 하나의 액션을 지정하지만, 필터는 하나 이상의 액션을 지정할 수도 있다. 액션 리스트는 비어 있어서는 안된다. 필터는 적어도 하나의 〈action〉 엘리먼트를 포함해야 한다. 그렇지 않다면, 그것은 모든 인텐트들을 차단할 것이다.

이 테스트를 통과하기 위해 인텐트 오브젝트 내에 명시된 액션은, 필터에 나열된 액션들 중의 하나와 일치해야 한다. 만약 오브젝트 또는 필터가 액션을 명시하지 않았다면, 그 결과는 아래와 같다.

- 만일 필터가 어떤 액션도 나열하지 않았다면, 인텐트에 일치할 수 있는 어떤 것도 없다. 그러므로 모든 인텐트는 테스트에 실패한다. 어떤 인텐트도 해당 필터를 통해 얻을 수 없다.
- 반면에, 액션을 명시하지 않은 인텐트 오브젝트는 자동으로 테스트를 통과한다 – 그 필터가 적어도 하나의 액션을 포함하고 있는 한 그렇다. (역주 : 액션을 명시하지 않은 인텐트 오브젝트 역시 필터 테스트를 통과하지 못하는 것으로 확인되었으나, 추후 좀 더 상세한 검토가 필요한 듯하다.)

카테고리 테스트(Category test)

〈intent-filter〉 엘리먼트는 또한 서브엘리먼트로 카테고리들을 리스트 한다. 예를 들어 아래와 같다.

```
<intent-filter . . . >
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    . . .
</intent-filter>
```

이전에 설명되었던, 액션과 카테고리에 대한 상수들은 매니페스트 파일에서는 사용되지 않는 다는 것에 유의하라. 대신 전체 문자열 값이 사용된다. 예를 들어 위

에 예제에 있는 “android.intent.category.BROWSABLE” 스트링은 앞서 이 문서에서 언급되었던 CATEGORY_BROWSABLE 상수에 대응한다. 비슷하게 “android.intent.action.EDIT”는 문자열은 ACTION_EDIT에 대응한다.

인텐트가 카테고리 테스트를 통과하기 위해서는 인텐트 오브젝트 안의 모든 카테고리가 필터에 있는 카테고리와 일치해야 한다. 필터는 추가적으로 카테고리들을 나열할 수 있지만, 그것은 인텐트 안에 있는 그 어떤 것도 생략할 수는 없다.

그러므로 원칙적으로, 카테고리를 가지지 않은 인텐트 오브젝트는 필터에 무엇이 있든 관계 없이 항상 이 테스트을 통과해야 한다. 그것은 대체로 사실이다. 하지만 하나의 예외가 있는데, 그것은 안드로이드가 startActivity()에 전달되는 모든 암시적implicit 인텐트가 마치 적어도 하나의 카테고리인 “android.intent.category.DEFAULT”(CATEGORY_DEFAULT 상수)를 포함하고 있는 것처럼 취급된다는 것이다. 그러므로 암시적implicit 인텐트를 수신하고자 하는 액티비티는 그들의 인텐트 필터에서 “android.intent.category.DEFAULT”를 포함해야 한다(단, “android.intent.action.MAIN”과 “android.intent.category.LAUNCHER”가 설정된 필터는 예외다. 이것들은 액티비티를 신규 태스크로 시작시키고 런처 스크린에 표시되도록 마크한다. 그것들은 카테고리 리스트에 “android.intent.category.DEFAULT”를 포함할 수는 있지만, 필요치는 않는다). 이 필터에 대한 상세한 것은 이후 나오는 “인텐트 매칭 사용하기”를 나중에 보라.

데이터 테스트(Data test)

액션과 카테고리와 같이, 인텐트 필터에 대한 데이터 명세가 서브엘리먼트에 포함된다. 그리고 그런 경우와 같이 서브엘리먼트는 여러 번 나타날 수도 있고, 전혀 나타나지 않을 수도 있다. 예를 들어 아래와 같다.

```
<intent-filter . . . >
    <data android:type=”video/mpeg” android:scheme=”http” . . . />
    <data android:type=”audio/mpeg” android:scheme=”http” . . . />
    . . .
</intent-filter>
```

각각의 〈data〉 엘리먼트는 URI와 데이터 타입(MIME 미디어 타입)을 지정할 수 있다. 그곳에는 URI의 각 영역에 대한 별도의 애트리뷰트인 schema(스키마), host(호스트), port(포트), 그리고 path(경로)가 있다.

```
scheme://host:port/path
```

예를 들어 아래의 URI에서

```
content://com.example.project:200/folder/subfolder/etc
```

scheme은 “content”이고, host는 “com.example.project”이며, port는 “20Q”그리고 path는 “folder/subfolder/etc” 이다. host와 port는 함께 URI authority(당국)을 구성한다. 만약 host가 지정되지 않는다면, port는 무시된다.

이러한 각각의 애트리뷰트는 선택사항이다. 그러나 그것들은 서로 독립적이지는 않다. Authority가 의미를 가지기 위해서는 scheme 또한 명시되어야 한다. path가 의미를 가지기 위해서는 sheme과 authority 두 가지 모두가 명시되어야 한다.

인텐트 오브젝트에 있는 URI가 필터에 있는 URI 명세와 비교될 때, 그것은 실제로 필터에서 언급된 URI의 부분들에 대해서만 비교된다. 예를 들어 만약 필터가 scheme만을 명시한다면, 해당 scheme를 보유한 모든 URI는 필터와 일치한다. 만약 필터가 path없이 scheme과 authority를 명시한다면, 그들의 path와 관계없이, 같은 scheme과 authority를 가지는 모든 URI가 일치한다. 만약 필터가 scheme, authority, path를 명시한다면, 같은 scheme, authority, 그리고 path를 가지는 URI가 일치한다. 그러나 필터에서 path 명시는 path의 부분적 일치만을 요구하기 위해 와일드카드(*)를 포함시킬 수 있다.

〈data〉 엘리먼트 type 애트리뷰트는 데이터의 MIME 타입을 명시한다. 그것은 필터에서 URI보다는 보다 보편적이다. 인텐트 오브젝트와 필터는 둘 다 하위타입 필드에 대해서 “*” 와일드카드를 사용할 수 있다. 예를 들어 어떤 하위타입이든 일치하는 것을 지칭하는 “text/*” 또는 “audio/*” 를 사용할 수 있다.

데이터 테스트는 인텐트 오브젝트 내에 있는 URI와 데이터 타입을 필터에 명시된 URI와 데이터 타입과 비교한다. 규칙은 다음과 같다:

A. URI와 데이터 타입 모두를 포함하고 있지 않은 인텐트 오브젝트는, 필터가 마찬가지로 어떤 URI와 데이터 타입도 명시하지 않는 경우에 한하여 테스트를 통과한다.

B. URI를 포함하고 있지만 데이터 타입을 포함하지 않은 (그리고 URI로부터 데이터 타입이 추측될 수 없는) 인텐트 오브젝트는 그 URI가 필터의 URI와 일치하고, 필터가 마찬가지로 타입을 명시하지 않는 경우에 한하여 테스트를 통과한다. 이것은 실질적인 데이터를 참조하지 않는 mailto: 와 tel: 같은 URI에 대한 경우일 것이다.

C. 데이터 타입을 포함하지만, URI를 포함하지 않은 인텐트 오브젝트는 필터가 동일한 데이터 타입을 리스트하고 있고, 비슷하게 URI를 명시하지 않는 경우에 한하여 테스트를 통과한다.

D. URI와 데이터 타입 모두를 포함하거나 데이터 타입이 없더라도 그것이 URI로부터 추측될 수 있는 경우의 인텐트 오브젝트는, 그것의 type이 필터에 나열된 type과 일치하는 경우에 한하여 데이터 타입 영역에 대한 테스트를 통과한다. 그리고 만약 인텐트의 URI가 필터 안의 URI와 일치하거나, 그것이 content: 또는 file: 형식의 URI이고 필터가 URI를 명시하고 있지 않는 경우에 한하여 URI 영역에 대한 테스트를 통과한다. 바꿔 말하면, 그것의 필터가 단지 하나의 데이터 타입을 나열하고 있다면, 컴포넌트는 content: 과 file: 데이터를 지원하는 것으로 간주된다.

만약 인텐트가 하나 이상의 액티비티 혹은 서비스의 필터를 통과할 수 있다면, 사용자에게 어떤 컴포넌트가 활성화되어야 하는지 물어지게 된다. 만약 어떤 타겟도 발견될 수 없다면 예외exception가 발생한다.

일반적 경우(Common cases)

위에서 보여진 데이터 테스트 대한 마지막 규칙인 규칙 D는 컴포넌트들이 파일 또는 컨텐트 프로바이더로부터 로컬 데이터를 얻을 수 있다는 기대를 반영한다. 그러므로 그 컴포넌트들의 필터는 단지 데이터 타입을 리스트할 수 있기 때문에 명시적으로 content: 와 file: 스키마scheme 이름을 필요로 하지 않는다. 이것은 전형적인 경우다. 예를 들어 다음과 같은 〈data〉 엘리먼트는 그 컴포넌트가 컨텐트 프로바이더로부터 이미지 데이터를 얻을 수 있고 그것을 표시할 수 있다는 것을 안드로이드에게 알려준다:

```
<data android:type="image/*" />
```

대부분의 사용가능한 데이터가 컨텐트 프로바이더에 의해 분배되기 때문에, URI

이외에 데이터 타입을 지정하는 필터는 아마 가장 보편적이다.

또 다른 보편적인 설정은 스키마scheme와 데이터 타입을 가지는 필터이다. 예를 들어 다음과 같은 〈data〉 엘리먼트는 컴포넌트가 네트워크로부터 비디오 데이터를 얻고 표시할 수 있다는 것을 안드로이드에게 말한다:

```
<data android:scheme="http" android:type="video/*" />
```

예를 들어 사용자가 웹 페이지의 하나의 링크를 따라갈 때, 브라우저 애플리케이션이 무엇을 하는지 생각해보라. 그것은 먼저 데이터를 표시하려고 시도한다(만약 링크가 HTML 페이지라면 그것이 할 수 있듯이). 만약 그것이 데이터를 표시할 수 없다면, 그것은 암시적implicit 인텐트에 스키마scheme와 데이터 타입을 함께 넣은 다음 그 작업을 할 수 있는 액티비티를 시작시키려고 시도할 것이다. 만약 거기에 인텐트를 받을 액티비티가 없다면, 그것은 다운로드 매니저에게 데이터를 다운로드할 것을 요청할 것이다. 그것은 컨텐트 프로바이더의 제어 하에 놓인다. 그러므로 잠재적으로 더 많은 액티비티들(단지 데이터 타입만을 명시하는 필터를 가진 것들)의 풀pool이 응답할 수 있다.

대부분의 애플리케이션은 또한 어떤 특정 데이터를 참조하지 않고 새롭게 시작하는 방법을 가진다. 애플리케이션을 초기화할 수 있는 액티비티들은 액션으로 "android.intent.action.MAIN"이 지정된 필터를 가진다. 만약 그들이 애플리케이션 런처에서 보여지게 된다면, 그들은 또한 "android.intent.category.LAUNCHER" 카테고리를 지정하고 있다.

```
<intent-filter . . . >
    <action android:name="code android.intent.action.MAIN" />
    <category android:name="code android.intent.category.LAUNCHER" />
</intent-filter>
```

인텐트 매칭 사용하기(Using intent matching)

인텐트는 활성화할 타겟 컴포넌트를 찾기 위해서 뿐만아니라, 또한 디바이스 상의 컴포넌트의 집합에 관한 어떤 것을 찾기 위해 인텐트 필터와 일치되어야 한다. 예를 들어 안드로이드 시스템은 "android.intent.action.MAIN" 액션과 "android.intent.category.LAUNCHER" 카테고리(이전 섹션에서 설명된 것과 같이)를 지정하는 인텐트 필터를 가진 모든 액티비티를 찾아냄으로써, 사용자가 실행시킬

수 있는 애플리케이션을 보여주는 최상위 스크린인 애플리케이션 런처를 구성한다. 그런 다음 시스템에 그 액티비티들의 아이콘과 라벨을 표시한다. 비슷하게, 그 시스템은 필터 안에 "android.intent.category.HOME"을 가지는 액티비티를 찾음으로써 홈스크린을 발견한다.

여러분의 애플리케이션은 비슷한 방법으로 일치하는 인텐트를 사용할 수 있다. 패키지 매니저PackageManager는 특정 인텐트를 수용할 수 있는 모든 컴포넌트들을 리턴해 주는 query...() 메쏘드 집합과 인텐트에 응답할 최고의 컴포넌트를 결정하는 유사한 일련의 resolve...() 메쏘드들이 있다. 예를 들어 queryIntentActivities()는 아규먼트로 전달된 인텐트를 수행할 수 있는 모든 액티비티 리스트를 리턴하며, queryIntentServices()는 유사한 서비스 리스트를 리턴한다. 위의 모든 메쏘드들은 컴포넌트를 활성화시키지는 않는다. 그것들은 다만 응답할 수 있는 컴포넌트들을 리스트할 뿐이다. 비슷한 방법으로 브로드캐스트 리시버에 대한 queryBroadcastReceivers()가 있다.

노트 패트 예제(Note Pad Example)

노트 패드 샘플 애플리케이션은 사용자가 노트 목록을 열람하고, 목록에 있는 개별 아이템을 상세히 보고, 아이템을 수정하고, 그리고 목록에 신규 아이템을 추가할 수 있게 한다. 이 섹션은 그것의 매니페스트 파일에 선언된 인텐트 필터들을 살펴본다. (만약 여러분이 SDK에서 오프라인 작업을 하고 있다면, 여러분은 〈sdk〉/samples/NotePad/index.html에서 그것의 매니페스트 파일을 포함해서 샘플 애플리케이션에 대한 모든 소스 파일들을 발견할 수 있다. 만약 여러분이 문서를 온라인으로 보고 있다면, 여기 있는 튜토리얼과 샘플 코드 섹션에 소스파일들이 있다.)

그것의 매니페스트 파일에서, 노트 패드 애플리케이션을 세 개의 액티비티를 선언하고 있으며, 그것들 각각은 적어도 하나의 인텐트 필터를 가지고 있다. 노트 데이트를 관리하는 컨텐트 프로바이더 또한 선언되어 있다. 여기에 전체 그대로의 매니페스트 파일이 있다.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.android.notepad">
  <application android:icon="@drawable/app_notes"
               android:label="@string/app_name" >
    <provider android:name="NotePadProvider"
              android:authorities="com.google.provider.NotePad" />
    <activity android:name="NotesList"
```

```
                android:label="@string/title_notes_list">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <action android:name="android.intent.action.EDIT" />
            <action android:name="android.intent.action.PICK" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType
                 ="vnd.android.cursor.dir/vnd.google.note" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.GET_CONTENT" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType
                 ="vnd.android.cursor.item/vnd.google.note" />
        </intent-filter>
    </activity>
    <activity android:name="NoteEditor"
              android:theme="@android:style/Theme.Light"
              android:label="@string/title_note" >
        <intent-filter android:label="@string/resolve_edit">
            <action android:name="android.intent.action.VIEW" />
            <action android:name="android.intent.action.EDIT" />
            <action android:name
                 ="com.android.notepad.action.EDIT_NOTE" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType
                 ="vnd.android.cursor.item/vnd.google.note" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.INSERT" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType
                 ="vnd.android.cursor.dir/vnd.google.note" />
        </intent-filter>
    </activity>
    <activity android:name="TitleEditor"
              android:label="@string/title_edit_title"
              android:theme="@android:style/Theme.Dialog">
        <intent-filter android:label="@string/resolve_title">
            <action android:name
                 ="com.android.notepad.action.EDIT_TITLE" />
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name
                 ="android.intent.category.ALTERNATIVE" />
            <category android:name
                 ="android.intent.category.SELECTED_ALTERNATIVE" />
            <data android:mimeType
                 ="vnd.android.cursor.item/vnd.google.note" />
        </intent-filter>
    </activity>

  </application>
</manifest>
```

첫번째 액티비티인 NotesList는 단일 노트가 아닌 노트 디렉토리(노트 목록)를 조작한다는 점에서 다른 액티비티와 구분된다. 그것은 일반적으로 애플리케이션에 대한 초기 사용자 인터페이스를 제공할 것이다. 그것의 세 개의 인텐트 필터에 의해 설명되고 있듯이 세 가지 것을 할 수 있다.

1.

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

이 필터는 노트 패드 애플리케이션에 대한 메인main 시작 점을 선언한다. 표준 MAIN 액션은 인텐트 내에 어떤 다른 정보도 요구하지 않으며(예를들어, 데이터 명세가 없음), LAUNCHER 카테고리는 이 시작점이 애플리케이션 런처안에 나열되어야 한다는 것을 말한다.

2.

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.PICK" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
```

이 필터는 노트 디렉토리에서 액티비티가 할 수 있는 것들을 선언한다. 그것은 사용자에게 (VIEW와 EDIT 액션을 통해) 디렉토리를 보거나 수정할 수 있게 하며, 또한 (PICK 액션을 통해) 그 디렉토리에서 특정 노트를 가져올 수 있게 한다.

〈data〉 엘리먼트의 mimeType 애트리뷰트는 이러한 액션들이 처리하는 데이터의 종류를 지정한다. 그것은 해당 액티비티가 노트 패드 데이터(vnd.google.note)를 보유하고 있는 컨텐트 프로바이더로부터 0개 또는 그 이상의 아이템들(vnd.android.cursor.dir)을 담고있는 커서를 얻을 수 있다는 것을 가리킨다. 그 액티비티를 런치하는 인텐트 오브젝트는 그 액티비트가 오픈해야하는 정확한 데이터 타입을 명시하는 content: URI를 포함할 것이다.

이 인텐트 필터에 DEFAULT 카테고리가 제공되었다는 것에 주의하라. 그것이 있는 것은, Context.startActivity()와 Activity.startActivityForResult() 메쏘드가 마치 DAFAULT 카테고리를 포함하고 있는 것처럼 아래의 두개의 예외사항을 보유한 채로, 모든 인텐트를 취급하기 때문이다.

- 타겟 액티비트를 명시적으로 지칭하는 인텐트
- MAIN 액션과 LAUNCHER 카테고리로 구성된 인텐트

그러므로, DEFAULT 카테고리는 모든 필터에서 필수이다 – MAIN 액션과 LAUNCHER 카테고리를 가진 것은 제외함. (인텐트 필터는 명시적 인텐트를 염두에 둔 것이 아니다.)

3.

```
<intent-filter>
    <action android:name="android.intent.action.GET_CONTENT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType
       ="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

이 필터는 사용자가 선택해야하는 디렉토리에 대한 어떤 명세도 요구하지 않고 사용자에 의해 선택된 노트를 반환하는 액티비티의 능력을 설명한다. GET_CONTENT 액션은 PICK 액션과 유사하다. 두 경우 모두, 액티비티는 사용자에 의해 선택된 노트에 대한 URI를 반환한다.(각각의 경우에, NoteList 액티비티를 시작시키기 위해 startActivityForResult()를 호출했던 액티비티에 그것이 반환된다.) 하지만, 여기에서 호출자는 사용자가 가져오게 될 데이터 디렉토리 대신에 원하는 데이터 타입을 지정하고 있다.

데이터 타입 vnd.android.cursor.item/vnd.google.note는 하나의 노트에 대한 URI로, 액티비티가 반환할 수 있는 데이터 타입을 가리킨다. 반환된 URI로부터, 호출자는 노트 패드 데이터(vnd.google.note)를 보유하고 있는 컨텐트 프로바이더로부터 정확하게 하나의 아이템(vnd.android.cursor.item)에 대한 커서를 얻을 수 있다.

바꿔 말하면, 앞의 필터에 있던 PICK 액션에 대한 데이터 타입은 사용자에게 표시될 수 있는 액티비티의 데이터 타입을 가리킨다. GET_CONTENT 필터의 경우에, 그것은 호출자에게 반환될 수 있는 액티비티의 데이터 타입을 가리킨다.

이와 같이 주어진 능력들은, 다음과 같은 인텐트들을 NoteList 액티비티에 확정할 것이다.

action: android.intent.action.MAIN

지정된 어떤 데이터도 없이 액티비티를 런치한다.

action: android.intent.action.MAIN

category: android.intent.category.LAUNCHER

선택되어 지정된 어떤 데이터도 없이 액티비티를 런치한다. 이것은 런처가 그것의 최상위 목록을 구성하기 위해 사용되는 실제 인텐트이다. 이 액션과 카테고리와 일치되는 필터를 가진 모든 액티비티들은 그 목록에 추가된다.

action: android.intent.action.VIEW
data: content://com.google.provider.NotePad/notes

content://com.google.provider.NotePad/notes 아래의 모든 노트 목록을 표시할 액티비티를 요청한다. 그런 다음, 사용자는 그 목록을 열람할 수 있고, 그곳에서 아이템에 대한 정보를 얻을 수 있다.

action: android.intent.action.PICK
data: content://com.google.provider.NotePad/notes

content://com.google.provider.NotePad/notes 아래의 노트 목록을 표시할 액티비티를 요청한다. 그런 다음에, 사용자는 그 목록에서 노트를 가져올 수 있고, 그 액티비티는 NoteList 액티비티를 시작했던 액티비티에게 그 아이템에 대한 URI를 반환할 것이다.

action: android.intent.action.GET_CONTENT
data type: vnd.android.cursor.item/vnd.google.note

하나의 노트 패드 데이터 아이템을 제공하는 액티비티를 요청한다.

두번 째 액티비티인 NoteEditor는 사용자에게 하나의 노트 입력란을 보여주며, 그것을 편집하는 것을 가능하게 한다. 그것은 두개의 인텐트 필터에 의해 설명되듯이 두 가지 것을 할 수 있다.

1.

```
<intent-filter android:label="@string/resolve_edit">
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="com.android.notepad.action.EDIT_NOTE" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType
      ="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

이 액티비트의 첫번째 기본적인 목적은, 사용자가 노트를 VIEW 하거나 그것을 EDIT 하도록 하나의 노트와 상호작용하는 것을 가능하게 하는 것이다. (EDIT_NOTE 카테고리는 EDIT와 동의어이다.) 그 인텐트는 MIME 타입 vnd.android.

cursor.item/vnd.google.note와 일치하는 데이터에 대한 URI, 즉 하나의 특정 노트에 대한 URI를 포함할 것이다. 그것은 전형적으로 NoteList 액티비티의 PICK 또는 GET_CONTENT 액션에 의해 반환되는 URI가 될 것이다.

앞에서 처럼, 이 필터는 액티비티가 NoteEditor 클래스를 명시적으로 지정하지 않은 인텐트에 의해서 런치될 수 있도록 DEFAULT 카테고리를 포함한다.

2.

```
<intent-filter>
    <action android:name="android.intent.action.INSERT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
```

이 액티비티의 두번째 목적은 사용자가 신규 노트를 생성하는 것을 가능하게 하는 것이다. 그 노트는 이미 존재하는 노트 디렉토리에 INSERT 될 것이다. 그 인텐트는 MIME 타입 vnd.android.cursor.dir/vnd.google.note와 일치하는 데이터에 대한 URI, 즉 노트가 놓여져야 하는 디렉토리게 대한 URI를 포함할 것이다.

이와 같이 주어진 능력들은, 다음과 같은 인텐트들을 NoteEditor 액티비티에 확정할 것이다.

action: android.intent.action.VIEW
data: content://com.google.provider.NotePad/notes/ID

ID로 식별되는 노트 컨텐트를 보여줄 액티비티를 요청한다. (content: URI가 어떤 그룹의 개별 구성물을 지정하는 상세한 방법에 대해서는, Content Providers를 보라.)

action: android.intent.action.EDIT
data: content://com.google.provider.NotePad/notes/ID

ID로 식별되는 노트 컨텐트를 보여주고 사용자가 그것을 편집할 수 있게 할 액티비티를 요청한다, 만약 사용자가 변경분을 저장하면, 액티비티는 컨텐트 프로바이더에 있는 노트에 대한 데이터를 갱신한다.

action: android.intent.action.INSERT
data: content://com.google.provider.NotePad/notes

content://com.google.provider.NotePad/notes에 있는 노트 목록에 신규로 빈 노트를 생성하고 사용자가 그것을 편집할 수 있게 할 액티비티를 요청하다. 만약 사용자가 노트를 저장하면, 그것의 URI가 호

출자에게 반환된다.

마지막 액티비티인 TitleEditor는 사용자가 노트의 제목을 편집하는 것을 가능하게 한다. 이것은 (명시적으로 인텐트에 그 컴포넌트 이름을 셋팅함으로써) 인텐트 필터를 사용하지 않고 직접적으로 그 액티비티를 호출함으로써 구현될 수 있다. 그러나 우리는 여기에서 존재하는 데이터에 택일적 오퍼레이션을 배포하는 방법을 보여주는 기회를 갖는다.

```
<intent-filter android:label="@string/resolve_title">
  <action android:name="com.android.notepad.action.EDIT_TITLE" />
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.ALTERNATIVE" />
  <category android:name
       ="android.intent.category.SELECTED_ALTERNATIVE" />
  <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

이 액티비티에 대한 하나의 인텐트 필터는 "com.android.notepad.action.EDIT_TITLE"라 불리는 커스텀 액션을 사용한다. 그것은 이전의 VIEW와 EDIT 액션과 같이 특정 노트(데이터 타입 vnd.android.cursor.item/vnd.google.note)에 호출되어야 한다. 하지만, 여기에서 그 액티비티는 노트 컨텐트 자체가 아닌, 그 노트 데이터에 포함된 제목을 보여준다.

통상적인 DEFAULT 카테고리를 지원하는 것에 추가해서, 제목 편집기는 또한 두개의 다른 표준 카테고리인 ALTERNATIVE와 SELECTED_ALTERNATIVE를 지원한다. 이 카테고리들은 (LAUNCHER 카테고리가 애플리케이션 런처에서 사용자에게 보여져야하는 액티비티를 구분하는 것처럼) 옵션 메뉴에서 사용자에게 보여질 수 있는 액티비티를 구분한다. 이 필터는 또한 (android:label="@string/resolve_title" 를 통해) 명시적인 라벨을 제공하고 있음에 주의하라. 이것은 사용자가 현재 보고 있는 데이터에 대한 택일적 액션으로써 이 액비티비를 표시될 때 사용자가 보는 것을 더 좋게 보도록 하기 위한 것이다. (이 카테고리에 대한 더 자세한 정보와 옵션 메뉴 구성은, PackageManager.queryIntentActivityOptions()와 Menu.addIntentOptions() 메쏘드를 보라.)

이와 같이 주어진 능력들은, 다음과 같은 인텐트들을 TitleEditor 액티비티에 확정할 것이다.

action: com.android.notepad.action.EDIT_TITLE
data: content://com.google.provider.NotePad/notes/ID

노트 ID와 결합된 제목을 보여주고, 사용자에게 제목을 편집할 수 있게 하는 액티비티를 요청한다.

Section | 6

데이터 저장공간

6.1. 프레퍼런스

6.2. 파일

6.3. 데이터베이스

6.4. 네트워크

6. 데이터 저장공간

전형적인 데스크탑 운영 체제는 파일을 다른 애플리케이션이 읽을 수 있도록 저장하기 위해, 모든 애플리케이션이 사용할 수 있는 보편적 파일 시스템을 제공한다(아마 약간의 접근 제한 설정을 가진다). 안드로이드는 다른 형태의 시스템을 사용한다. 안드로이드 상에서, 모든 애플리케이션의 데이터(파일을 포함함)는 그 애플리케이션에게 프라이빗private하다.

하지만 안드로이드는 또한 – 컨텐트 프로바이더를 통해서 – 그것의 프라이빗private 데이터를 다른 애플리케이션에게 보여주기 위해, 애플리케이션에 대한 표준 방법을 제공한다. 컨텐트 프로바이더는 애플리케이션 데이터에 읽기/쓰기 접근을 제공하는 애플리케이션의 선택사항(역주: 있어도 없어도 되는)에 속하는 컴포넌트이다. 그 읽기/쓰기 접근은 그 컨텐트 프로바이더가 부과할 수도 있는 제약의 조건에 따른다. 그 컨텐트 프로바이더는 데이터를 요청하고 수정하는 것에 대한 표준 문법과, 그리고 리턴된 데이터를 읽는 것에 대한 표준 메커니즘을 구현한다. 안드로이드는 이미지, 오디오 그리고 비디오 파일과 개인 컨택 정보와 같은 표준 데이터 타입에 대한 다수의 컨텐트 프로바이더를 지원한다. 컨텐트 프로바이더를 사용하는 것에 대한 더 많은 정보에 대해서는 7장. “컨텐트 프로바이더”를 보라.

여러분이 다른 애플리케이션에게 자신의 애플리케이션 데이터를 내보내길 원하던 또는 원하지 않던 간에, 여러분은 그것을 저장하는 방법이 필요하다. 안드로이드는 데이터를 저장하고 가져오기 위해 다음의 네 가지 메커니즘을 제공한다: 프레퍼런스Preferences, 파일, 데이터베이스, 그리고 네트워크이 그것이다.

프레퍼런스(Preferences)

프레퍼런스Preference는 기본적인 데이터 타입에 대한 키와 값의 쌍key-value pair을 저장하고 가져오는 가벼운 메커니즘이다. 이것은 전형적으로 애플리케이션이 시작될 때마다 로드되어야 하는 기본적인 환영 인사말이나 텍스트 폰트와 같은, 애플리케이션의 환경설정 정보를 저장하기 위해 사용된다. 이곳에 값을 읽고 쓰기 위해서는 Context.getSharedPreferences()를 호출하라. 만약 여러분이 같은 애플리케이션 내의 다른 컴포넌트들에게 이것을 공유하고자 한다면, 여러분의 프레퍼런스 집합에 이름을 할당하라. 또한 호출하는 액티비티의 프라이빗private 영역에 그것을 유지하기 위해서는, 이름없이 Activity.getPreferences()를 사용하라. 여러분은 (컨텐트 프로바이더를 사용하는 것에 의하지 않고서는) 프레퍼런스를 애플리케이션의 경계를 넘어서 공유할 수 없다.

여기에 계산기에 대한 소리없는 키입력 모드를 위한 사용자 프레퍼런스를 설정하는 예제가 있다.

```
import android.app.Activity;
import android.content.SharedPreferences;

public class Calc extends Activity {
public static final String PREFS_NAME = "MyPrefsFile";
    . . .
   @Override
   protected void onCreate(Bundle state){
     super.onCreate(state);
     . . .
     // Restore preferences
     SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
     boolean silent = settings.getBoolean("silentMode", false);
     setSilent(silent);
   }

   @Override
   protected void onStop(){
     super.onStop();
     // Save user preferences. We need an Editor object to
     // make changes. All objects are from android.context.Context
     SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
```

```
        SharedPreferences.Editor editor = settings.edit();
        editor.putBoolean("silentMode", mSilentMode);
        // Don't forget to commit your edits!!!
        editor.commit();
    }
}
```

파일(Files)

여러분은 모바일 디바이스 상에 또는 분리될 수 있는 저장 매체 상에 직접적으로 파일들을 저장할 수 있다. 디폴트로 다른 애플리케이션들은 이러한 파일들에 접근할 수 없다.

파일로부터의 데이터를 읽기 위해서는 Context.openFileInput()을 호출하고, 그것에 파일의 로컬 이름과 경로path를 전달하라. 그것은 표준 Java FileInputStream 오브젝트를 리턴한다. 파일에 쓰기 위해서는, 이름과 경로를 가지고 Context.openFileOutput()을 호출하라. 그것은 FileOutputStream 오브젝트를 리턴한다. 다른 애플리케이션에서 이름과 경로 문자열를 가지고 이 메쏘드를 호출하는 것은 동작하지 않을 것이다; 여러분은 단지 로컬 파일들에만 접근할 수 있다.

만약 여러분이 컴파일 시점에 여러분의 애플리케이션과 함께 패키지할 정적static 파일을 가지고 있다면, 여러분 프로젝트의 res/raw/myDataFile로 그 파일을 저장할 수 있으며, 그리고 나서 Resources.openRawResource(R.raw.myDataFile)를 사용해서 그것을 오픈할 수 있다. 그것은 여러분이 파일로로부터 읽기 위해 사용할 수 있는 InputStream 오브젝트를 리턴한다.

데이터베이스(Databases)

안드로이드 API는 SQLite 데이터베이스를 생성하고 사용하는 것에 대한 지원을 포함한다. 각각의 데이터베이스는 그것을 생성한 애플리케이션에게 프라이빗private하다.

SQLiteDatabase 오브젝트는 데이터베이스를 나타내고, 그것과 상호작용하는

– 쿼리query를 하고 데이터를 관리하는 – 메쏘드들을 가지고 있다. 데이터베이스를 생성하기 위해서는 SQLiteDatabase.create()를 호출하고, 또한 서브클래스 SQLiteOpenHelper를 호출하라.

SQLite 데이터베이스 시스템을 위한 지원support의 일부로써, 안드로이드는 데이터베이스 관리 기능을 제공한다. 그것은 여러분이 유용한 오브젝트들로 감싸진wrapped 복잡한 데이터 집합들collection을 저장하게 한다. 예를 들어 안드로이드는 컨택contact 정보에 대한 데이터 타입을 정의한다. 그것은 성과 이름(문자열), 주소와 전화번호(또한 문자열), 사진(비트맵 이미지), 그리고 사람을 설명하는 다른 많은 정보를 포함하는 많은 필드들로 구성되어 있다.

안드로이드는 여러분이 테이블 컨텐츠를 브라우즈하고, SQL 명령어를 실행하고, 그리고 SQL 데이터베이스의 다른 유용한 기능을 수행하는 것을 가능하게 하는 sqlite3 데이터베이스 도구를 함께 배포하고 있다. 이 프로그램을 실행하는 방법을 배우기 위해서는 아래의 내용을 보라.

원격 쉘(shell)에서 sqlite3 데이터베이스 사용하기

adb remote shell 에서, 여러분은 안드로이드 애플리케이션이 생성한 SQLite 데이터베이스를 관리하기 위해 sqlite3 커맨드-라인command-line 프로그램을 사용할 수 있다. 그 sqlite3 툴tool은 테이블의 컨텐츠를 출력하기 위한 .dump, 존재하는 테이블에 대한 SQL 생성문(CREATE statement)을 출력하는 .schema와 같은 많은 유용한 명령어를 포함하고 있다. 그 툴tool은 또한 작동 중에on the fly 여러분이 SQLite 명령어를 실행할 수 있도록 해 준다.

sqlite3를 사용하기 위해서 위에서 언급했듯이, 에뮬레이터 인스턴스에 remote shell을 입력하라. 그런 다음에 sqlite3 명령어를 사용해서 그 툴tool을 불러라. 선택 사항으로 sqlite3를 부를 때, 여러분은 탐색하고자 하는 데이터베이스의 전체 경로를 지정할 수 있다. 에뮬레이터/디바이스 인스턴스는 SQLite3 데이터베이스를 /data/data/<package_name>/databases/ 폴더에 저장한다.

여기에 예제가 있다.

```
$ adb -s emulator-5554 shell
```

```
# sqlite3 /data/data/com.example.google.rss.rssexample/databases/
rssitems.db
SQLite version 3.3.12
Enter ".help" for instructions
.... enter commands, then quit. ...
sqlite> .exit
```

일단 여러분이 sqlite3를 불렀다면, 여러분은 그 shell에서 sqlite3 명령어를 실행할 수 있다. sqlite3를 종료하고 adb remote shell로 리턴하기 위해서는, .exit 또는 CTRL-D를 사용하라.

모든 데이터베이스(SQLite와 그 밖의 것들)는 디바이스 상의 /data/data/package_name/databases 안에 저장된다.

얼마나 많은 테이블들이 생성되는지, 그것들이 포함하는 필드들이 무엇인지, 그리고 그것들은 어떻게 연결되는 지에 대한 논의는 이 노트의 범위를 벗어난 것이다. 그러나 안드로이드는 표준 SQLite 컨셉concept을 넘어서는 어떤 제한도 강제하지 않는다. 우리는 빠르게 레코드를 찾기 위해, 레코드에 대한 고유한 ID로 사용될 수 있는 자동증가 값 키 필드autoincrement value key field를 포함할 것을 권장한다. 이것은 프라이빗private 데이터를 위해서는 요구되지 않지만, 만약 여러분이 컨텐트 프로바이더를 구현한다면, 여러분은 그러한 고유한 ID 필드를 포함해야 한다. 이 필드에 대한 더 많은 정보에 대해서는 컨텐트 프로바이더 문서를 보고, 새로운 데이터베이스를 생성하고 거주시키는populate(역주: 테이블들을 그 안에 놓는) 예제에 대해서는 NotePad 샘플 코드의 NotePadProvider를 보라. 여러분이 생성한 어떤 데이터베이스는 애플리케이션 내의 다른 어떤 클래스에서든 이름에 의해 접근할 수 있다. 그러나 애플리케이션 밖에서는 아니다.

네트워크(Network)

여러분은 또한 (네트워크가 사용 가능하다면) 데이터를 저장하고 가져오기 위해 네트워크를 사용할 수 있다. 네트워크 오퍼레이션을 하기 위해서는, 다음의 패키지에 있는 클래스를 사용하라:

```
java.net.*
android.net.*
```

Section | 7

컨텐트 프로바이더

7.1. 컨텐트 프로바이더 기초

7.2. 컨텐트 프로바이더 쿼리하기

7.3. 프로바이더 데이터 수정하기

7.4. 컨텐트 프로바이더 생성하기

7.5. 컨텐트 URI 개요 요약

7. 컨텐트 프로바이더

컨텐트 프로바이더는 데이터를 저장하고 가져오며, 그것에 대한 접근을 모든 애플리케이션에게 가능하게 한다. 이것은 애플리케이션들 간에 데이터를 공유할 수 있는 유일한 방법이다; 모든 안드로이드 패키지가 접근할 수 있는 공통된 저장공간은 존재하지 않는다.

안드로이드는 일반적인 데이터 타입들(오디오, 비디오, 이미지, 개인 컨택 정보, 기타 등등)에 대한 몇 개의 컨텐트 프로바이더를 포함하고 있다. 여러분은 android.provider 패키지에서 그것들 중의 몇 개의 목록을 볼 수 있다. 여러분은 그것들이 보유하고 있는 데이터를 위해 이러한 프로바이더에게 쿼리query할 수 있다(비록, 어떤 것에 대해서는 여러분이 데이터를 읽기 위한 적절한 퍼미션을 획득해야 하지만).

만약 자기 자신의 데이터를 퍼블릭public하게 만들고자 한다면, 여러분은 두 가지 옵션을 가진다: 자기 자신의 컨텐트 프로바이더(ContentProvider 서브클래스)를 만들거나, 또는 이미 존재하는 프로바이더에 그 데이터를 추가할 수 있다 – 이것은 그곳에 같은 타입의 데이터를 제어하는 것이 있고, 여러분이 그곳에 쓰기 위한 퍼미션을 가지고 있다면 가능하다.

이 문서는 컨텐트 프로바이더 사용에 대한 소개이다. 여기에서는 기본적인 것들에 대해 간단히 설명한 후, 컨텐트 프로바이더를 쿼리하는 방법, 컨텐트 프로바이더에 의해 관리되는 데이터를 수정하는 방법, 그리고 여러분 자신의 컨텐트 프로바이더를 만드는 방법을 설명한다.

컨텐트 프로바이더 기초

컨텐트 프로바이더라는 덮개cover 아래에 있는 데이터를 실제로 저장하는 방법은 컨텐트 프로바이더를 설계한 사람에게 달려있다. 하지만 모든 컨텐트 프로바이더는 데이터 추가, 수정, 삭제와 같은 프로바이더를 쿼리하고 그 결과를 리턴받는 보편적인 인터페이스를 구현한다.

그것은 가장 일반적으로 컨텐트 리졸버ContentResolver 오브젝트를 통해, 클라이언트가 간접적으로 사용하는 인터페이스이다. 여러분은 액티비티나 다른 애플리케이션 컴포넌트를 구현한 내부에서 getContentResolver()를 호출함으로써 컨텐트 리졸버를 얻게 된다.

```
ContentResolver cr = getContentResolver();
```

그런 다음, 여러분이 관심 갖는 컨텐트 프로바이더가 무엇이든 간에 그것과 상호작용하기 위해 컨텐트 리졸버의 메쏘드들을 사용할 수 있다.

하나의 쿼리가 개시될 때, 안드로이드 시스템은 쿼리의 대상이 되는 컨텐트 프로바이더를 식별하고, 그것이 로드되고 실행되도록 한다. 시스템은 모든 컨텐트 프로바이더ContentProvider 오브젝트를 인스턴스화 한다; 여러분은 스스로 그것을 할 필요는 결코 없다. 실제로, 여러분은 결코 컨텐트 프로바이더ContentProvider 오브젝트를 직접 다루지 않는다. 일반적으로는 각각의 컨텐트 프로바이더 타입 별로 하나의 인스턴스가 있을 뿐이다. 하지만 그것은 서로 다른 애플리케이션과 프로세스에 있는 여러 개의 컨텐트 리졸버ContentResolver와 커뮤니케이션할 수 있다. 프로세스들 간의 상호작용은 컨텐트 리졸버ContentResolver와 컨텐트 프로바이더ContentProvider 클래스에 의해 제어된다.

데이터 모델(The data model)

컨텐트 프로바이더는 데이터베이스 모델 상의 간단한 테이블처럼 그것의 데이터를 보여준다. 그곳에서 각 행은 레코드이며, 각 열은 특정 타입과 의미의 데이터이다. 예를 들어 사람들과 그들의 전화번호에 관한 정보는 아래와 같이 보여질 수 있다.

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

모든 레코드는 하나의 숫자numeric _ID 필드를 포함하며, 이것은 테이블 내에서 그 레코드를 고유하게 구분한다. ID들은 관련된 테이블 내의 레코드들을 매칭하기 위해 사용될 수 있다. 예를 들어 ID들은 하나의 테이블 내에서 특정인의 전화번호를 찾아서 다른 테이블에서 그 사람의 사진을 찾기 위해 사용될 수 있다.

쿼리는 각각의 필드의 내용들을 읽어 들이기 위해, 레코드에서 레코드로의 이동과 컬럼에서 컬럼으로의 이동을 가능하게 하는 커서Cursor 오브젝트를 리턴한다. 그것은 각각의 데이터 타입을 읽기 위한 특별한 메쏘드를 가진다. 따라서 특정 필드를 읽기 위해서 여러분은 그 필드가 어떤 타입의 데이터 보유하고 있는 지를 알아야 한다(쿼리의 결과와 커서 오브젝트에 대한 더 많은 것들은 나중에 설명된다).

URIs

각각의 컨텐트 프로바이더는 그 데이터 집합을 고유하게 식별하는 하나의 퍼블릭public URI(Uri 오브젝트로 감싸여짐)를 제공한다. 여러 개의 데이터 집합(여러 개의 테이블)을 제어하는 컨텐트 프로바이더는 각각에 대한 별도의 URI를 제공한다. 프로바이더를 위한 모든 URI는 "content://" 라는 문자열로 시작된다. content: 스키마scheme는 그 데이터가 컨텐트 프로바이더에 의해 제어되는 것으로 간주한다.

만약 여러분이 컨텐트 프로바이더를 정의하고 있다면, 클라이언트 코드를 단순화하고 나중에 업데이트가 더 쉽도록 그것의 URI를 상수로 정의하는 것이 좋다. 안드로이드는 안드로이드 플랫폼에 있는 모든 프로바이더에 대해 CONTENT_URI 상수를 정의하고 있다. 예를 들어 (컨택 프로바이더에 의해 둘 다 제어되는) 전화번호를 사람에 매칭시키는 테이블에 대한 URI와 사람에 대한 사진을 보유하고 있는 테이블에 대한 URI는 다음과 같다.

```
android.provider.Contacts.Phones.CONTENT_URI
android.provider.Contacts.Photos.CONTENT_URI
```

유사한 형태로, 최근 전화 통화 테이블과 캘린더 테이블에 대한 URI는 다음과 같다.

```
android.provider.CallLog.Calls.CONTENT_URI
android.provider.Calendar.CONTENT_URI
```

URI 상수는 컨텐트 프로바이더와의 모든 상호작용에서 사용된다. 모든 컨텐트 리졸버ContentResolver 메쏘드는 그것의 첫 번째 매개변수로 URI를 갖는다. 그것은 컨텐트 리졸버ContentResolver가 어떤 프로바이더와 말해야 하는 지, 그리고 프로바이더의 어떤 테이블이 대상이 되어야 하는가를 구분한다.

컨텐트 프로바이더 쿼리하기

여러분이 컨텐트 프로바이더에게 쿼리하기 위해서는 세 가지 정보가 필요하다.

- 프로바이더를 식별하는 URI
- 여러분이 받고자 하는 데이터 필드들의 이름들
- 그 필드들의 데이터 타입들

만일 여러분이 특정 레코드를 쿼리를 하다면, 여러분은 또한 그 레코드에 대한 ID가 필요하다.

쿼리 만들기(Making the query)

컨텐트 프로바이더에 쿼리하기 위해서, 여러분은 ContentResolver.query() 메쏘드 또는 Activity.managedQuery() 메쏘드를 사용할 수 있다. 두 메쏘드 모두는 동일한 매개변수 집합을 가지며, 둘 다 커서 오브젝트를 리턴한다. 하지만 managedQuery()는 액티비티로 하여금 커서Cursor의 생명주기를 관리할 수 있게 한다. 매니지드 커서managed Cursor는 액티비티가 멈출 때 스스로를 제거unloading하고, 액티비티가 다시 시작될 때 스스로를 재쿼리하는 것과 같이 모든 것을 세심하게 관리한다. 여러분은 Activity.startManagingCursor()를 호출함으로써, 관리되지 않는 커서unmanaged Cursor 오브젝트를 액티비티가 관리하도록 할 수 있다.

query()나 managedQuery()에 대한 첫 번째 매개변수는 프로바이더 URI이다. CONTENT_URI 상수는 특정 컨텐트 프로바이더와 데이터 집합을 식별한다(앞쪽의 URI를 보라).

쿼리를 단지 하나의 레코드로 제한하기 위해서, 여러분은 URI에 그 레코드에 대한 _ID 값을 추가할 수 있다. 다시 말하면 URI path(경로) 영역의 마지막 위치에 ID와 일치하는 문자열을 넣으면 된다. 예를 들어 ID가 23인 경우에 URI는 다음과 같게 된다.

```
content://. . . ./23
```

이곳에는 몇 개의 헬퍼helper 메쏘드가 있는데, 특별히 ContentUris.withAppendedId()와 Uri.withAppendedPath()는 URI에 ID를 추가하는 것을 용

이하게 한다. 두 메쏘드 모두는 추가된 ID를 보유한 Uri 오브젝트를 리턴하는 정적static 메쏘드들이다. 그래서 예를 들어 만약 여러분이 사람들의 전화번호 데이터베이스에서 레코드 23을 찾는다면, 여러분은 다음과 같이 쿼리를 만들 수 있다.

```
import android.provider.Contacts.People;
import android.content.ContentUris;
import android.net.Uri;
import android.database.Cursor;

// Use the ContentUris method to produce the base URI
// for the contact with _ID == 23.
Uri myPerson = ContentUris.withAppendedId(People.CONTENT_URI, 23);

// Alternatively, use the Uri method to produce the base URI.
// It takes a string rather than an integer.
Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");
// Then query for this specific record:
Cursor cur = managedQuery(myPerson, null, null, null, null);
```

query()와 managedQuery() 메쏘드에 대한 다른 매개변수들은 더욱 상세하게 쿼리의 범위를 규정한다. 이것들은 다음과 같은 것을 한다.

- 리턴받아야 하는 데이터 컬럼의 이름들. 널null 값은 모든 모든 컬럼을 리턴한다. 그렇지 않다면, 그 이름으로 리스트된 컬럼만 리턴된다. 그 플랫폼에 속하는 모든 컨텐트 프로바이더는 그들의 컬럼에 대한 상수를 정의한다. 예를 들어 android.provider.Contacts.Phones 클래스는 앞에서 설명된 phone 테이블에 있는 _ID, NUMBER, NUMBER_KEY, NAME, 그리고 기타 등등의 컬럼 이름에 대한 상수를 정의한다.
- SQL WHERE 절(WHERE 자신은 제외함)로 형성되는 것으로, 리턴할 행들을 기술하는 필터. 널(null) 값은 모든 행들을 리턴한다(단, URI가 하나의 레코드로 쿼리를 제한하지 않는 한).
- 선택할 것에 대한 아규먼트들
- SQL ORDER BY 절(ORDER BY 자신은 제외함)로 형성되는 것으로, 리턴될 행에 대한 정렬 순서. 널(null) 값은 테이블에 대한 디폴트 순서, 어쩌면 정렬되지 않을 수도 있는 형태로 레코드들을 리턴한다.

컨택contact의 이름들과 그들의 기본적인 전화번호에 대한 리스트를 가져오기 위한 하나의 쿼리 예제를 살펴보자.

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Form an array specifying which columns to return.
String[] projection = new String[] {
                            People._ID,
                            People._COUNT,
                            People.NAME,
                            People.NUMBER
                        };

// Get the base URI for the People table
// in the Contacts content provider.
Uri contacts =  People.CONTENT_URI;

// Make the query.
Cursor managedCursor = managedQuery(contacts,
                      projection, // Which columns to return
                      null,       // Which rows to return (all rows)
                      null,       // Selection arguments (none)
                      // Put the results in ascending order by name
                      People.NAME + " ASC");
```

이 쿼리는 컨택Contacts 컨텐트 프로바이더의 People 테이블에서 데이터를 가져온다. 그것은 각 컨택contact에 대한 이름, 주요 전화 번호, 그리고 고유한 레코드 ID를 얻는다. 그것은 또한 각각의 레코드의 _COUNT 필드에 리턴되는 레코드들의 개수를 리포트한다.

컬럼의 이름에 대한 상수는 다양한 인터페이스로 정의된다 – BaseColumns에 있는 _ID와 _COUNT, PeopleColumns에 있는 NAME, 그리고 PhoneColumns에 있는 NUMBER가 그것이다. Contacts.People 클래스는 이러한 각각의 인터페이스를 구현한다. 이것이 위의 코드 예제가 클래스 이름만을 사용해서 그것들을 참조할 수 있는 지에 대한 이유이다.

쿼리는 무엇을 리턴하는가

쿼리는 0개 또는 그 이상의 데이터베이스 레코드들의 집합을 리턴한다. 컬럼들의 이름들, 그것들의 디폴트 순서, 그리고 그것들의 데이터 타입은 각각의 컨텐트 프로바이더에 따라 다르다. 그러나 모든 프로바이더는 각각의 레코드에 대한 고유한 숫자 ID를 보유하는 _ID 컬럼을 갖는다. 모든 프로바이더는 또한 _COUNT 컬럼으로 리턴되는 레코드들의 개수를 리포트 할 수 있다. 그 값은 모든 행에서 동일하다.

여기에 이전 섹션에서의 쿼리에 대한 예제 결과 집합이 있다.

_ID	_COUNT	NAME	NUMBER
44	3	Alan Vain	212 555 1234
13	3	Bully Pulpit	425 555 6677
53	3	Rex Cars	201 555 4433

가져온 데이터는 커서 오브젝트에 의해 보여지게 된다. 커서 오브젝트는 그 결과 집합set을 뒤 또는 앞으로 반복하기iterate 위해 사용될 수 있다. 여러분은 단지 데이터를 읽기 위해서만 이 커서 오브젝트를 사용할 수 있다. 데이터를 추가하고, 수정하고, 또는 삭제하기 위해서, 여러분은 컨텐트 리졸버ContentResolver 오브젝트를 사용해야 한다.

얻어온 데이터 읽기

쿼리에 의해 리턴되는 커서 오브젝트는 결과의 레코드 집합set에 대한 접근을 제공한다. 만약 여러분이 ID로 특정 레코드를 쿼리했다면, 이 집합은 오직 하나의 값만을 포함할 것이다. 그렇지 않다면, 그것은 다수의 값들을 포함할 수 있다(만약 아무것도 일치하지 않는다면, 그것은 또한 빈(empty) 것이 될 수 있다). 여러분은 레코드에 있는 특정 필드에서 데이터를 읽을 수 있다. 그러나 여러분은 필드의 데이터 타입을 알아야만 한다. 왜냐하면 커서 오브젝트는 각각의 데이터 타입을 읽기 위한 별도의 메쏘드를 가지고 있기 때문이다 – getString(), getInt(), 그리고 getFloat() 같은 것이 그것이다(하지만 대부분의 타입에 대해, 여러분이 스트링을 읽기 위한 메쏘드를 호출한다면, 커서 오브젝트는 여러분에게 데이터에 대한 스트링 표현을 제공할 것이다). 커서는 컬럼의 인덱스로부터 컬럼 이름을, 또는 컬럼 이름으로부터 인덱스 숫자를 여러분이 요청하는 것을 가능하게 한다.

다음에 나오는 단편snippet은 앞서 설명된 쿼리에서 이름과 전화번호를 읽는 것을 보여준다:

```
import android.provider.Contacts.People;

private void getColumnData(Cursor cur){
    if (cur.moveToFirst()) {

        String name;
        String phoneNumber;
        int nameColumn = cur.getColumnIndex(People.NAME);
        int phoneColumn = cur.getColumnIndex(People.NUMBER);
        String imagePath;

        do {
            // Get the field values
            name = cur.getString(nameColumn);
            phoneNumber = cur.getString(phoneColumn);

           // Do something with the values.
            ...

        } while (cur.moveToNext());

    }
}
```

만약 쿼리가 이미지나 사운드 같은 바이너리 데이터를 리턴할 수 있다면, 그 데이터가 테이블 안에 직접 입력되어 있을 수도 있고, 또는 그 데이터에 대한 테이블 엔트리는 여러분이 데이터를 얻을 수 있는 content: URI를 지정하는 문자열이 될 수도 있을 것이다. 일반적으로 더 작은 양의 데이터(20에서 50K 이하까지, 또는 그 이하를 말함)는 흔히 테이블에 직접적으로 입력되며, Cursor.getBlob()를 호출함으로써 읽어질 수 있다. 그것은 바이트 배열을 리턴한다.

만일 테이블 엔트리가 content: URI라면, 여러분은 직접 파일을 오픈하고 읽는 것을 결코 시도해서는 안된다(첫 째는 퍼미션 문제가 이것을 실패하게 만들 수 있다). 대신에, 여러분은 데이터를 읽기 위해 사용할 수 있는 InputStream 오브젝트를 얻기 위해 ContentResolver.openInputStream()을 호출해야 한다.

컨텐트 프로바이더 데이터 수정하기

컨텐트 프로바이더에 의해 유지되는 데이터는 다음과 같은 작업들에 의해 수정될 수 있다:

- 새로운 레코드를 추가하기
- 기존의 레코드에 새로운 값을 추가하기
- 기존의 레코드를 배치로 업데이트 하기
- 레코드를 삭제하기

모든 데이터 수정은 컨텐트 리졸버ContentResolver 메쏘드를 사용해서 이루어진다. 몇몇 컨텐트 프로바이더는 데이터를 읽는 것보다 쓰는 것을 더 제약하는 퍼미션을 요구한다. 만약 여러분이 컨텐트 프로바이더에 쓰기 위한 퍼미션이 없다면, 컨텐트 리졸버ContentResolver 메쏘드는 실패할 것이다.

레코드 추가하기(Adding records)

새로운 레코드를 컨텐트 프로바이더에 추가하기 위해, 먼저 ContentValues 오브젝트에 키-값 쌍key-value pair의 맵map을 구성하라. 그 오브젝트의 각각의 키는 컨텐트 프로바이더에 있는 컬럼의 이름과 매칭되며, 그 값은 그 컬럼 내의 신규 레코드에서 요구되어지는 값이다. 그런 다음에 ContentResolver.insert()를 호출하고, 그것에게 프로바이더의 URI와 ContentValues 맵을 전달하라. 이 메쏘드는 새로운 레코드에 대한 전체 URI를 리턴한다 - 즉, 새로운 레코드를 위해 추가된 ID를 가지는 프로바이더의 URI이다. 나중에 여러분은 쿼리를 위해 이 URI를 사용할 수 있고, 새로운 레코드에 대한 커서를 얻을 수 있고, 더 나아가 레코드를 수정하기 위해 사용할 수 있다. 여기에 예제가 있다:

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

ContentValues values = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
```

```
values.put(People.NAME, "Abraham Lincoln");
// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
values.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

새로운 값 추가하기(Adding new values)

일단 하나의 레코드가 존재한다면, 여러분은 그것에 새로운 정보를 추가하거나 또는 기존의 정보를 수정할 수 있다. 예를 들어 위의 예제의 다음 절차는 새로운 엔트리에 – 전화번호 또는 IM 또는 전자메일 같은 – 정보를 추가하는 것이 될 것이다.

컨택Contacts 데이터베이스의 하나의 레코드에 추가하는 최고의 방법은 그 레코드에 대한 URI에 새로운 데이터가 들어갈 테이블의 이름을 덧붙이는append 것이다. 그런 다음에 새로운 데이터 값을 추가하기 위해 수정된 URI를 사용하는 것이다. 각각의 컨택Contacts 테이블은 이러한 목적을 위해 CONTENT_DIRECTORY 상수로써 이름을 제공한다. 아래의 코드는 방금 생성된 레코드에 대해 전화번호와 전자메일 주소를 추가함으로써 이전 예제를 계속한다.

```
Uri phoneUri = null;
Uri emailUri = null;

// Add a phone number for Abraham Lincoln.  Begin with the URI for
// the new record just returned by insert(); it ends with the _ID
// of the new record, so we don't have to add the ID ourselves.
// Then append the designation for the phone table to this URI,
// and use the resulting URI to insert the phone number.
phoneUri = Uri.withAppendedPath(uri, People.Phones.CONTENT_DIRECTORY);

values.clear();
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
values.put(People.Phones.NUMBER, "1233214567");
getContentResolver().insert(phoneUri, values);

// Now add an email address in the same way.
emailUri
```

```
= Uri.withAppendedPath(uri, People.ContactMethods.CONTENT_DIRECTORY);

values.clear();
// ContactMethods.KIND is used to distinguish different kinds of
// contact methods, such as email, IM, etc.
values.put(People.ContactMethods.KIND, Contacts.KIND_EMAIL);
values.put(People.ContactMethods.DATA, "test@example.com");
values.put(
       People.ContactMethods.TYPE, People.ContactMethods.TYPE_HOME);
getContentResolver().insert(emailUri, values);
```

여러분은 바이트 배열을 가지는 ContentValues.put()을 호출함으로써, 테이블 안에 적은 양의 바이너리 데이터를 넣을 수 있다. 예를 들어 그것은 작은 아이콘 같은 이미지나 짧은 오디오 클립을 위한 작업이 될 것이다. 그러나 만약 여러분이 사진이나 완전한 노래같은 것들을 추가하기 위한 큰 양의 바이너리 데이터를 가진다면, 그 데이터에 대한 content:: URI를 테이블에 넣고, 그리고 그 파일의 URI를 가지고 ContentResolver.openOutputStream()을 호출하라(그것은 컨텐트 프로바이더가 파일로 데이터를 저장하고, 숨겨진 레코드 필드에 그 파일 경로를 기록하게 할 것이다).

이런 점에 있어서 이미지, 오디오, 그리고 비디오 데이터를 배분하는 메인main 프로바이더인 미디어 스토어MediaStore 컨텐트 프로바이더는 특별한 규정convention을 채택하고 있다. (주어진 사진의 제목 또는 사진이 찍힌 날짜 같은) 바이너리 데이터에 관한 메타 정보를 얻기 위해 query() 또는 managedQuery()에서 사용된 동일한 URI가 데이터 그 자체를 얻기 위한 openInputStream()에서 사용된다. 비슷하게, 메타 정보를 미디어 스토어MediaStore 레코드에 넣기 위해 insert()에서 사용된 동일한 URI가 그곳에 바이너리 데이터를 넣기 위한 openOutputStream()에서 사용된다. 다음에 오는 코드 단편snippet은 이 규정을 설명한다. (역주 : 위의 내용에서 query() 또는 insert()에서 사용된 동일한 URI가 아니라, 그 결과 URI가 openInputStream()과 openOutputStream()에서 사용된다고 봐야 할 듯 하다.)

```
import android.provider.MediaStore.Images.Media;
import android.content.ContentValues;
import java.io.OutputStream;

// Save the name and description of an image in a ContentValues map.
ContentValues values = new ContentValues(3);
values.put(Media.DISPLAY_NAME, "road_trip_1");
```

```
values.put(Media.DESCRIPTION, "Day 1, trip to Los Angeles");
values.put(Media.MIME_TYPE, "image/jpeg");

// Add a new record without the bitmap, but with the values just set.
// insert() returns the URI of the new record.
Uri uri
= getContentResolver().insert(Media.EXTERNAL_CONTENT_URI, values);

// Now get a handle to the file for that record,
// and save the data into it.
// Here, sourceBitmap is a Bitmap object representing the file
// to save to the database.
try {
    OutputStream outStream
       = getContentResolver().openOutputStream(uri);
    sourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outStream);
    outStream.close();
} catch (Exception e) {
    Log.e(TAG, "exception while writing image", e);
}
```

레코드를 배치로 업데이트하기

레코드들의 그룹을 배치로 업데이트 하기 위해서는(예를 들어 모든 필드에서 "NY"를 "" New York"으로 변경하기 위해), 변경하기 위한 컬럼과 값을 가지는 ContentResolver.update()를 호출하라.

레코드 삭제하기

하나의 레코드를 삭제하기 위해서는, 특정 행에 대한 URI를 가지는 ContentResolver.delete()를 호출하라..

다수의 행을 삭제하기 위해서는, 삭제하고자 하는 레코드 타입의 URI(예를 들어 android.provider.Contacts.People.CONTENT_URI)와 삭제할 행을 정의하는 SQL WHERE 절(clause)을 가지고 ContentResolver.delete()를 호출하라(주의: 만약 여러분이 일반적 타입을 삭제하거나 또는 여러분이 의도했던 것보다 더 많은 레코드를 삭제할 위험이 있다면, 유효한 WHERE 절(clause)을 포함했는가를 확인하라).

컨텐트 프로바이더 생성하기

컨텐트 프로바이더를 만들기 위해, 여러분은 다음과 같이 해야 한다.

- 데이터를 저장하기 위한 시스템을 구성하라. 대부분의 컨텐트 프로바이더는 안드로이드의 파일 저장 메쏘드 또는 SQLite 데이터베이스를 사용하여 자신의 데이터를 저장한다. 그러나 여러분은 자신이 원하는 임의의 방법으로 여러분의 데이터를 저장할 수 있다. 안드로이드는 여러분이 데이터베이스를 생성하는 것을 돕기 위해, SQLiteOpenHelper 클래스와 그것을 관리하기 위한 SQLiteDatabase를 제공한다.
- 데이터에 대한 접근을 제공하기 위해 컨텐트 프로바이더 클래스를 확장하라.
- 여러분의 애플리케이션에 대한 매니페스트 파일(AndroidManifest.xml)에 컨텐트 프로바이더를 선언하라.

아래의 섹션은 이러한 작업의 마지막 두 개에 대해 언급한다.

컨텐트 프로바이더 클래스 확장하기

여러분은 컨텐트 리졸버ContentResolver와 커서Cursor 오브젝트에 의해 기대되는 규약convension을 사용하는 다른 것들에게, 여러분의 데이터를 보여주기 위해 컨텐트 프로바이더ContentProvider 서브클래스를 정의한다. 주로 이것은 컨텐트 프로바이더ContentProvider 클래스에 선언된 여섯 개의 추상 메쏘드를 구현하는 것을 의미한다.

```
query()
insert()
update()
delete()
getType()
onCreate()
```

query() 메쏘드는 요청받은 데이터를 반복할iterate 수 있는 커서Cursor 오브젝트를 리턴해야 한다. 커서 자체는 하나의 인터페이스이다. 하지만 안드로이드는 여러분이 사용할 수 있는 미리 만들어진 몇 개의 커서 오브젝트를 제공한다. 예를 들어 SQLiteCursor는 SQLite 데이터베이스에 저장된 데이터를 반복할iterate 수 있다. 여러분은 임의의 SQLiteDatabase 클래스의 query() 메쏘드들을 호출함

으로써 커서 오브젝트를 얻는다. 거기에는 MatrixCursor와 같은 데이터베이스에 저장되지 않은 데이터에 대한 또 다른 커서 구현들이 있다.

이러한 컨텐트 프로바이더 메쏘드들은 다른 프로세스와 쓰레드에 있는 다양한 컨텐트 리졸버ContentResolver 오브젝트에서 호출될 수 있기 때문에, 그들은 쓰레드-세이프thread-safe 방식으로 구현되어야 한다.

관례에 따라, 여러분은 또한 데이터에 대한 수정이 있을 때, 리스너listener에게 통지하기 위해 ContentResolver.notifyChange()를 호출하고 싶을 수도 있다.

서브클래스 그 자체를 정의하는 것을 넘어서, 여러분이 클라이언트의 작업을 단순화시키고 그 클래스를 더 접근가능하게 만드는 다른 절차들이 있다.

- CONTENT_URI 라고 명명된 public static final Uri를 정의하라. 이것은 여러분의 컨텐트 프로바이더가 취급하는 전체 content: URI 를 나타내는 문자열이다. 여러분은 이 값에 대한 고유한 문자열을 정의해야 한다. 최선의 해법은 (소문자로 만들어진) 컨텐트 프로바이더의 전체 클래스 이름fully-qualified class name을 사용하는 것이다. 그러므로 예를 들어 TransportationProvider 클래스를 위한 URI는 다음과 같이 정의될 수 있다:

  ```
  public static final Uri CONTENT_URI
  = Uri.parse("content://com.example.codelab.transporationprovider");
  ```

  만약 프로바이더가 서브테이블들을 가진다면, 또한 각각의 서브테이블들에 대한 CONTENT_URI 상수들을 정의하라. 이러한 URI 들은 모두 동일한 authority(권위)를 가져야 하며(왜냐하면, 컨텐트 프로바이더를 식별해야 하기 때문에), 그것들의 경로path에 의해서만 구별되어야 한다. 예를 들어 다음과 같다.

  ```
  content://com.example.codelab.transporationprovider/train
  content://com.example.codelab.transporationprovider/air/domestic
  content://com.example.codelab.transporationprovider/air/interna-
  tional
  ```

  content: URI에 대한 개요에 대해서는, 이 장의 끝에 있는 컨텐트 URI 요약을 보라.

- 컨텐트 프로바이더가 클라이언트에게 리턴할 컬럼 이름들을 정의하라. 만약 여러분이 데이터베이스를 사용하고 있다면, 이 컬럼 이름은 그들이 나타내는

SQL 데이터베이스 컬럼 이름과 전형적으로 동일하다. 또한 클라이언트가 쿼리와 다른 명령에서 컬럼들을 지정하기 위해 사용할 수 있는 public static String 상수를 정의하라.

레코드의 ID에 대한 "_id"(상수 _ID를 가지는)라 이름 부쳐진 정수integer 컬럼을 포함하는 것을 확인하라. 여러분은 또한 (URL과 같은) 다른 필드를 가지는 지와 상관없이 모든 레코드에 고유한 이 필드를 가져야 한다. 만약 여러분이 SQLite 데이터베이스를 사용한다면, _ID 필드는 다음과 같은 타입이 되어야 한다.

```
INTEGER PRIMARY KEY AUTOINCREMENT
```

AUTOINCREMENT 기술자(descriptor)는 선택사항이다. 하지만 그것이 없다면, SQLite는 그 컬럼에 존재하는 가장 큰 숫자 위의 다음 숫자로 ID 카운터 필드를 증가시킨다. 만약에 여러분이 마지막 레코드를 삭제한다면, 다음에 추가된 행은 삭제된 행과 같은 ID를 가질 것이다. AUTOINCREMENT는 그 값이 삭제되었는지 여부와 상관없이 SQLite에 의해 다음으로 가장 큰 값으로 증가됨으로써 이것을 회피한다.

- 신중하게 각각의 컬럼의 데이터 타입을 문서화하라. 클라이언트는 데이터를 읽기 위해 이 정보가 필요하다.
- 만약 여러분이 새로운 데이터 타입을 취급하고 있다면, 여러분은 ContentProvider.getType()을 구현하는 곳에서 새로운 MIME 타입을 리턴하도록 정의해야 한다. getType()에 제시된 content: URI가 특정 레코드에 대한 요청을 제한하던 안하던간에 그 타입은 부분적으로 content: URI에 의존한다. 그것에는 하나의 레코드에 해당되는 MIME 타입 형식 한 개와, 여러 개의 레코드에 해당되는 또 다른 MIME 타입 형식이 있다. 요구되는 것이 무엇인가를 결정하는 것을 돕기 위한 URI 메쏘드들을 사용하라. 여기에 각각의 타입에 대한 일반적인 포맷이 있다.
  - 하나의 레코드에 대해 :

    vnd.android.cursor.item/vnd.yourcompanyname.contenttype
    예를 들어, 아래의 URI와 같이 train 레코드 122를 요청할 경우,

    content://com.example.transportationprovider/trains/122

    아래의 MIME 타입을 리턴할 수 있다.

    vnd.android.cursor.item/vnd.example.rail

- 다수의 레코드에 대해 :

  vnd.android.cursor.dir/vnd.yourcompanyname.contenttype

  예를 들어, 아래의 URI와 같이 모든 train 레코드를 요청할 경우
  content://com.example.transportationprovider/trains

  아래의 MIME 타입을 리턴할 수 있다.
  vnd.android.cursor.dir/vnd.example.rail

- 만약 여러분이 – 큰 비트맵 파일 같은 – 테이블 자체에 넣기에는 너무 큰 바이트 데이터를 보여주고 있다면, 클라이언트에게 그 데이터를 보여주는 그 필드는 실제 content: URI 문자열을 포함해야 한다. 이것은 그 데이터 파일에 클라이언트가 접근하도록 주어지는 필드이다. 그 레코드는 또한 그 파일에 대한 디바이스 상의 정확한 파일 경로를 나타내는 "_data"라고 명명된 또 다른 필드가 있어야 한다. 이 필드는 클라이언트가 아니라, 컨텐트 리졸버ContentResolver에 의해 읽혀지도록 의도된 것이다. 클라이언트는 그 아이템을 위한 URI를 보유하는 사용자가 접하는 필드에 대해서 ContentResolver.openInputStream()를 호출할 것이다. 컨텐트 리졸버ContentResolver는 그 레코드에 대한 "_data" 필드를 요청할 것이다. 그리고 그것이 클라이언트 보다 더 높은 퍼미션을 가지고 있기 때문에, 그것은 그 파일에 직접적으로 접근할 수 있어야 하며, 그 클라이언트에게 파일에 대한 읽기 위한 용기wrapper를 리턴한다.

개인적인 컨텐트 프로바이더 구현 예제를 대해서는, SDK와 함께 배포되는 노트패드NotePad 샘플 애플리케이션에 있는 NodePadProvider 클래스를 보라.

컨텐트 프로바이더 선언하기

안드로이드 시스템에게 여러분이 개발한 컨텐트 프로바이더에 관해 알게하기 위해서는, 애플리케이션의 AndroidManifest.xml 파일의 〈provider〉 엘리먼트를 사용해서 그것을 정의하라. 매니페스트에 선언되지 않은 컨텐트 프로바이더는 안드로이드 시스템에게 보여지지 않는다.

name 애트리뷰트는 컨텐트 프로바이더ContentProvider 서브클래스의 전체 이름이다. authorities 애트리뷰트는 프로바이더를 식별하는 content: URI의

authority 영역이다. 예를 들어 만약 컨텐트 프로바이더ContentProvider 서브클래스가 AutoInfoProvider라면, 〈provider〉 엘리먼트는 아래와 같이 보일 수 있다.

```
<provider name="com.example.autos.AutoInfoProvider"
          authorities="com.example.autos.autoinfoprovider"
          . . . />
</provider>
```

authorities 애트리뷰트가 content: URI의 path(경로) 영역을 생략한다는 것에 주의하라. 예를 들어 만약 AutoInfoProvider가 다른 타입의 자동차 또는 다른 제조업체에 대한 서브테이블을 제어한다면,

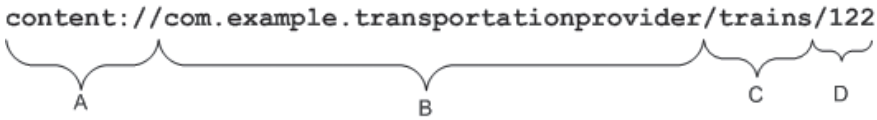
```
content://com.example.autos.autoinfoprovider/honda
content://com.example.autos.autoinfoprovider/gm/compact
content://com.example.autos.autoinfoprovider/gm/suv
```

그러한 path는 매니페스트 안에서 선언되지 않을 것이다. path(경로)가 아니라, authority(권위)가 프로바이더를 식별하는 것이다. 여러분의 프로바이더는 여러분이 선택한 어떤 방식으로 URI의 path(경로) 영역을 해석할 수 있다.

다른 〈provider〉 애트리뷰트들은 데이터를 읽고 쓰는 퍼미션을 설정할 수 있으며, 사용자에게 보여질 수 있는 아이콘과 텍스트를 제공하며, 프로바이더가 인스턴스화될 수 있게 하거나 없게 하며, 기타 등등을 설정한다. 만약 데이터가 컨텐트 프로바이더의 여러 개의 실행 버전들 간에 동기화될 필요가 없다면, multiprocess 애트리뷰트를 "참true"로 설정하라. 이것은 프로바이더의 인스턴스가 각각의 클라이언트 프로세스에서 생성될 수 있도록 허용함으로써, IPC를 실행할 필요를 제거한다.

컨텐트 URI 요약

여기에 content URI의 중요한 부분에 대한 요약이 있다.



A. 데이터가 컨텐트 프로바이더에 의해 제어된다는 것을 가리키는 표준 접두어. 그것은 결코 수정되지 않는다.

B. URI의 authority(권위) 영역; 그것은 컨텐트 프로바이더를 식별한다. 제3자 third-party 애플리케이션에 대해서, 이것은 유일성을 보장하기 위해 (소문자로 된) 전체 클래스 이름이 되어야 한다. authority는 〈provider〉 엘리먼트의 authorities 애트리뷰트에서 선언된다.

```
<provider name=".TransportationProvider"
authorities="com.example.transportationprovider"
. . .  >
```

C. path(경로)는 컨텐트 프로바이더가 요청되는 데이터의 종류를 결정하기 위해 사용된다. 이것은 없을 수도 있으며, 또한 길게 많이 세그먼트될 수 있다. 만약 컨텐트 프로바이더가 오직 하나의 데이터 타입(예를 들어 기차만)만을 보여준다면, 이것은 존재하지 않을 수 있다. 만약 프로바이더가 서브타입을 포함하여 여러 개의 타입을 보여준다면, 그것은 길게 몇 개로 세그먼트 될 수 있다. 예를 들어 네 개의 가능성을 주기 위해, "land/bus,""land/train,""sea/ship"그리고 "sea/submarine"가 될 수 있다.

D. 만약 있다면, 요청되는 특정 레코드의 ID. 이것은 요청된 레코드의 _ID 값이다. 만약 요청이 하나의 레코드에 제한되지 않는다면, 이 부분과 마직막 슬래쉬(/)는 생략된다.

```
content://com.example.transportationprovider/trains
```

Section I

8

보안 및 퍼미션(Permission)

8.1. 보안 아키텍처

8.2. 애플리케이션 사인하기

8.3. 사용자 ID와 파일 접근

8.4. 퍼미션 사용하기

8.5. 퍼미션 선언 및 강제

8.6. URI 퍼미션

8. 보안 및 퍼미션(Permission)

안드로이드는 각각의 애플리케이션과 시스템 내의 각 영역들이, 자신의 고유한 프로세스 상에서 실행되는 멀티 프로세스 시스템이다. 애플리케이션과 시스템간의 대부분의 보안은 애플리케이션에 할당된 유저User나 그룹 ID와 같은 표준 리눅스 설비를 통해 프로세스 레벨에서 강제된다. 보다 섬세한 추가적인 보안 기능은 퍼미션permission 메커니즘에 의해 제공된다. 퍼미션은 특정 프로세스가 수행할 수 있는 특정 오퍼레이션에 제약을 부가한다. 그리고 데이터의 특정 영역에 대한 임시적 접근을 부여하는 URI 별(per-URI) 퍼미션이 있다.

보안 아키텍처

안드로이드 보안 아키텍처의 핵심 디자인 요점은, 디폴트로 어떤 애플리케이션들도 다른 애플리케이션과 운영체제, 또는 사용자에게 나쁜 영향을 미칠 수 있는 임의의 오퍼레이션을 수행할 수 있는 퍼미션을 가지 않는다는 것이다. 이것은 (컨택 또는 이메일과 같은) 사용자의 개인적인 데이터에 대한 읽고 쓰기 및 다른 애플리케이션의 파일에 대한 읽고 쓰기, 네트워크 접근, 디바이스에 대한 활성 상태 유지 등을 포함한다.

애플리케이션의 프로세스는 보안 안전판sandbox이다. 그것은 기본적인 안전판에 의해 제공되지 않는 추가적인 기능들을 위해 필요한 퍼미션을 명시적으로 선언하는 않고서는, 다른 애플리케이션에 혼란을 줄 수 없다. 그것이 요구하는 이러한 퍼미션들은 다양한 방법들로 운영체제에 의해 제어될 수 있으며, 전형적으로는 인증서를 기반으로 자동으로 허용하거나 또는 불허되며, 사용자에게 확인을 요청하는 것에 의해 제어된다. 하나의 애플리케이션에 의해 요청된 퍼미션은 그 애플리케이션 내에 정적static으로 선언된다. 그러므로 그 퍼미션은 그 애플리케이션의 설치 전에 알 수 있으며, 그 후에는 변경될 수 없다.

애플리케이션 사인하기(Application Signing)

모든 안드로이드 애플리케이션들(.apk 파일들)은 그것의 개발자가 보유한 개인키 인증서로 서명되어야 한다. 이 인증서는 애플리케이션의 작성자를 식별한다. 인증서는 공식적인 인증기관에 의해 서명될 필요가 없다: 안드로이드 애플리케이션에서 자체적으로 서명된 인증서를 사용하는 것은, 완벽하게 허용되며 전형적이다. 인증서는 애플리케이션들 간의 신뢰관계를 확립하기 위해서 사용될 뿐이며, 애플리케이션이 설치될 수 있는 가에 대한 판매 통제를 위한 것이 아니다. 서명이 보안에 미치는 가장 중요한 사항들은 서명 기반의 퍼미션에 누가 접근 가능하며 누가 사용자 ID를 공유할 수 있는 지를 결정하는 것이다.

유저 ID와 파일 접근

디바이스 상에 설치된 각각의 안드로이드 패키지(.apk) 파일에는 그것 자신의 고유한 리눅스 유저 ID가 부여된다. 이것은 패키지에 대한 안전판을 만들며 그것이 다른 애플리케이션를 터치하는 것을 방지한다(또는 다른 애플리케이션이 그것을 터치하는 것을 방지한다). 이 유저 ID는 애플리케이션이 디바이스에 설치될 때 할당되며, 그 디바이스에 그것이 존속하는 동안 변하지 않고 유지된다.

보안에 대한 강제enforcement는 프로세스 레벨에서 이루어지기 때문에, 임의의 두 개의 패키지 내의 코드는 일반적으로 동일한 프로세스에서 실행될 수 없다. 따라서 그것은 서로 다른 리눅스 유저로써 실행되어야 한다. 여러분은 그것들에게 동일한 유저 ID를 부여하기 위해, 각각의 패키지 내의 AndroidManifest.xml의 manifest 태그의 sharedUserId 애트리뷰트를 사용할 수 있다. 이렇게 함으로써, 보안적 측면에서 두 개의 패키지는 동일한 애플리케이션으로 간주되며 동일한 유저 ID와 파일 퍼미션을 갖는다. 보안을 유지하기 위해, 동일한 서명을 가진 (그리고 동일한 sharedUserId를 요청하고 있는) 두 개의 애플리케이션에게만 동일한 유저 ID가 부여된다는 것에 주의하라.

애플리케이션에 의해 저장된 모든 데이터는 그 애플리케이션의 유저 ID가 부여될 것이며, 일반적으로는 다른 패키지들에게는 접근 불가하다. 여러분이 어떤 파일을 getSharedPreferences(String, int), openFileOutput(String,int), open

OrCreateDatabase(String,int,SQLiteDatabase.CursorFactory)를 사용하여 생성할 경우, 여러분은 파일에 대한 읽기/쓰기를 다른 패키지에 허용하기 위해 MODE_WORLD_READABLE 과 MODE_WORLD_WRITEABLE 플래그를 사용할 수 있다. 이러한 플래그를 설정하면 그 파일은 여전히 여러분의 애플리케이션 소유하에 있지만, 다른 애플리케이션이 그것을 볼 수 있도록 그것에 대한 글로벌global 읽기 쓰기 퍼미션이 적절하게 설정된다.

퍼미션 사용하기

기본적인 안드로이드 애플리케이션은 그것과 관련된 어떠한 퍼미션도 가지고 있지 않다. 이것이 의미하는 바는, 그것이 사용자의 경험experience과 디바이스 상의 임의의 데이터에 나쁜 영향을 미칠 어떠한 것도 할 수 없다는 것이다. 디바이스의 보호된 기능을 사용하기 위해서는, 여러분의 AndroidManifest.xml 내에 여러분의 애플리케이션이 필요로 하는 퍼미션을 선언하는 하나 또는 그 이상의 〈uses-permission〉 태그를 포함해야 한다.

예를 들어 SMS 메시지 수신을 모니터링할 필요가 있는 애플리케이션은 다음과 같이 지정할 수 있다.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
</manifest>
```

애플리케이션 설치 시점에, 애플리케이션에 의해 요청된 퍼미션은 해당 퍼미션을 선언하고 있는 애플리케이션의 서명 확인을 기반으로, 그리고(또는) 사용자와의 상호작용에 기반해서 패키지 인스톨러에 의해 그 애플리케이션에게 부여된다. 애플리케이션이 실행되는 동안에는 사용자와의 어떤 확인 절차도 이루어지지 않는다: 애플리케이션이 설치될 때 특정 퍼미션이 부여되어 해당 기능을 원하는 대로 사용할 수 있거나, 또는 그 퍼미션이 부여되지 않아 그 기능을 사용하려는 어떠한 시도도 사용자에게 알리지 않고 실패할 것이다.

종종 퍼미션 실패는 애플리케이션에 보안 예외(SecurityException)가 발생하게 된다. 그러나, 이것은 모든 곳에서 발생하는 것을 보장받지는 못한다. 예를 들

어, sendBroadcast(Intent) 메쏘드는 메쏘드 호출이 리턴된 후 데이터가 각각의 리시버에게 전달될 때 퍼미션들을 체크한다. 그러므로 여러분은 그곳에서 퍼미션 실패가 있더라도 그 예외(Exception)를 수신할 수 없을 것이다. 그러나, 거의 대부분의 경우에 퍼미션 실패는 시스템 로그에 기록될 것이다.

안드로이드 시스템에 의해 제공되는 퍼미션은 Manifest.permission에서 볼 수 있다. 모든 애플리케이션은 자기 자신의 퍼미션을 정의하고 강제할 수 있다. 그러므로 이것은 모든 가능한 퍼미션들을 포함하는 목록은 아니다.

여러분이 프로그램이 오퍼레이션되는 동안에 특정 퍼미션이 여러 곳에서 강제될지도 모른다.

- 시스템 내부로의 호출 시점. 애플리케이션이 특정함수를 수행하는 것을 방지하기 위해.
- 액티비티를 시작하는 시점. 애플리케이션이 다른 애플리케이션의 액티비티를 런치하는 것을 방지하기 위해.
- 브로드캐스트 보내고 받는 시점. 누가 여러분의 브로드캐스트를 수신할 수 있는가를 제어하고, 누가 여러분에게 브로드캐스트를 보낼 수 있는가를 제어하기 위해.
- 컨텐트 프로바이더에 접근해서 오퍼레이션 하는 시점.
- 서비스를 바인딩하거나 시작하는 시점.

퍼미션 선언 및 강제

여러분 자신의 퍼미션을 사용하기 위해서, 여러분은 하나 또는 그 이상의 〈permission〉 태그를 사용해서 AndroidManifest.xml 에 그것들을 먼저 선언해야 한다.

예를 들어 누가 자신의 액티비티들 중의 하나를 시작할 수 있는 가를 제어하길 원하는 애플리케이션은, 아래와 같은 방식으로 이러한 오퍼레이션에 대한 퍼미션을 선언할 수 있다.

```
<manifest
       xmlns:android="http://schemas.android.com/apk/res/android"
       package="com.me.app.myapp" >
    <permission
       android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
       android:label="@string/permlab_deadlyActivity"
       android:description="@string/permdesc_deadlyActivity"
       android:permissionGroup="android.permission-group.COST_MONEY"
       android:protectionLevel="dangerous" />
</manifest>
```

9장. "AndroidManifest.xml"의 permission 엘리먼트(protectionLevel을 보라.)에서 설명되듯이, 사용자가 퍼미션을 요구하는 애플리케이션들을 어떻게 알게 되는지, 또는 해당 퍼미션를 보유하는 것이 누구에게 허용되는지를 시스템에 알려주기 위해 〈protectionLevel〉 애트리뷰트가 요구된다.

〈permissionGroup〉 애트리뷰트는 선택 사항이다. 그것은 단지 시스템이 사용자에게 퍼미션을 표시하는 것을 돕기 위하여 사용된다. 여러분은 일반적으로 (android.Manifest.permission_group 에서 리스트 된) 표준 시스템 그룹을 이곳에 설정하길 원할 수 있으며, 또는 다소 드문 경우지만 스스로 직접 정의한 것을 설정할 수도 있다. 사용자에게 보여지는 퍼미션 UI를 단순화하고자 한다면, 이미 존재하는 퍼미션 그룹을 사용하는 것이 좋다.

퍼미션을 위해서는 라벨label과 설명description 두 가지 모두가 제공되어야 하는 것에 주의하라. 이것은 문자열 리소스로써, android:lable은 사용자가 퍼미션 리스트를 볼 때 표시되며, android:description은 한 개의 퍼미션을 상세히 볼 때 표시된다. 라벨은 퍼미션이 보호하고 있는 기능의 핵심 부분을 몇 개의 단어로 기술하도록 간결해야 한다. 설명descrption은 그 퍼미션을 보유하고 있는 것에게 무엇이 허용되는 지를 설명하는 두 개의 문장이어야 한다. 그 설명description을 기술하는 방법은 두 개의 문장인데, 첫 번째 문장은 퍼미션을 설명하는 것이며, 두 번째 문장은 애플리케이션에 해당 퍼미션이 부여되었을 때 발생할 수 있는 나쁜 것이 무엇인지를 사용자에게 경고하는 것이다.

여기에 CALL_PHONE 퍼미션에 대한 라벨label과 설명description의 예제가 있다.

```
<string name="permlab_callPhone">directly call phone numbers</string>
<string name="permdesc_callPhone">Allows the application to call
        phone numbers without your intervention. Malicious applications may
        cause unexpected calls on your phone bill. Note that this does not
        allow the application to call emergency numbers.</string>
```

여러분은 shell 명령어, adb shell pm list permissions를 사용해서, 시스템에 현재 정의되어 있는 퍼미션들을 볼 수 있다. 특히, '-s' 옵션은 사용자가 퍼미션들을 보는 방식과 대략적으로 비슷한 형식으로 퍼미션을 보여준다.

```
$ adb shell pm list permissions -s
All Permissions:

Network communication: view Wi-Fi state, create Bluetooth connections,
full
Internet access, view network state

Your location: access extra location provider commands, fine (GPS) lo-
cation,
mock location sources for testing, coarse (network-based) location

Services that cost you money: send SMS messages, directly call phone
numbers
...
```

AndroidManifest.xml 에서 퍼미션 강제하기

시스템이나 애플리케이션의 모든 컴포넌트에 대한 접근을 제약하는 상위계층의 퍼미션들은 여러분의 AndroidManifest.xml을 통해서 적용될 수 있다. 이를 위해 요구되는 것은 필요한 컴포넌트에 android:permission 애트리뷰트를 포함하고, 그 컴포넌트에 대한 접근을 제어하기 위해 사용될 퍼미션 이름을 지정하는 것이 전부이다.

〈activity〉 태그에 적용된 액티비티 퍼미션은, 누가 해당 액티비티를 시작할 수 있는 지를 제약한다. 그 퍼미션은 Context.startActivity()와 Activity.startActivityForResult() 함수 내에서 점검된다. 만약 호출자caller가 필요한 퍼미션을 가지고 있지 않으면, 보안예외SecurityException가 호출되는 곳에서 발생된다.

〈service〉 태그에 적용된 서비스 퍼미션은, 누가 해당 서비스를 시작하거나 바인

드할 수 있는가를 제약한다. 퍼미션은 Context.startService(), Contextstop Service() 그리고 Context.bindService() 함수내에서 점검된다. 만약 호출자caller가 필요한 퍼미션을 가지고 있지 않다면, 보안예외SecurityException가 호출되는 곳에서 발생된다.

〈receiver〉 태그에 적용된 보로드캐스트 리시버 퍼미션은, 누가 해당 리시버에게 브로드캐스트를 발송할 수 있는가를 제약한다. 퍼미션은 Context.send Broadcast()가 리턴된 후, 시스템이 제출된 브로드캐스트를 주어진 리시버에 전달하기를 시도할 때 점검된다. 결론적으로, 퍼미션 실패는 호출자caller에게 예외exception 결과를 발생시키지 않는다; 다만 퍼미션 실패는 인텐트를 전달하지 못할 뿐이다. 같은 방식으로, 프로그램적으로 등록된 리시버에게 누가 브로드캐스트를 할 수 있는가를 제어하기 위해 Context.registerReceiver()에 퍼미션을 제공할 수 있다. 다른 방법으로 어떤 브로드캐스트 리시버 오브젝트가 브로드캐스트를 수신하는 것이 허용되는 지를 제약하기 위해, Context.sendBroadcast()를 호출할 때 퍼미션이 제공될 수 있다(아래를 보라).

〈provider〉 태그에 적용된 컨텐트 프로바이더 퍼미션은, 누가 컨텐트 프로로바이더에 있는 데이터에 접근할 수 있는 가를 제약한다(컨텐트 프로바이더는 그들에게 유용한 추가적인 중요한 보안 설비를 가지고 있다. 그것은 URI 퍼미션이라 불리며 나중에 설명된다). 다른 컴포넌트와 달리, 그곳에는 여러분이 설정할 수 있는 두 개의 별도의 퍼미션 애트리뷰트가 있다. android:readPermission은 누가 프로바이더로부터 읽기를 할 수 있는가를 제약하며, android:writePermission은 누가 그것에 쓰기를 할 수 있는가를 제약한다. 만약 프로바이더가 읽기와 쓰기 퍼미션 두 가지로 보호되고 있다면, 쓰기 퍼미션만을 가지고는 여러분이 프로바이더로부터 읽기를 할 수 없다는 것을 의미한다. 퍼미션은 여러분이 처음으로 프로바이더를 얻어올 때 점검되며(만약 여러분이 두 가지 퍼미션 중 어느 하나라도 가지고 있지 않다면, 보안예외(SecurityException)가 발생할 것이다), 그런 다음에 여러분은 프로바이더에 대한 오퍼레이션을 수행한다. ContentResolver.query()를 사용하는 것은 읽기 퍼미션을 가지는 것을 필요로 하며, ContentResolver.insert(), ContentResolver.update(), ContentResolver.delete()를 사용하는 것은 쓰기 퍼미션을 필요로 한다. 이 모든 경우에서, 필요한 퍼미션을 보유하지 않으면 호출하는 곳으로 보안예외SecurityException가 발생하게 된다.

브로드캐스트 발송 시 보안 강제하기

(위에서 기술한 것처럼) 누가 등록된 브로드캐스트 리시버에 인텐트를 발송할 수 있는가를 강제하는 퍼미션에 추가해서, 여러분은 브로드캐스트를 보낼 때 필요한 퍼미션을 또한 지정할 수 있다. 퍼미션 문자열을 가지는 Context.sendBroadcast()를 호출하는 것으로, 여러분은 브로드캐스트를 수신하기 위해서는 리시버 애플리케이션이 그 퍼미션을 보유해야 한다는 것을 요구한다.

리시버와 브로드캐스터 둘 다 퍼미션을 요구할 수 있다는 것에 주목하라. 이것이 일어날 때, 관련된 타겟에 인텐트가 전달되도록 하기 위해서는 양쪽 모두의 퍼미션 점검이 통과pass되어야 한다.

그 밖의 퍼미션 강제

임의의 정교한 퍼미션들이 하나의 서비스 안으로 호출이 일어날 때 강제될 수 있다. 이것은 Context.checkCallingPermission() 메쏘드를 통해 이뤄진다. 요구되는 퍼미션 문자열을 가지고 호출하라. 그러면 그것은 그 퍼미션이 현재 호출하는 프로세스에게 부여되었는지 여부를 가리키는 정수integer를 리턴할 것이다. 이것은 여러분이 다른 프로세스로부터 들어오는 호출을 실행할 때 사용될 수 있을 뿐이라는 것에 주의하라. 외부로부터 들어오는 호출은 보통 서비스에서 배포된 IDL 인터페이스를 통하거나 또는 다른 프로세스에 제공된 몇가지 다른 방법들에 기반한다.

퍼미션을 점검하기 위한 다른 유용한 방법들이 많이 존재한다. 만약 여러분이 다른 프로세스의 pid(프로세스 ID)를 가지고 있다면, 여러분은 그 pid에 대한 퍼미션을 점검하기 위해 컨텍스트 메쏘드 Context.checkPermission(String, int, int)을 이용할 수 있다. 만약 여러분이 다른 애플리케이션의 패키지 이름을 가지고 있다면, 여러분은 특정한 패키지가 특정 퍼미션을 부여받았는 지를 알아 내기 위해 직접 패키지 매니저의 메쏘드인 PackageManager.checkPermission(String, String)을 사용할 수 있다.

URI 퍼미션(URI Permissions)

지금까지 기술된 표준 퍼미션 시스템은 컨텐트 프로바이더를 사용할 때는 가끔 충분치 않다. 컨텐트 프로바이더는 읽기와 쓰기 퍼미션을 가지고 그것 자신을 보호하길 원할 수도 있다. 동시에, 프로바이더의 직접적인 클라이언트 또한 그 안에서 다른 애플리케이션들이 오퍼레이션을 할 수 있도록 그것들에 대한 특정 URI들을 건네줄 필요가 있다. 일반적인 예제가 메일 애플리케이션의 첨부파일이다. 메일에 대한 접근은 퍼미션에 의해 보호되어야 한다. 왜냐하면, 이것은 민감한 사용자 데이터이기 때문이다. 하지만 만약 이미지 첨부파일에 대한 URI가 이미지 뷰어에게 주어진다면, 이미지 뷰어는 모든 이메일에 대한 접근 퍼미션을 가질 이유가 없기 때문에 첨부파일을 오픈하기 위한 퍼미션를 가지지 않을 것이다.

이 문제의 해결책은 per-URI 퍼미션이다. 액티비티를 시작시킬 때나 혹은 결과를 액티비티에 리턴할 때, 그것을 호출한 곳에서 Intent.FLAG_GRANT_READ_URI_PERMISSION 및(또는) Intent.FLAG_GRANT_WRITE_URI_PERMISSION을 설정할 수 있다. 이것은 인텐트를 수신하는 액티비티가 인텐트에 대응되는 컨텐트 프로바이더 내의 데이터 접근 퍼미션을 보유하고 있는지와 상관없이, 그 액티비티에게 그 인텐트 내의 특정 데이터 URI에 대한 접근 퍼미션을 부여한다.

이 메커니즘은 사용자 상호작용(첨부파일을 여는 것, 컨택을 리스트에서 선택하는 것, 기타)이 임시로 정교한 퍼미션을 부여하게 하는 곳에서 일반적인 기능-스타일capability-style 모델을 가능하게 한다. 이것은 애플리케이션들에 의해 필요한 퍼미션을 오직 그것들이 직접적으로 관련된 동작에만 국한하는 핵심 설비가 될 수 있다.

정교한 URI 퍼미션을 부여하는 것은, 어쨌거나 그러한 URI를 보유한 컨텐트 프로바이더와의 약간의 협력을 요구한다. 컨텐트 프로바이더가 이 설비를 구현하고 android:grantUriPermissions 애트리뷰트 또는 〈grant-uri-permissions〉 태그를 통해 그것을 지원한다는 것을 선언하는 것이 강력히 권장된다.

Context.grantUriPermission(), Context.revokeUriPermission(), Context.checkUriPermission() 메쏘드에서 더 많은 정보들을 찾을 수 있다.

Section | 9

AndroidManifest.xml 파일

9.1. 매니페스트 파일 구조
9.2. 파일 규약
9.3. 파일 특성
9.4. 〈action〉
9.5. 〈activity〉
9.6. 〈activity-alias〉
9.7. 〈application〉
9.8. 〈category〉
9.9. 〈data〉
9.10. 〈grant-uri-permission〉
9.11. 〈instrumentation〉
9.12. 〈intent-filter〉
9.13. 〈manifest〉
9.14. 〈meta-data〉
9.15. 〈path-permission〉
9.16. 〈permission〉
9.17. 〈permission-group〉
9.18. 〈permission-tree〉
9.19. 〈provider〉
9.20. 〈receiver〉
9.21. 〈service〉
9.22. 〈supports-screens〉
9.23. 〈uses-configuration〉
9.24. 〈uses-feature〉
9.26. 〈uses-library〉
9.26. 〈uses-permission〉
9.27. 〈uses-sdk〉

9. AndroidManifest.xml 파일

모든 애플리케이션은 그것의 루트 디렉토리에 AndroidManifest.xml 파일(정확하게 그 이름을 가짐)이 있어야만 한다. 매니페스트는 안드로이드 시스템에 애플리케이션에 대한 필수적인 정보를 제공하며, 시스템은 애플리케이션의 코드 중 어떤 것을 실행할 수 있기 전에 그 정보를 가져야 한다. 그 중에서도 특히 매니페스트는 다음과 같은 것들을 한다.

- 그것은 애플리케이션에 대한 Java 패키지를 명시한다. 패키지 이름은 애플리케이션에 대한 고유한 식별자로서 사용된다.
- 그것은 애플리케이션의 컴포넌트를 기술한다 – 애플리케이션을 구성하는 액티비티, 서비스, 브로드캐스트 리시버, 그리고 컨텐트 프로바이더. 그것은 컴포넌트 각각을 구현하는 클래스를 지정하며, 그것들의 기능을 공표한다(예를 들어 그들이 취급할 수 있는 인텐트 메시지). 이러한 선언은 컴포넌트가 무엇이고, 그것들이 어떤 조건 하에서 런치될 수 있는가를 안드로이드 시스템이 알게끔 한다.
- 그것은 어떤 프로세스가 애플리케이션 컴포넌트를 보유할 것인가를 결정한다.
- 그것은 API의 보호된 영역에 접근하기 위해, 그리고 다른 애플리케이션과 상호작용하기 위해 애플리케이션이 가지고 있어야 하는 퍼미션이 어떤 것인가를 선언한다.
- 그것은 또한 애플리케이션의 컴포넌트와 상호작용하기 위해, 다른 애플리케이션이 가지도록 요구되는 퍼미션들을 선언한다.
- 그것은 애플리케이션이 실행되는 동안의 프로파일링과 다른 정보를 제공하는 인스트루멘테이션instrumentation 클래스들을 나열한다. 이러한 선언들은 애플리케이션이 개발되어 테스트되는 동안에만 매니페스트 안에 존재한다. 그것들은 애플리케이션이 배포되기 전에 제거된다.
- 그것은 애플리케이션이 요구하는 Android API의 최소 레벨을 선언한다.
- 그것은 애플리케이션에 링크되어야 할 라이브러리를 나열한다.

매니페스트 파일 구조

아래의 다이어그램은 매니페스트 파일의 일반적인 구조와 그것이 포함할 수 있는 모든 엘리먼트를 보여준다. 엘리먼트의 모든 애트리뷰트와 함께 각각의 엘리먼트들이 별도의 파일로 전부 문서화되어 있다. 어떤 엘리먼트에 대한 상세한 정보를 보기 위해서는, 아래 다이어그램 다음에 오는 알파벳 순서의 엘리먼트 리스트에 있는 엘리먼트 이름, 또는 엘리먼트 이름에 대한 또 다른 언급들을 살펴보라.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
```

```
        <provider>
            <grant-uri-permission />
            <path-permission />
            <meta-data />
        </provider>
        <uses-library />
    </application>
</manifest>
```

매니페스트 파일에서 나타날 수 있는 모든 엘리먼트는 알파벳 순서로 아래에 나열되어 있다. 이것들이 유일하게 허용되는 엘리먼트들이다. 여러분 자신의 엘리먼트 또는 애트리뷰트를 추가할 수는 없다.

```
<action>
<activity>
<activity-alias>
<application>
<category>
<data>
<grant-uri-permission>
<instrumentation>
<intent-filter>
<manifest>
<meta-data>
<path-permission />
<permission>
<permission-group>
<permission-tree>
<provider>
<receiver>
<service>
<supports-screens>
<uses-configuration>
<uses-feature>
<uses-library>
<uses-permission>
<uses-sdk>
```

파일 규약

약간의 규약과 규칙이 매니페스트 안의 모든 엘리먼트와 애트리뷰트에 일반적으로 적용된다.

엘리먼트(Elements)

오직 〈manifest〉와 〈application〉 엘리먼트만이 필수사항이다. 그것들 각각은 존재해야 하며, 오직 한 번 있어야 한다. 다른 것들 대부분은 여러 번 있을 수 있거나 또는 전혀 없을 수도 있다. – 그럼에도 불구하고, 적어도 그것들 중 몇 개는 매니페스트가 의미를 가지기 위해서는 그곳에 존재해야 한다.

만약 엘리먼트가 무언가를 포함한다면, 그것은 다른 엘리먼트를 포함한다. 모든 값은 그 엘리먼트 안의 CDATA(character data)가 아닌, 애트리뷰트를 통해 설정된다.

동일한 레벨에 있는 엘리먼트는 일반적으로 순서가 없다. 예를 들어 〈activity〉, 〈provider〉 그리고 〈service〉 엘리먼트는 어떤 어떤 순서로든 섞일 수 있다(〈activity-alias〉 엘리먼트는 이 규칙에 대해 예외다. 그것은 앨리어스(alias) 하는 〈activity〉 뒤에 있어야 한다).

애트리뷰트(Attributes)

형식적인 면에서 모든 애트리뷰트는 선택 사항이다. 하지만 엘리먼트가 그것의 목적을 달성하기 위해 지정되어야 하는 것들이 있다. 가이드로써 이 문서를 사용하라. 선택 사항인 애트리뷰트에 대해서, 그것은 그 명세가 없을 때 발생하는 디폴트 값 또는 상태를 언급한다.

루트 〈manifest〉 엘리먼트의 몇 개의 애트리뷰트를 제외하고, 모든 애트리뷰트 이름은, 예를 들어 android:alwaysRetainTaskState와 같이 android: 접두어를 가지고 시작한다. 그 접두어가 보편적이기 때문에, 이 문서에서는 이름에 의해 애트리뷰트를 참조할 때 일반적으로 그것을 생략한다.

클래스 이름 선언하기

많은 엘리먼트들은 Java 오브젝트에 대응한다. 그것은 애플리케이션 자체를 위한 엘리먼트(〈application〉 엘리먼트)와 그것의 주요 컴포넌트인 액티비티(〈activity〉), 서비스(〈service〉), 브로드캐스트 리시버(〈receiver〉), 그리고 컨텐트 프로바이더(〈provider〉)를 포함한다.

만약 여러분이 서브클래스를 정의한다면, 여러분은 거의 항상 컴포넌트 클래스(액티비티, 서비스, 브로드캐스트 리시버, 그리고 컨텐트 프로바이더)에 대한 것일 것이고, 그 서브클래스는 name 애트리뷰트를 통해 선언된다. 그 name은 전체 패키지 이름을 포함해야 한다. 예를 들어 서비스 서브클래스는 다음과 같이 선언될 수 있다.

```
<manifest . . . >
  <application . . . >
     <service android:name="com.example.project.SecretService" . . . >
          . . .
     </service>
       . . .
  </application>
</manifest>
```

하지만 약칭으로 만약 문자열의 첫 번째 문자가 마침표라면, 그 문자열은 애플리케이션의 패키지 이름에게 덧붙여진다(〈manifest〉 엘리먼트의 package 애트리뷰트에 의해 지정된 대로). 아래에 나오는 선언은 위에 있는 것과 같은 것이다.

```
<manifest package="com.example.project" . . . >
    <application . . . >
        <service android:name=".SecretService" . . . >
            . . .
        </service>
        . . .
    </application>
</manifest>
```

컴포넌트를 시작할 때 안드로이드는 지명된 서브클래스의 인스턴스를 생성한다. 만약 서브클래스가 지정되지 않았다면, 그것은 베이스 클래스의 인스턴스를 생성한다.

복수의 값(Multiple values)

만약 하나 이상의 값이 지정될 수 있다면, 엘리먼트 내에서 여러 개의 값을 나열하기 보다는 거의 항상 엘리먼트들이 반복된다. 예를 들어 인텐트 필터는 몇 개의 액션을 나열할 수 있다.

```
<intent-filter . . . >
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.INSERT" />
    <action android:name="android.intent.action.DELETE" />
    . . .
</intent-filter>
```

리소스 값(Resource values)

어떤 애트리뷰트들은 사용자에게 보여질 수 있는 값을 가지고 있다 – 예를 들어 액티비티에 대한 label과 icon이 그것이다. 이러한 애트리뷰트들의 값은 로컬라이즈localize 되어야 하며, 그러므로 리소스 또는 테마로 설정된다. 리소스 값은 다음과 같은 포맷으로 표현된다,

@[package:]type:name

그곳에서 패키지 이름은 만약 그 리소스가 그 애플리케이션과 같은 패키지에 있다면 생략될 수 있다. 타입은 "string" 또는 "drawable" 과 같은 리소스의 타입이며, name은 특정 리소스를 구분하는 이름이다. 예를 들어 다음과 같다.

〈activity android:icon="@drawable/smallPic" . . . 〉

테마 값도 비슷한 방법으로 표현된다. 그러나 이니셜 '@'가 아닌 '?'를 가진다.

?[package:]type:name

문자열 값(String values)

애트리뷰트 값이 문자열인 곳에서, 두 개의 백슬래시('\\')는 이스케이프escape 문자를 위해 사용되어야 한다. 예를 들어 뉴라인newline에 대한 '\\n' 또는 유니코드 문자에 대한 '\\uxxxx' 이 있다.

파일 특징

다음의 섹션은 몇가지 안드로이드 특징들이 매니페스트 파일에 어떻게 반영되어 있는지를 기술한다.

인텐트 필터(Intent Filters)

애플리케이션의 핵심 컴포넌트(그것의 액티비티, 서비스, 그리고 브로드캐스트 리시버)는 인텐트에 의해 활성화된다. 인텐트는 원하는 액션을 설명하는 정보의 묶음(인텐트 오브젝트)이다. 그것은 그것에서 다루어져야 하는 데이터, 액션을 수행해야 하는 컴포넌트의 카테고리, 그리고 다른 관련된 명령들을 포함한다. 안드로이드는 인텐트에 응답하는 적절한 컴포넌트를 찾는다. 만약 신규 인스턴스가 필요하다면 컴포넌트의 신규 인스턴스를 런치하며, 그것에 인텐트 오브젝트를 전달한다.

컴포넌트들은 인텐트 필터를 통해 그것들의 기능을 – 그것들이 응답할 수 있는 인텐트들의 종류 – 통지한다. 안드로이드 시스템은 컴포넌트를 런치하기 전에 컴포넌트가 취급할 수 있는 인텐트가 어떤 것인가를 알아야 하기 때문에, 인텐트 필터는 〈intent-filter〉 엘리먼트로써 매니페스트 안에 지정되어진다. 컴포넌트는 몇 개의 필터든 가질 수 있으며, 그것들 각각은 다른 기능을 기술한다.

명시적으로explicitly 타겟 컴포넌트를 지명하는 인텐트는 그 컴포넌트를 활성화할 것이다; 그 필터는 역할을 담당하지 않는다. 그러나 이름에 의해 타겟을 지정하지 않은 인텐트는 오직 그것이 해당 컴포넌트의 필터들 중 하나를 통과했을 때만 컴포넌트를 활성화할 수 있다.

인텐트 오브젝트가 인텐트 필터에 대해 어떻게 테스트되는가에 대한 정보에 대해서는 5장. “인텐트와 인텐트 필터”를 보라.

아이콘과 라벨(Icons and Labels)

다수의 엘리먼트들은 사용자에게 표시될 작은 아이콘과 텍스트 라벨을 위한 icon과 label 애트리뷰트를 가진다. 일부는 또한 스크린 상에 보여질 수 있는 더 긴 설명적인 텍스트를 위한 description 애트리뷰트를 가진다. 예를 들어 〈permission〉 엘리먼트는 그것을 요구했던 애플리케이션에게 퍼미션을 부여할지 여부를 사용자에게 물을 때, 퍼미션을 나타내는 icon, 퍼미션의 name, 그리고 그것이 일으킬 수 있는 것에 대한 description이 사용자에게 모두 표시될 수 있도록 이러한 세 가지 애트리뷰트 모두를 가진다.

모든 경우에 있어서, 어떤 것을 포함하고 있는 엘리먼트에 있는 icon과 label 집

합set은 그 컨테이너의 서브엘리먼트들 모두에 대한 디폴트 icon과 label 설정이 될 수 있다. 따라서, 〈application〉 엘리먼트에서의 icon과 label 집합set은 애플리케이션의 컴포넌트들 각각에 대한 디폴트 icon과 label이다. 비슷하게 하나의 컴포넌트, 예를 들어 하나의 〈activity〉 엘리먼트에 대한 icon과 label 집합set은 그 컴포넌트의 〈intent-filter〉 엘리먼트에 대한 디폴트 설정이다. 만약 애플리케이션이 하나의 label을 설정하지만 액티비티와 그것의 인텐트 필터는 label을 설정하지 않는다면, 그 애플리케이션 label은 액티비티와 그 인텐트 필터 모두에 대한 label로써 취급되어진다.

인텐트 필터에 대한 icon과 label 집합set은 컴포넌트가 사용자에게 제공될 때마다 컴포넌트가 필터에 의해 통보된 기능을 수행한다는 것을 나타내기 위해 사용된다. 예를 들어 "android.intent.action.MAIN"와 "android.intent.category.LAUNCHER" 설정을 가지는 필터는 그것이 애플리케이션을 시작시키는 액티비티, 즉 그것이 애플리케이션 런처에 표시되어져야 하는 액티비티임을 공표한다. 그러므로 그 필터 안의 icon과 label 집합set은 런처에서 보여지는 것들이 된다.

퍼미션(Permissions)

퍼미션은 코드의 일부나 디바이스 상의 데이터에 대한 접근을 제한하는 제약이다. 그 제한은 사용자 경험을 왜곡하거나 훼손시키기 위해 악용될 수 있는 중요한 데이터와 코드를 보호하기 위해 부과된다.

각 퍼미션은 하나의 고유한 label에 의해 구분된다. 종종 label은 제약된 액션을 가리킨다. 예를 들어 안드로이드에 의해 정의된 약간의 퍼미션이 여기 있다:

```
android.permission.CALL_EMERGENCY_NUMBERS
android.permission.READ_OWNER_DATA
android.permission.SET_WALLPAPER
android.permission.DEVICE_POWER
```

하나의 기능은 많아야 하나의 퍼미션에 의해 보호될 수 있다.

애플리케이션이 퍼미션에 의해 보호된 기능에 접근을 필요로 한다면, 그것은 매니페스트 안에 〈uses-permission〉 엘리먼트를 가지고 그 퍼미션을 필요로 한다는

것을 선언해야 한다. 그런 다음에, 애플리케이션이 디바이에서 설치될 때 인스톨러 installer는 애플리케이션의 인증서를 사인한 당국authorities을 체크하고, 그리고 어떤 경우에는 사용자에게 물어봄으로써 요청된 퍼미션에 대한 부여 여부를 결정한다. 만약 퍼미션이 부여된다면, 애플리케이션은 보호된 기능을 사용할 수 있다. 만일 그렇지 않다면, 그러한 기능에 접근하고자 하는 시도는 사용자에게 어떤 통보도 없이 간단하게 실패할 것이다.

애플리케이션은 또한 그것 자신의 컴포넌트(액티비티, 서비스, 브로드캐스트 리시버, 그리고 컨텐트 프로바이더)들을 퍼미션을 가지고 보호할 수 있다. 그것은 (android.Manifest.permission에서 나열된) 안드로이드에 의해 정의되거나 또는 다른 애플리케이션에 의해 선언된 퍼미션들 중 일부를 이용할 수 있다. 또는 그것은 그것 자신의 퍼미션을 정의할 수 있다. 새로운 퍼미션은 〈permission〉 엘리먼트를 사용해서 선언된다. 예를 들어 액티비티는 다음과 같이 보호될 수 있다:

```
<manifest . . . >
    <permission android:name="com.example.project.DEBIT_ACCT" . . . />
    . . .
    <application . . .>
        <activity android:name="com.example.project.FreneticActivity" . . . >
                  android:permission="com.example.project.DEBIT_ACCT"
                  . . . >
             . . .
        </activity>
    </application>
    . . .
    <uses-permission android:name="com.example.project.DEBIT_ACCT" />
    . . .
</manifest>
```

이 예제에서, DEBIT_ACCT 퍼미션은 〈permission〉 엘리먼트를 가지고 선언되었을 뿐만 아니라, 그것의 사용 또한 〈uses-permission〉 엘리먼트를 가지고 요청된다는 것에 주의하라. 그 퍼미션이 애플리케이션 스스로에 의해 부과되었다 하더라도, 애플리케이션의 다른 컴포넌트가 보호된 액티비티를 런치하기 위해서 그것의 사용이 요청되어야 한다.

만약 같은 예제에서, 퍼미션 애트리뷰트가 다른 어디에서 선언된 퍼미션으로 설정

되었다면 (android.permission.CALL_EMERGENCY_NUMBERS와 같이), 〈permission〉 엘리먼트를 가지고 그 퍼미션을 다시 선언하는 것은 필요하지 않다. 그러나 여전히 〈uses-permission〉을 사용해서 그 퍼미션의 사용을 요청하는 것이 필요하다.

〈permission-tree〉 엘리먼트는 코드에서 정의될 퍼미션 그룹에 대한 네임스페이스(namespace)를 선언한다. 그리고 〈permission-group〉는 퍼미션(〈permission〉 엘리먼트를 가지고 매니페스트에서 선언된 것들과 다른 곳에서 선언된 것들 모두) 집합set에 대한 label을 정의한다. 그것은 단지 사용자에게 표시될 때, 퍼미션들이 어떻게 그룹화되는 지에만 영향을 미친다. 〈permission-group〉 엘리먼트는 어떤 퍼미션이 그룹에 속하는지를 지정하지 않는다; 그것은 단지 그룹에 이름을 부여한다. 퍼미션은 〈permission〉 엘리먼트의 permissionGroup 애트리뷰트에 그룹 이름을 할당함으로써 해당 그룹 내에 배치된다.

라이브러리(Libraries)

모든 애플리케이션은 애플리케이션을 만들기 위한 기본적인 패키지(액티비티, 서비스, 인텐트, 뷰, 버튼, 애플리케이션, 컨텐트프로바이더, 그리고 기타 등등과 같은 보편적 클래스들을 가짐)를 포함하는 디폴트 안드로이드 라이브러리에 링크된다.

하지만 어떤 패키지들은 그들 자신의 라이브러리 내에 존재한다. 만약 여러분의 애플리케이션이 이러한 패키지들 중 어떤 것으로부터의 코드를 사용한다면, 그것은 명시적으로 그 패키지들을 링크하도록 요구해야 한다. 매니페스트는 라이브러리의 각각을 지명하는 별도의 〈uses-library〉 엘리먼트를 포함해야 한다(라이브러리 이름은 패키지에 대한 문서에서 찾을 수 있다).

〈action〉 API Level 1

문법:

```
<action android:name="string" />
```

상위 엘리먼트:

```
<intent-filter>
```

설명:

인텐트 필터에 액션을 추가한다. 〈intent-filter〉 엘리먼트는 하나 또는 그 이상의 〈action〉 엘리먼트를 포함해야 한다. 만약 그것이 아무 것도 포함하지 않는다면, 어떤 인텐트 오브젝트도 필터를 통해 얻어지지 않을 것이다. 인텐트 필터와 필터 내부의 액션 명세의 역할에 대한 상세 내용에 대해서는 5장. "인텐트와 인텐트 필터"를 보라.

애트리뷰트:

android:name

액션의 이름. 어떤 표준 액션들은 ACTION_문자열 상수로써 인텐트 클래스에서 정의된다. 이 애트리뷰트에 이러한 액션 중 하나를 할당하기 위해서는, ACTION_ 뒤에 나오는 문자열 앞에 "android.intent.action."을 추가하라. 예를 들어 ACTION_MAIN에 대해서는, "android.intent.action.MAIN"를 사용하고, ACTION_WEB_SEARCH에 대해서는, "android.intent.action.WEB_SEARCH를 사용하라.

여러분이 정의하는 액션에 대해, 유일함을 보장하기 위해 접두어로 패키지 이름을 사용하는 것이 최선이다. 예를 들어 TRANSMOGRIFY 액션은 다음과 같이 지정될 수 있다.:

```
<action android:name="com.example.project.TRANSMOGRIFY" />
```

도입 :

API 레벨 1

더 보기 :

```
<intent-filter>
```

〈activity〉 API Level 1

문법:

```
<activity android:allowTaskReparenting=["true" | "false"]
      android:alwaysRetainTaskState=["true" | "false"]
      android:clearTaskOnLaunch=["true"" | "false"]
      android:configChanges=[one or more of: "mcc" "mnc" "locale"
                  "touchscreen" "keyboard" "keyboardHidden"
                  "navigation" "orientation" "fontScale"]
      android:enabled=["true" | "false"]
      android:excludeFromRecents=["true" | "false"]
```

```
        android:exported=["true" | "false"]
        android:finishOnTaskLaunch=["true" | "false"]
        android:icon="drawable resource"
        android:label="string resource"
        android:launchMode=["multiple" | "singleTop" |
                "singleTask" | "singleInstance"]
        android:multiprocess=["true" | "false"]
        android:name="string"
        android:permission="string"
        android:process="string"
        android:screenOrientation=["unspecified" | "user" | "behind" |
                                   "landscape" | "portrait" |
                                   "sensor" | "nonsensor"]
        android:stateNotNeeded=["true" | "false"]
        android:taskAffinity="string"
        android:theme="resource or theme"
        android:windowSoftInputMode=[one or more of: "stateUnspecified"
                                     "stateUnchanged" "stateHidden"
                                     "stateAlwaysHidden" "stateVisible"
                                     "stateAlwaysVisible" "adjustUnspecified"
                                     "adjustResize" "adjustPan"] >
    . . .
    </activity>
```

상위 엘리먼트:

```
<application>
```

하위 엘리먼트:

```
<intent-filter>
<meta-data>
```

설명:

애플리케이션의 비쥬얼 사용자 인터페이스의 일부를 구현하는 액티비티(Activity 서브클래스)를 선언한다. 모든 액티비티는 매니페스트 파일에 있는 〈activity〉 엘리먼트에 의해 표현되어야 한다. 그곳에 선언되지 않은 어떤 것도 시스템에 의해 보여지지 않을 것이고 결코 실행되지 않을 것이다.

애트리뷰트:

android:allowTaskReparenting

액티비티가 그것을 시작했던 태스크로 부터 그것과 친화력affinity이 있는 태스

크로, 그 태스크가 다음번에 프론트로 오게 될 때 이동할 수 있는지 여부. – 만약 이동할 수 있다면, "참true"이고, 그것이 시작되었던 태스크에 남아있어야 한다면 "거짓false" 이다.

만약 이 애트리뷰트가 설정되지 않는다면, 〈application〉 엘리먼트의 해당 allowTaskReparenting 애트리뷰트에 의해 설정된 값이 그 액티비티에 적용된다. 디폴트 값은 "거짓false"이다.

보통 액티비티가 시작될 때, 그것은 그것을 시작했던 액티비티의 태스크와 결합되며, 그것은 전체 생명주기 동안 그곳에 머무른다. 여러분은 그것의 현재 태스크가 더 이상 보여지지 않을 때, 그것이 친화력을 갖는 태스크로 부모를 바꾸게 하기 위해 이 애트리뷰트를 사용할 수 있다. 전형적으로, 그것은 애플리케이션의 액티비티가 그 애플리케이션과 결합된 메인 태스크로 이동하게 하기 위해 사용된다.

예를 들어 이메일 메시지가 웹 페이지에 대한 링크를 포함한다면, 링크를 클릭하는 것은 그 페이지를 표시할 수 있는 액티비티를 가져온다. 그 액티비티는 브라우저 애플리케이션에 의해 정의된다. 하지만 이메일 태스크의 일부로써 런치된다. 만약 그것이 브라우저 태스크로 부모를 바꾸게 된다면, 브라우저가 다음에 프론트로 오게 될 때 그것은 보여질 것이고, 이메일 태스크가 다시 앞으로 나올 때 사라질 것이다.

액티비티의 친화력affinity는 taskAffinity 애트리뷰트에 의해 정의된다. 태스크의 친화력은 그것의 루트 액티비티의 친화력을 읽음으로써 결정된다. 그러므로 정의된 대로 루트 액티비티는 같은 친화력을 가진 태스크 안에 항상 있다. "singleTask" 또는 "singleInstance" 런치 모드를 가지는 액티비티는 오직 태스크의 루트에만 있을 수 있기 때문에, 부모를 바꾸는 것은 "standard"과 "singleTop" 모드에 제한된다(launchMode 애트리뷰트를 보라).

android:alwaysRetainTaskState

액티비티가 존재하는 태스크의 상태가 시스템에 의해 항상 유지되어질 것인가 여부. – 만약 그것이 존재할 것이라면 "참true". 만약 시스템이 특정 상황에서 그것의 초기 상태로 태스크를 재설정하는 것이 허용된다면 "거짓false". 디폴트 값은 "거짓false". 이 애트리뷰트는 태스크의 루트 액티비티에 대해서만 의미를 가진다; 그것은 다른 모든 액티비티에 대해선 무시된다.

보통 시스템은 사용자가 홈스크린으로부터 그 태스크를 재선택할 때, 특정 상황에서 그 태스크를 비운다(루트 액티비티 위에 있는 스택의 모든 액티비티들을 제거한다). 전형적으로 이것은 사용자가 30분 가량의 일정 시간 동안 그 태스크를 방문하지 않을 때 행하여진다.

하지만 이 애트리뷰트가 "참true" 일 때, 사용자들이 어떻게 그 태스크에 왔는지와 상관없이 해당 태스크의 마지막 상태로 항상 돌아갈 것이다. 예를 들어 이것은 웹 브라우저와 같은 애플리케이션에서 사용자가 잃고 싶지 않는 (다양하게 열린 탭과 같은) 많은 상태가 있을 때 유용하다.

android:clearTaskOnLaunch

태스크가 홈 스크린에서 다시 런치될 때, 루트 액티비티를 제외한 모든 액티비티가 태스크에서 제거될 것인지 여부. – 만약 태스크가 그것의 루트 액티비티까지 항상 벗겨진다면 "참true" 그리고 아니라면 "거짓false". 디폴트 값은 "거짓false". 이 애트리뷰트는 신규 태스크를 시작하는 액티비티(루트 액티비티)에 대해서만 의미가 있다; 그 태스크 안의 다른 모든 액티비티에 대해서는 무시된다.

값이 "참true"일 때, 사용자가 태스크를 다시 시작할 때마다 그 태스크에서 그들이 마지막으로 했던 것이 무엇이든 상관없이, 그리고 그들이 그 태스크를 마지막으로 떠나기 위해 BACK 또는 HOME을 사용했던 것과 상관없이, 그들은 태스크의 루트 액티비티로 보내진다. 값이 "거짓false"일 때, 태스크는 어떤 상황(alwaysRetainTaskState 애트리뷰트를 보라.)에서 액티비티를 비우게 될 수도 있다. 하지만 항상은 아니다..

예를 들어 누군가가 홈스크린으로부터 액티비티 P를 런치하고, 거기에서 액티비티 Q로 간다고 가정하자. 사용자가 다음에 HOME을 누르고, 그런 다음에 액티비티 P로 돌아간다. 보통 그 사용자는 액티비티 Q를 볼 것이다. 왜냐하면 그것이 그들이 P의 태스크에서 마지막으로 하고 있었던 것이기 때문이다. 하지만, 만약 P가 이 플래그를 "참true"으로 설정한다면, 그것 위에 있는 모든 액티비티들(이 경우에는 Q)은 사용자가 HOME을 눌러서 그 태스크가 백그라운드로 갔을 때 제거된다. 그러므로 사용자가 그 태스크로 돌아올 때 오직 P만 보게 된다.

이 애트리뷰트와 allowTaskReparenting 모두가 "참true"이라면, 부모를 바꿀 수 있는 모든 액티비티들은 그들이 친화력을 공유하는 태스크로 이동된다; 위에서 처럼, 남겨진 액티비티는 이후 누락된다.

android:configChanges

액티비티가 스스로가 처리할 환경설정configuration 변화를 나열한다. 나열되지 않은 변화가 일어날 때, 액티비티는 종료되고 다시 시작된다. 나열된 변화가 일어날 때, 액티비티는 실행상태를 유지하고 그것의 onConfigurationChanged() 메쏘드가 호출된다.

다음에 나오는 문자열 중 어떤 것 또는 전부는 이 애트리뷰트를 설정하기 위해 사용될 수 있다. 값은 '|'에 의해 구분된다 – 예를 들어, "locale|navigation|orientation".

값	설명
"mcc"	IMSI 모바일 국가 코드(MCC)가 변경되었다. – 즉, SIM이 인식되었고, 그 MCC를 업데이트했다.
"mnc"	IMSI 모바일 네트워크 코드(MNC)가 변경되었다 – 즉, SIM이 인식되었고, 그 MNC를 업데이트했다.
"locale"	로케일이 변경되었다. – 예를 들어 사용자가 텍스트가 표시되어야 하는 새로운 언어를 선택했다.
"touchscreen"	터치스크린이 변경되었다. (이것은 결코 일반적으로 발생하지는 않는다.)
"keyboard"	키보드 타입이 변경되었다 – 예를 들어 사용자가 외부 키보드를 연결했다.
"keyboardHidden"	키보드 접근성이 변경되었다 – 예를 들어 사용자가 슬라이드 키보드가 보이도록 열었다.
"navigation"	네비게이션 타입이 변경되었다. (이것은 결코 일반적으로 발생하지 않는다.)
"orientation"	스크린 오리엔테이션이 변경되었다 – 즉, 사용자는 디바이스를 회전했다.
"fontScale"	폰트 크기 요소가 변경되었다 – 즉, 사용자는 새로운 글로벌 폰트사이즈를 선택했다.

이러한 모든 환경설정 변화는 애플리케이션에 의해 보여지는 리소스 값들에 영향을 줄 수있다. 그러므로 onConfigurationChanged()가 호출될 때, 정확하게 변화를 처리하기 위해 모든 리소스(뷰 레이아웃, 드로어블, 그리고 기타등등을 포함)를 다시 가져오는 것이 일반적으로 필요하게 될 것이다.

android:enabled

시스템에 의해 액티비티가 인스턴스화 될 수 있는 지 여부. – 그것이 될 수 있다면 "참true" 아니라면 "거짓false"이다. 디폴트 값은 "참true"이다.

〈application〉 엘리먼트는 액티비티를 포함하는 모든 애플리케이션 컴포넌트들에 적용되는 그것 자신의 enabled 애트리뷰트를 가진다. 〈application〉과 〈activity〉의 애트리뷰트는 시스템이 액티비티를 인스턴스화 할 수 있도록 하기 위해 둘 다 "참true"이 되어야 한다(디폴트로 둘 다 "참true"이다). 만약 어느

쪽이든 "거짓false"이라면, 그것은 인스턴스화 될 수 없다.

android:excludeFromRecents

액티비티가 사용자에게 표시될 수 있는 최근에 런치된 액티비티의 리스트에서 제외되어야 하는지 여부 – 만약 그것이 제외되어야 한다면 "참true"이고, 그것이 포함되어져야 한다면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

android:exported

액티비티가 다른 애플리케이션의 컴포넌트에 의해 런치될 수 있는지 여부.– 만약 그것이 될 수 있다면 "참true" 아니라면 "거짓false". 만약 "거짓false"이라면, 액티비티는 같은 애플리케이션 또는 같은 유저 ID를 가진 애플리케이션의 컴포넌트에 의해서만 런치될 수 있다.

디폴트 값은 액티비티가 인텐트 필터를 포함하는 지 여부에 달려 있다. 어떤 필터도 없다는 것은 액티비티가 그것의 정확한 클래스 이름을 명시하는 것에 의해서만 호출될 수 있다는 것을 의미한다. 이것은 액티비티가 애플리케이션 내부용으로만 의도되었다는 것을 의미한다(왜냐하면 다른 것들은 클래스 이름을 모를 것이기 때문이다). 그러므로 이런 경우에, 디폴트 값은 "거짓false"이다. 반면에, 적어도 하나의 필터가 존재한다는 것은 액티비티가 외부 사용을 위해 의도되었다는 것을 의미한다. 그래서 디폴트 값은 "참true"이다.

이 애트리뷰트가 다른 애플리케이션에게 액티비티의 노출을 제한하는 유일한 방법은 아니다. 여러분은 액티비티를 호출할 수 있는 외부 엔티티를 제한하는 퍼미션을 또한 사용할 수 있다(permission 애트리뷰트를 보라).

android:finishOnTaskLaunch

사용자가 다시 그것의 태스크(홈스크린에서 태스크를 선택한다)을 런치할 때마다 액티비티의 기존 인스턴스가 종료되어야 하는 지 여부 – 만약 종료되어야 한다면 "참true" 아니라면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

만약 이 애트리뷰트와 allowTaskReparenting 모두가 "참true"이라면, 이 애트리뷰트가 다다른 것에 우선한다. 그 액티비티의 친화력은 무시된다. 액티비티는 부모를 바꾸지 않고 파괴된다.

android:icon

액티비티를 표시하는 아이콘. 액티비티에 대한 표시가 스크린 상에 요구될 때 아이콘이 사용자에게 보여진다. 예를 들어 태스크를 시작시키는 액티비티에

대한 아이콘은 런처 윈도우에서 보여진다. 아이콘은 종종 라벨을 수반한다(label 애트리뷰트를 보라).

이 애트리뷰트는 이미지 정의를 포함하는 드로어블drawable 리소스에 대한 레퍼런스로써 설정되어야 한다. 만약 그것이 설정되지 않으면, 전체 애플리케이션을 위해 지정된 아이콘이 대신 사용된다(〈application〉 엘리먼트의 icon 애트리뷰트를 보라).

액티비티의 아이콘 – 여기서 설정되던 또는 〈application〉 엘리먼트에 의해 설정되던 – 은 또한 모든 액티비티의 인텐트 필터를 위한 디폴트 아이콘이다(〈intent-filter〉 엘리먼트 icon 애트리뷰트를 보라).

android:label

액티비티에 대해 사용자가 읽을 수 있는 라벨. 라벨은 액티비티가 사용자에게 표시되어져야 할 때 화면 상에 보여진다. 그것은 액티비티 아이콘과 함께 자주 보여진다.

만약 이 애트리뷰트가 설정되지 않으면, 전체 애플리케이션에 대한 라벨 설정이 대신 사용된다(〈application〉 엘리먼트의 label 애트리뷰트를 보라).

액티비티의 라벨 – 여기서 설정되던 또는 〈application〉 엘리먼트에 의해 설정되던 – 은 또한 모든 액티비티의 인텐트 필터에 대한 디폴트 라벨이다(〈intent-filter〉 엘리먼트의 label 애트리뷰트를 보라).

라벨은 사용자 인터페이스에 있는 다른 문자열 처럼 로컬라이즈localize될 수 있도록 하기 위해 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다. 하지만 여러분이 애플리케이션을 개발하는 동안 편의상, 그것은 원시 문자열로 또한 설정될 수 있다.

android:launchMode

액티비티가 어떻게 런치되어야 하는 지에 대한 명령. 인텐트를 처리하기 위해 액티비티가 호출될 때 무엇이 발생해야 하는 지를 결정하기 위해, 인텐트 오브젝트의 액티비티 플래그(FLAG_ACTIVITY_* 상수)와 함께 동작하는 네 가지 모드가 있다. 그것은 다음과 같다.

```
"standard"
"singleTop"
"singleTask"
```

"singleInstance"

디폴트의 모드는 "standard"이다

그 모드들은 두 가지 주요 그룹으로 나뉘어 진다. 한쪽에는 "standard" 와 "singleTop" 액티비티들이 있으며, 다른 한쪽에는 "singleTask" 와 "singleInstance"가 있다. "standard" 또는 "singleTop" 런치 모드를 가지는 액티비티는 여러 번 인스턴스화 될 수 있다. 인스턴스는 어떤 태스크에도 속할 수 있으며, 그리고 액티비티 스택의 어느 곳에든 놓여질 수 있다. 일반적으로 그것들은 startActivity()를 호출한 태스크에서 런치된다(인텐트 오브젝트가 FLAG_ACTIVITY_NEW_TASK 명령을 포함하지 않는 한 그러하며, 이 경우에는 다른 태스크가 선택되어 진다 – taskAffinity 애트리뷰트를 보라).

그에 반하여, "singleTask" 와 "singleInstance" 액티비티들은 오직 하나의 태스크를 시작할 수 있다. 그것들은 항상 액티비티 스택의 루트에 위치한다. 더욱이, 디바이스는 한 번에 – 오직 하나의 그 태스크에서 오직 하나의 액티비티 인스턴스를 보유할 수 있다.

"standard" 와 "singleTop" 모드는 단지 한 측면에서만 서로 차이가 있다: "standard" 액티비티를 위한 신규 인텐트가 있을 때마다, 신규 클래스 인스턴스가 그 인텐트에 응답하기 위해 생성된다. 각각의 인스턴스는 하나의 인텐트를 처리한다. 비슷하게 "singleTop" 액티비티에 대한 신규 인스턴스 또한 신규 인텐트를 처리하기 위해 생성될 수 있다. 하지만 타겟 태스크가 이미 그것의 스택의 최상위에 그 액티비티에 대한 인스턴스를 가지고 있다면, 그 인스턴스가 신규 인텐트를 수신할 것이다(onNewIntent() 호출에서); 신규 인스턴스는 생성되지 않는다. 다른 상황에서는 – 예를 들어 만약 "singleTop" 액티비티에 대한 이미 존재하는 인스턴스가 타겟 태스크에 있으나 그것이 스택의 최상위에 있지 않다면, 또는 그것이 스택의 최상위에 있으나 타겟 태스크에는 있지 않다면 – 신규 인스턴스는 생성될 것이고 스택에 푸쉬된다.

"singleTask" 와 "singleInstance" 모드들 또한 오직 한 측면에서만 서로 차이가 있다: "singleTask" 액티비티는 다른 액티비티가 그것의 태스크의 일부가 되는 것을 허용한다. 그것은 액티비티 스택의 루트에 있으나, 다른 액티비티들(반드시 "standard" 와 "singleTop" 액티비티)은 같은 태스크에서 런치될 수 있다. 반면에, "singleInstance" 액티비티는 어떤 액티비티도 그것의 태스크의 일부가 되는 것을 허용하지 않는다. 그것은 태스크 안에 있는 유일한 액티비티이

다. 만약 그것이 다른 액티비티를 시작한다면, 그 액티비티는 다른 태스크에 할당된다 – 마치 FLAG_ACTIVITY_NEW_TASK가 인텐트에 있었던 것처럼.

런치 모드와 인텐트 플래그를 사용한 그것들의 상호작용에 대한 더 많은 정보는, 2장. "애플리케이션 기초"의 "액티비티와 태스크"를 보라.

android:multiprocess

액티비티의 인스턴스가 그것을 시작했던 컴포넌트의 프로세스 안으로 런치될 수 있는지 여부 – 만약 그것이 될 수 있다면 "참true" 아니라면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

보통, 액티비티의 신규 인스턴스는 그것을 정의한 애플리케이션의 프로세스 안으로 런치된다. 그러므로 액티비티의 모든 인스턴스는 같은 프로세스에서 실행된다. 하지만 만약 이 플래그가 "참true"으로 설정된다면, 그 액티비티의 인스턴스는 다양한 프로세스에서 실행될 수 있다. 그것은 시스템이 그것이 사용되는 (퍼미션이 그것을 허용한다면) 곳이 어디든 간에 그것을 생성하는 것을 허용한다. 하지만 그것은 결코 거의 필요치 않으며 바람직하지 않다.

android:name

액티비티를 구현하는 클래스의 이름, 액티비티의 서브클래스. 그 애트리뷰트 값은 전체 클래스 이름을 가져야 한다("com.example.project.ExtracurricularActivity" 과 같이). 하지만 축약으로 만약 이름의 첫 번째 캐릭터가 마침표라면(예를 들어 ".ExtracurricularActivity), 그것은 〈manifest〉 엘리먼트에 지정된 패키지 이름 끝에 붙여진다. 이곳에선 어떤 디폴트도 없다. 이름은 명시되어야 한다.

android:noHistory API Level 3

사용자가 액티비티를 떠나서 더이상 그것이 스크린에 보이지 않게 되었을 때, 해당 액티비티가 액티비티의 스택에서 제거되고 종료(액티비티의 finish() 메쏘드가 호출됨)되어야 하는 지 여부 – 만약 그것이 종료되어야 한다면 "참true" 아니라면 "거짓false". 디폴트 값은 "거짓false"이다.

"참true" 값은 그 액티비티가 과거의 자취를 남기지 않을 것임을 의미한다. 그것은 태스크를 위한 액티비티의 스택에 존재하지 않을 것이다. 그러므로 사용자는 그것에 돌아갈 수 없게 될 것이다.

android:permission

클라이언트가 액티비티를 런치하기 위해 보유해야 하거나 또는 그렇지 않다면 인텐트에 응답하기 위해 얻어야 하는 퍼미션의 이름. 만약 startActivity() 또는 startActivityForResult()의 호출자가 명시된 퍼미션을 부여받지 않으면, 그것의 인텐트는 해당 액티비티에 전달되지 않는다.

만약 이 애트리뷰트가 설정되지 않으면, 퍼미션은 〈application〉 엘리먼트의 퍼미션 애트리뷰트에 의해 설정된 퍼미션이 액티비티에 적용된다. 만약 어느 것도 애트리뷰트를 설정하지 않는다면, 액티비티는 퍼미션에 의해 보호되지 않는다.

퍼미션에 관한 더 많은 정보에 대해서는, 앞에 나왔던 퍼미션 섹션과 8장. "보안과 퍼미션"을 보라.

android:process

액티비티가 실행되어야 하는 프로세스의 이름. 보통 애플리케이션의 모든 컴포넌트는 애플리케이션을 위해 생성되는 디폴트 프로세스 안에서 실행된다. 그 프로세스 이름은 애플리케이션 패키지와 같은 이름을 가진다. 〈application〉 엘리먼트의 process 애트리뷰트는 모든 컴포넌트를 위한 다른 디폴트를 설정할 수 있다. 그러나 각각의 컴포넌트는 그 디폴트를 오버라이드 할 수 있으며, 여러분의 애플리케이션을 여러 개의 프로세스에 걸쳐서 분포시키는 것을 가능하게 한다.

만약 이 애트리뷰트에 지정된 이름이 콜론(':')으로 시작된다면, 그 애플리케이션에게 프라이빗private한 신규 프로세스가 그것이 필요할 때 생성된다. 그리고 액티비티는 해당 프로세스에서 실행된다. 만약 그 프로세스 이름이 소문자로 시작된다면, 액티비티는 그 이름을 가진 글로벌 프로세스에서 실행될 것이며, 그 프로세스는 그렇게 하기 위한 퍼미션을 가진다. 이것은 다른 애플리케이션의 컴포넌트가 리소스 사용을 줄이기 위해 하나의 프로세스를 공유하는 것을 가능하게 한다.

android:screenOrientation

디바이스 상에서 액티비티가 보여지는 오리엔테이션. 그 값은 아래 나오는 문자열들 중에 하나가 될 수 있다.

"unspecified"	디폴트 값. 시스템이 오리엔테이션을 선택한다. 그것이 사용하는 정책과 그러므로 특정 컨텍스트에서 만들어진 선택은 디바이스 마다 다를 수 있다.

"landscape"	가로 방향 오리엔테이션 (화면은 높이보다 너비가 크다.)
"portrait"	세로 방향 오리엔테이션 (화면은 너비보다 높이가 크다.)
"behind"	액티비티 스택에서 그것 아래에 바로 있는 액티비티와 같은 오리엔테이션.
"user"	사용자가 현재 선호하는 오리엔테이션.
"sensor"	물리적 오리엔테이션 센서에 의해 결정된 오리엔테이션. 화면의 오리엔테이션은 사용자가 디바이스를 어떻게 쥐고 있는가에 의존한다; 그것은 사용자가 디바이스를 회전시킬 때 변한다.
"nosensor"	물리적 오리엔테이션 센서를 참조하는 것 없이 결정되는 오리엔테이션. 센서는 무시된다. 그러므로 그 화면은 사용자가 디바이스를 움직이는 방법에 기반해서 회전하지 않을 것이다. 이 차이를 제외하고는, 시스템은 "unspecified" 셋팅에 대한 것과 같은 정책을 사용하여 오리엔테이션을 선택한다.

android:stateNotNeeded

액티비티가 그것의 상태를 저장하지 않고 강제 종료되고 성공적으로 다시 시작될 수 있는 지 여부 – 만약 그것이 이전 상태를 참조하지 않고 재시작될 수 있다면 "참true" 그리고 만약 그것의 이전 상태가 요구되어진다면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

보통, 액티비티가 임시로 종료되기 전에 리소스를 저장하기 위해, 그것의 onSaveInstanceState() 메쏘드가 호출된다. 이 메쏘드는 액티비티가 재시작될 때, onCreate()에 전달되는 번들Bundle 오브젝트에 그 액티비티의 현재 상태를 저장한다. 만약 이 애트리뷰트가 "참true"으로 설정되면 onSaveInstanceState()는 호출되지 않을 것이며, onCreate()는 번들Bundle 대신에 널null을 전달받을 것이다 – 그것은 마치 액티비티가 처음 시작될 때와 같다.

"참true" 설정은 액티비티가 보유한 상태 없이 재시작 될 수 있는 것을 가능하게 한다. 예를 들어 홈 스크린을 보여주는 액티비티는 어떤 이유로 망가진다면 그것이 제거되지 않도록 보장하기 위해 이 설정을 사용한다.

android:taskAffinity

액티비티가 친화력affinity을 가지는 태스크. 동일 친화력을 가지는 액티비티는 개념적으로 동일한 태스크(사용자 관점에서는 같은 "application")에 속한다. 태스크의 친화력은 그것의 루트 액티비티의 친화력에 의해 결정된다.

친화력은 두 가지 것을 결정한다 – 액티비티가 다시 부모로 설정하게 되는 태스크(allowTaskReparenting 애트리뷰트를 보라)와 액티비티가 FLAG_ACTIVITY_NEW_TASK 플래그를 가지고 런치될 때 그것을 보유할 태스크.

디폴트로, 애플리케이션의 모든 액티비티는 동일한 친화력을 가진다. 여러분은 그것들을 다르게 그룹화하기 위해 이 애트리뷰트를 설정할 수 있고, 심지어 동일한 태스크 내에 다른 애플리케이션 안에서 정의된 액티비티를 배치할 수도 있다. 액티비티가 어떤 태스크에 대해 친화력을 가지지 않게 지정하기 위해서는, 그것에 빈 문자열을 설정하라.

만약 이 애트리뷰트가 설정되지 않으면, 액티비티는 애플리케이션에 대해 설정된 친화력을 상속한다(〈application〉 엘리먼트의 taskAffinity 애트리뷰트를 보라). 애플리케이션에 대한 디폴트 친화력의 이름은 〈manifest〉 엘리먼트에 의해 설정된 패키지 이름이다.

android:theme

액티비티에 대한 전체 테마를 정의하는 스타일 리소스에 대한 레퍼런스. 이것은 자동으로 액티비티의 컨텍스트에 이 테마(setTheme()를 보라)를 사용하도록 설정한다. 그리고 액티비티가 런치되기에 앞서서 "시작starting" 애니메이션을 또한 만들 지도 모른다(그 액티비티는 실제로 보이는 것과 더 잘 일치하도록 하기 위해).

만약 이 애트리뷰트가 설정되지 않으면, 액티비티는 애플리케이션 전체에 설정된 테마를 상속한다 – 〈application〉 엘리먼트의 theme 애트리뷰트를 보라. 만약 그 애트리뷰트 또한 설정되지 않으면, 디폴트 시스템 테마가 사용된다.

android:windowSoftInputMode API Level 3

액티비티의 메인 윈도우가 스크린상의 소프트 키보드를 포함한 윈도우와 어떻게 상호작용하는 지를 기술한다. 이 애트리뷰트에 대한 설정은 다음 두가지에 영향을 준다:

- 소프트 키보드 상태 – 액티비티가 사용자 관심의 포커스를 받을 때 소프트 키보드가 보여지는 지 또는 숨겨지는 지의 여부.
- 액티비티 메인 윈도우에 대한 조정 – 소프트 키보드를 위한 공간을 만들기 위해 액티비티 메인 윈도우를 작게 줄일 지의 여부 또는 메인 윈도우 일부가 소프트 키보드에 의해 가려질 때 현재 포커스를 보이도록 하기 위해 메인 윈도우의 컨텐트가 상하로 움직일 지의 여부.

셋팅 값은 다음 테이블에 나열된 값들의 하나 또는 하나의 "state..." 값과 하나의 "adjust..." 값의 조합이어야 한다. 다수의 "state..." 값들과 같이 다수의 값들을 설정하는 것은 정의되지 않은 결과를 가진다. 개개의 값들은 수직

바(|)에 의해 의해 구분된다. 예를 들어 아래와 같다.

```
<activity android:windowSoftInputMode="stateVisible|adjustResize" ...>
```

여기에 설정된 값들("stateUnspecified"와 "adjustUnspecified" 제외)은 테마에 설정된 값들을 오버라이드한다.

값	설 명
"stateUnspecified"	소프트 키보드의 상태(소프트 키보드가 숨겨질 지 또는 보여질 지 여부)는 지정되지 않는다. 시스템이 적절한 상태를 선택하거나 테마 설정 값에 의존할 것이다. 이 값은 소프트 키보드에 대한 디폴트 설정 값이다.
"stateUnchanged"	액티비티가 포어그라운드로 올라올 때 소프트 키보드는 보여지든 숨겨지든 마지막으로 있었던 상태로 유지된다.
"stateHidden"	사용자가 액티비티를 선택할 때 소프트 키보드는 숨겨진다. 즉, 이것은 사용자가 또 다른 액티비티를 떠나서 그 액티비티로 뒤돌아가는 것보다는 앞으로 그 액티비티로 갈 때 해당된다.
"stateAlwaysHidden"	액티비티의 메인 윈도우가 입력 포커스를 가질 때 소프트 키보드는 항상 숨겨진다.
"stateVisible"	소프트 키보드가 정상적으로 적합할 때 (사용자가 액티비티의 메인 윈도우로 앞으로 갈 때) 소프트 키보드는 보여진다.
"stateAlwaysVisible"	사용자가 액티비티를 선택할 때 소프트 키보드는 보여지도록 만들어진다. 즉, 이것은 사용자가 또 다른 액티비티를 떠나서 그 액티비티로 뒤돌아 가는 것보다는 앞으로 그 액티비티로 갈 때 해당된다.
"adjustUnspecified"	소프트 키보드를 위한 공간을 만들기 위해 액티비티의 메인 윈도우가 작게 줄어지는 지의 여부 또는 메인 윈도우의 현재 포커스를 스크린 상에 보이도록 하기 위해 메인 윈도우의 컨텐츠가 상하로 움질일 지의 여부를 지정하지 않는다. 윈도우의 컨텐츠가 스크롤할 수 있는 레이아웃 뷰들을 가지고 있는 지 여부에 따라 시스템은 이 모드들 중의 하나를 자동으로 선택할 것이다. 만약 그런 뷰가 있다면, 스크롤링이 더 작은 영역에서 모든 윈도우 컨텐츠를 보이게 할 수 있다는 가정하에 윈도우의 크기는 재조정될 것이다. 이 값이 메인 윈도우에 대한 디폴트 설정 값이다.
"adjustResize"	액티비티의 메인 윈도우는 스크린 상의 소프트 키보드를 위한 공간을 만들기 위해 항상 그것의 크기가 재조정될 것이다.
"adjustPan"	소프트 키보드를 위한 공간을 만들기 위해서 액티비티의 메인 윈도우 크기는 재조정되지 않는다. 오히려, 윈도우의 현재 포커스가 키보드에 의해 결코 가려지지 않고 사용자가 입력하고 있는 걸들을 항상 볼 수 있도록 윈도우의 컨텐츠가 자동으로 상하로 움직여진다. 이것은 일반적으로 크기 재조정보다는 덜 바람직하다. 왜냐하면, 사용자는 윈도우의 가려진 부분에 접근해서 상호작용하기 위해 소프트 키보드를 종료할 수도 있기 때문이다.

도입 :

API 레벨 3에서 추가되었던 noHisitory와 windowSoftInputMode를 제외한 모드 애트리뷰트는 API 레벨 1에서 도입되었다.

더 보기 :

```
<application>
<activity-alias>
```

〈activity-alias〉 API Level 1

문법 :

```
<activity-alias android:enabled=["true" | "false"]
                android:exported=["true" | "false"]
                android:icon="drawable resource"
                android:label="string resource"
                android:name="string"
                android:permission="string"
                android:targetActivity="string" >
    . . .
</activity-alias>
```

상위 엘리먼트 :

```
<application>
```

하위 엘리먼트 :

```
<intent-filter>
<meta-data>
```

설명 :

targetActivity 애트리뷰트에 의해 지칭되는 액티비티에 대한 앨리어스alias 타겟은 앨리어스와 같은 애플리케이션 안에 있어야 하고 매니페스트 안의 앨리어스 이전에 선언되어야 한다.

앨리어스는 독립적인 엔티티로써 타겟 액티비티를 표시한다. 그것은 자기 자신의 인텐트 필터 셋을 가질 수 있고, 그리고 그 인텐트 필터들은, 타겟 액티비티 자제에 있는 인텐트 필터 보다는, 어떤 인텐트가 앨리어스를 통해 타겟을 활성화할 수 있는가와 시스템이 앨리어스를 어떻게 취급해야 하는가를 결정한다. 예를 들어 앨리어스에 대한 인텐트 필터는 그 앨리어스를 애플리케이션 런처에 표시하게 하는 "android.intent.action.MAIN" 과 "android.intent.category.LAUNCHER"를 지정할 수도 있다. 비록 이러한 플래그들이 타겟 액티비티의 필터에 없을 지라도.

targetActivity를 제외하고, 〈activity-alias〉 애트리뷰트는 〈activity〉 애트리뷰트의 부분 집합이다. 부분 집합에 있는 애트리뷰트에 있어서는, 타겟

에 대한 설정 값 중 어떤 것도 앨리어스에 전달되지 않는다. 하지만 부분 집합에 없는 애트리뷰트에 있어서는, 타겟 액티비티에 대한 설정 값은 앨리어스에도 또한 적용된다.

애트리뷰트 :

android:enabled

타겟 액티비티가 이 앨리어스를 시스템에 의해 인스턴스화 될 수 있는 지 여부 – 만약 그것이 될 수 있다면 "참true" 그리고 아니라면 "거짓false" 이다. 디폴트 값은 "참true"이다.

〈application〉 엘리먼트는 그것 자신의 enabled 애트리뷰트를 가지며, 그것은 액티비티 앨리어스를 포함하는 모든 애플리케이션 컴포넌트에게 적용된다. 〈application〉과 〈activity-alias〉 애트리뷰트는 시스템이 앨리어스를 통해 타겟 액티비티를 인스턴스화 하는 것이 가능하도록 하기 위해 둘 다 "참true"이 되어야 한다. 만약 어느 쪽이든 "거짓false"이라면, 앨리어스는 동작하지 않는다.

android:exported

다른 애플리케이션 컴포넌트들이 이 앨리어스를 통해 타겟 액티비티를 런치할 수 있는지 여부 – 만약 그것들이 할 수 있으면 "참true"이고, 아니라면 "거짓false". 만약 "거짓false"이라면, 타겟 액티비티는 앨리어스와 같은 애플리케이션 또는 같은 유저user ID를 가진 애플리케이션의 컴포넌트에 의해서만 앨리어스를 통해 런치될 수 있다.

디폴트 값은 앨리어스가 인텐트 필터를 포함하는 지 여부에 달려있다. 어떤 필터도 없다는 것은 액티비티가 앨리어스의 정확한 이름을 명시함으로써만 그 앨리어스를 통해 그 액티비티가 호출될 수 있다는 것을 의미한다. 이것은 앨리어스가 애플리케이션 내부 용도(다른 것들 것 그것의 이름을 알 지 못할 것이다)로만 의도되었다는 것을 의미하며 – 그러므로 그 디폴트 값은 "거짓false"이다. 반면에 적어도 한 개의 필터가 존재한다는 것은 앨리어스가 외부 용도를 위해 의도되었다는 것을 의미한다 – 그러므로 그 디폴트 값은 "참true"이다.

android:icon

앨리어스를 통해 사용자에게 보여지는 타겟 액티비티에 대한 아이콘. 더 많은 정보를 위해서는 〈activity〉 엘리먼트의 icon 애트리뷰트를 보라.

android:label

앨리어스를 통해 사용자에게 보여지는 앨리어스에 대한 사용자가 읽을 수 있는 라벨. 더 많은 정보를 위해서는 〈activity〉 엘리먼트의 label 애트리뷰트를 보라.

android:name

앨리어스에 대한 고유한 이름. 이름은 전체 클래스 이름과 비슷해야 한다. 그러나 타겟 액티비티의 이름는 달리, 앨리어스 이름은 임의로 정할 수 있다; 그것은 실제 클래스 이름을 참조하지는 않는다.

android:permission

클라이언트가 타겟 액티비티를 런치하기 위해, 또는 앨리어스를 통해 어떤 것을 하기 위해 얻어야만 하는 퍼미션의 이름. 만약 startActivity() 또는 startActivityForResult()의 호출자가 명시된 퍼미션을 부여받지 않았다면, 타겟 액티비티는 활성화되지 않을 것이다.

이 애트리뷰트는 타겟 액티비티 자체에 대한 임의의 퍼미션 집합set을 대체한다. 만약 그것이 설정되지 않는다면, 앨리어스를 통해 타겟을 활성화하기 위해 퍼미션이 필요하지는 않다.

퍼미션에 대한 더 많은 정보를 위해, 앞에 나왔던 퍼미션 섹션을 보라.

android:targetActivity

앨리어스를 통해 활성화될 수 있는 액티비티의 이름. 이 이름은 매니페스트의 앨리어스 앞에 나오는 〈activity〉 엘리먼트의 name 애트리뷰트와 일치해야 한다.

도입 :

API 레벨 1

더 보기:

〈activity〉

〈application〉 API Level 1

문법 :

```
<application android:allowClearUserData=["true" | "false"]
            android:allowTaskReparenting=["true" | "false"]
```

```
                android:debuggable=["true" | "false"]
                android:description="string resource"
                android:enabled=["true" | "false"]
                android:hasCode=["true" | "false"]
                android:icon="drawable resource"
                android:label="string resource"
                android:manageSpaceActivity="string"
                android:name="string"
                android:permission="string"
                android:persistent=["true" | "false"]
                android:process="string"
                android:task="string"
                android:theme="resource or theme" >
        . . .
    </application>
```

상위 엘리먼트 :

```
<manifest>
```

하위 엘리먼트 :

```
<activity>
<activity-alias>
<service>
<receiver>
<provider>
<uses-library>
```

설명 :

애플리케이션에 대한 선언. 이 엘리먼트는 애플리케이션의 컴포넌트 각각을 선언하는 서브엘리먼트를 포함하며, 모든 해당 컴포넌트에 영향을 미치는 애트리뷰트를 가진다. (icon, label, permission, process, taskAffinity, 그리고 allowTaskReparenting 같은) 이들 애트리뷰트 중 많은 것들은 컴포넌트 엘리먼트의 해당 애트리뷰트에 대한 디폴트 값을 설정한다. (debuggable, enabled, description, 그리고 allowClearUserData 같은) 다른 것들은 전체 애플리케이션에 대한 값을 설정하며, 컴포넌트에 의해 오버라이드 될 수는 없다.

애트리뷰트

android:allowClearUserData

사용자들에게 사용자 데이터를 제거할 수 있는 옵션을 부여할 것인지 여부 –

그들이 그런 옵션을 부여받으면 "참true" 아니라면 "거짓false"이다. 만약 값이 "참true"이라면, 애플리케이션 관리자는 사용자들이 데이터를 제거할 수 있게 하는 옵션을 포함한다.

android:allowTaskReparenting

애플리케이션이 정의하는 액티비티들이 그것을 시작시킨 태스크에서 그들이 친화력을 가지는 태스크로 그 태스크가 다음에 프론트로 오게 될 때 이동할 수 있는지 여부 – 그것들이 이동할 수 있다면 "참true" 그리고 그것들이 시작된 태스크에 남아있어야 한다면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

〈activity〉 엘리먼트는, 여기에서 설정한 값을 오버라이드할 수 있는 그것들 자신의 allowTaskReparenting 애트리뷰트를 가질 수 있다. 더 많은 정보에 대해서는 그 애트리뷰트를 보라.

android:debuggable

사용자 모드로 디바이스 상에서 실행 중일 때 조차 애플리케이션이 디버그될 수 있는지 여부 – 그것이 될 수 있다면 "참true" 아니라면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

android:description

애플리케이션 라벨보다 더 길고 더 설명적인 애플리케이션에 대한 사용자가 읽을 수 있는 텍스트. 값은 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다. 라벨과는 달리, 그것은 원시 문자열이 될 수는 없다. 디폴트 값은 없다.

android:enabled

안드로이드 시스템이 애플리케이션의 컴포넌트를 인스턴스화 할 수 있는지 여부– 그것이 가능하다면 "참true" 아니라면 "거짓false"이다. 만약 그 값이 "참true"이라면, 각 컴포넌트의 enabled 애트리뷰트가 그 컴포넌트의 활성화 여부를 결정한다. 만약 그 값이 "거짓false"이라면, 그 값이 컴포넌트에서 지정한 값들을 오버라이드한다; 모든 컴포넌트들은 비활성화된다.

디폴트 값은 "참true"이다.

android:hasCode

애플리케이션이 임의의 코드를 포함하고 있는지 여부 – 만약 그것이 포함하고 있다면 "참true" 아니라면 "거짓false"이다. 그 값이 "거짓false" 일 때, 시스템은 컴포넌트를 런치할 때 어떤 애플리케이션 코드도 적재하려고 시도하지 않는

다. 디폴트 값은 "참true"이다.

어떤 애플리케이션이, 드문 경우지만 AliasActivity 클래스를 사용하는 액티비티와 같이, 내장된built-in 컴포넌트 클래스를 제외하고 아무 것도 사용하지 않는다면, 그것 소유의 어떤 코드도 가지지 않는다.

android:icon

전체 애플리케이션에 대한 아이콘, 그리고 애플리케이션의 각 컴포넌트에 대한 디폴트 아이콘, 〈activity〉, 〈activity-alias〉, 〈service〉, 〈receiver〉 그리고 〈provider〉 엘리먼트들에 대한 각각의 icon 애트리뷰트를 보라.

이 애트리뷰트는 이미지 정의를 포함하는 드로어블drawable 리소스에 대한 레퍼런스로써 설정되어야 한다. 디폴트 아이콘은 없다.

android:label

전체 애플리케이션에 대한 사용자가 읽을 수 있는 라벨, 그리고 애플리케이션의 각 컴포넌트에 대한 디폴트 라벨. 〈activity〉, 〈activity-alias〉, 〈service〉, 〈receiver〉, 그리고 〈provider〉 엘리먼트에 대한 각각의 label 애트리뷰트를 보라.

라벨은 사용자 인터페이스의 다른 문자열들과 같이 로컬라이즈localize될 수 있도록, 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다. 하지만 여러분이 애플리케이션을 개발하는 동안 편리하도록, 그것은 또한 원시 문자열로 설정될 수 있다.

android:manageSpaceActivity

디바이스 상에서 애플리케이션에 의해 점유되는 메모리를 사용자가 관리할 수 있도록 시스템이 런치할 수 있는 액티비티 서브클래스의 전체 이름. 그 액티비티는 또한 〈activity〉 엘리먼트를 가지고 선언되어야 한다.

android:name

애플리케이션을 위해 구현된 애플리케이션 서브클래스의 전체 이름. 애플리케이션 프로세스가 시작될 때, 이 클래스는 애플리케이션의 모든 컴포넌트 이전에 인스턴스화된다.

그 서브클래스는 선택 사항이다; 대부분의 애플리케이션은 이것을 필요로 하지 않을 것이다. 만약 서브클래스가 없으면, 안드로이드는 베이스 애플리케이

션 클래스 인스턴스를 사용한다.

android:permission

클라이언트가 애플리케이션과 상호작용하기 위해 가져야 하는 퍼미션의 이름. 이 애트리뷰트는 모든 애플리케이션 컴포넌트에 퍼미션을 적용하는 편리한 방법이다. 그것은 각 컴포넌트의 퍼미션 속성을 설정함으로써 오버라이드될 수 있다.

퍼미션에 관한 더 많은 정보는, 앞에 나왔던 퍼미션 섹션과 8장. 보안과 퍼미션을 보라.

android:persistent

애플리케이션이 항상 실행 상태로 남아있어야 하는지 여부 – 만약 그것이 그래야 한다면 "참true" 아니라면 "거짓false"이다. 디폴드 값은 "거짓false"이다. 애플리케이션은 일반적으로 이 플래스를 설정해서는 안된다; 퍼시스턴스persistence 모드는 특정 시스템 애플리케이션만을 위해 의도된 것이다.

android:process

애플리케이션의 모든 컴포넌트가 실행되어야 하는 프로세스의 이름. 각 컴포넌트는 그것 자신의 process 애트리뷰트를 설정함으로써, 이 디폴트를 오버라이드 할 수 있다.

디폴트로, 안드로이드는 애플리케이션의 첫 번째 컴포넌트가 실행될 필요가 있을 때, 그 애플리케이션을 위한 프로세스를 생성한다. 모든 컴포넌트는 그 프로세스에서 작동한다. 디폴트 프로세스 이름은 〈manifest〉 엘리먼트에 의해 설정된 package 이름과 일치한다.

이 애트리뷰트에 다른 애플리케이션과 공유되는 프로세스 이름을 설정함으로써, 여러분은 두 애플리케이션의 컴포넌트들을 동일한 프로세스에서 실행되도록 배치할 수 있다 – 그러나 이것은 두 개의 애플리케이션 역시 사용자 ID를 공유하고, 동일 인증서로 사인된 것에 한한다.

만약 이 애트리뷰트에 할당된 이름이 콜론(':')으로 시작되면, 새로운 프로세스는 애플리케이션에 프라이빗private하며, 그것이 필요할 때 생성된다. 만약 그 프로세스 이름이 소문자로 시작한다면, 그 이름을 가진 글로벌 프로세스가 만들어진다. 글로벌 프로세스는 다른 애플리케이션에 의해 공유될 수 있으며, 이것은 리소스 사용을 감소시킨다.

android:taskAffinity

애플리케이션 내의 모든 액티비티들에 적용되는 친화력affinity 이름이며, 액티비티가 자기 자신의 taskAffinity 애트리뷰트를 사용해서 다른 친화력affinity를 설정하고 있다면 그것은 제외된다. 더 많은 정보에 대해서는 액티비티 애트리뷰트를 보라.

디폴트로, 애플리케이션 내의 모든 액티비티는 동일 친화력affinity을 공유한다. 그 친화력affinity이름은 〈manifest〉 엘리먼트에 의해 설정된 package 이름과 같다.

android:theme

애플리케이션 안의 모든 액티비티에 대한 디폴트 테마를 정의하는 스타일 리소스에 대한 레퍼런스. 각각의 액티비티는 그들 자신의 theme 애트리뷰트를 설정함으로써 디폴트를 오버라이드 할 수 있다; 더 많은 정보에 대해서는 액티비티 애트리뷰트를 보라.

도입 :

API 레벨 1

더 보기 :

```
<activity> <service> <receiver> <provider>
```

〈category〉 API Level 1

문법 :

```
<category android:name="string" />
```

상위 엘리먼트 :

```
<intent-filter>
```

설명 :

인텐트 필터에 카테고리 이름을 추가한다. 인텐트 필터에 대한 상세내용을 위해서 5장. "인텐트와 인텐트 필터" 그리고 필터 내에서의 카테고리 명세의 역할을 보라.

애트리뷰트 :

android:name

카테고리의 이름. 표준 카테고리들은 CATEGORY_name 상수처럼 인텐트 클래스 안에서 정의된다. 여기서 할당된 이름은 CATEGORY_ 뒤에 나오는 이름에 "android.intent.category." 접두어를 붙임으로써 그것들의 상수로부터 파생될 수 있다. 예를 들어 CATEGORY_LAUNCHER에 대한 문자열 값은 "android.intent.category.LAUNCHER"이다.

커스텀 카테고리는 그것의 유일함을 보장하기 위해 접두어로 패키지 이름을 사용해야 한다.

도입 :

API 레벨 1

더 보기 :

```
<action> <data>
```

〈data〉 API Level 1

문법 :

```
<data android:host="string"
      android:mimeType="string"
      android:path="string"
      android:pathPattern="string"
      android:pathPrefix="string"
      android:port="string"
      android:scheme="string" />
```

상위 엘리먼트 :

```
<intent-filter>
```

설명 :

인텐트 필터에 데이터 명세를 추가한다. 그 명세는 데이터 타입(mimeType 애트리뷰트)만으로 될 수도 있고, URI만일 수도 있고, 또는 데이터 타입과 URI 둘 다일 수도 있다. URI는 그것의 각각의 부분들에 대한 별도의 애트리뷰트들에 의해 지정된다.

scheme://host:port/path or pathPrefix or pathPattern

이 애트리뷰트들은 선택 사항이지만, 또한 상호 간에 의존적이다: 만약 인텐트

필터에 대한 스키마scheme가 지정되지 않는다면, 다른 모든 URI 애트리뷰트들은 무시된다. 만약 인텐트 필터에 대한 호스트host가 지정되지 않는다면, 포트port 애트리뷰트와 모든 경로path 애트리뷰트는 무시된다.

동일 〈intent-filter〉 엘리먼트 내에 포함된 모든 〈data〉 엘리먼트는 동일 필터에 기여한다. 예를 들면 다음의 필터 명세는,

```
<intent-filter . . . >
    <data android:scheme="something"
            android:host="project.example.com" />
    . . .
</intent-filter>
```

아래의 것과 동일하다.

```
<intent-filter . . . >
    <data android:scheme="something" />
    <data android:host="project.example.com" />
    . . .
</intent-filter>
```

여러분은 인텐트 필터에 다양한 데이터 옵션을 주기 위하여 〈intent-filter〉 안에 임의의 갯수의 〈data〉 엘리먼트를 놓을 수 있다. 그것의 애트리뷰트들 중 어떤 것도 디폴트 값을 가지지 않는다.

인텐트 오브젝트가 필터와 일치되는 방법에 대한 규칙을 포함하는 인텐트 필터의 동작방법에 대한 정보는 다른 문서, 인텐트와 인텐트 필터에서 찾을 수 있다. 앞에 나왔던 인텐트 필터 섹션을 또한 보라.

애트리뷰트 :

android:host

URI 권위authority의 호스트host 영역. 필터에 대한 스키마scheme 애트리뷰트가 또한 지정되지 않는 한, 이 애트리뷰트는 의미가 없다.

주의 : 공식 RFC와는 달리, 안드로이드 프레임워크에서의 호스트 이름 매칭은 대소문자를 구분한다. 결과적으로 여러분은 항상 호스트 이름을 소문자로 지정해야 한다.

android:mimeType

image/jpeg 또는 audio/mpeg4-generic과 같은 MIME 미디어 타입. 서

브타입은 어떤 서브타입과도 일치된다는 것을 가리키기 위해 별표 와일드카드(*)가 될 수 있다.

주의 : 공식 RFC MIME 타입과는 달리, 안드로이드 프레임워크에서 MIME 타입 매칭은 대소문자를 구분한다. 결과적으로 여러분은 항상 MIME 타입을 소문자를 사용해서 지정해야 한다.

android:path
android:pathPrefix
android:pathPattern

URI의 path(경로) 영역. path 애트리뷰트는 인텐트 오브젝트 내의 전체 경로path와 일치하는 전체 경로path를 지정한다. pathPrefix 애트리뷰트는 인텐트 오브젝트 내 경로path의 첫 머리만 일치하는 부분적인 경로path를 지정한다. pathPattern 애트리뷰트는 인텐트 오브젝트 내의 전체 경로path에 일치하는 전체 경로path를 지정하지만, 그것은 다음과 같은 와일드카드를 포함할 수 있다.

- 별표('*')는 바로 앞에 나오는 문자가 0번 이상 발생하는 것을 나타낸다.
- 마침표 뒤에 별표가 나오는 것(".*")은 어떤 문자든 0번 이상 발생하는 것을 나타낸다.

문자열이 XML로부터 읽어질 때(그것이 패턴으로써 분석되기 전에) "\"는 이스케이프escape 문자로써 사용되기 때문에, 여러분은 두 개의 이스케이프escape가 필요할 것이다.: 예를 들어 글자 그대로인 '*'는 "*"로 작성되어질 것이고, 글자 그대로인 '\'는 "\\\\"로 작성되어질 것이다. 이것은 여러분이 만약 Java 코드 안에서 문자열을 만들 때 작성할 필요가 있는 것과 기본적으로 같다.

이러한 세 가지 타입의 패턴에 대한 더 많은 정보에 대해서는, PatternMatcher 클래스 안의 PATTERN_LITERAL, PATTERN_PREFIX, 그리고 PATTERN_SIMPLE_GLOB에 대한 설명을 보라.

스키마scheme와 호스트host 애트리뷰트가 필터에 대해서 지정되기만 하면 이 애트리뷰트들은 의미를 갖는다.

android:port

URI 권위authority의 port 영역. 이 애트리뷰트는 필터에 대해 스키마scheme와

호스트host 애트리뷰트가 또한 지정될 때에만 의미를 갖는다.

android:scheme

URI의 scheme영역. 이것은 URI를 지정하기 위한 최소한의 필수 애트리뷰트이다; 적어도 필터를 위해선 하나의 scheme 애트리뷰트는 설정되어야 하며, 그렇지 않다면 URI의 나머지 애트리뷰트들은 의미가 없다.

스키마scheme는 뒤에 콜론 없이 지정된다(예를 들어 http: 이라기보다는 http이다).

만약 필터가 어떤 스키마scheme도 없이 데이터 타입 설정(mimeType 애트리뷰트)을 가진다면, content: 과 file: 스키마scheme로 가정된다.

주의 : RFC와는 달리, 안드로이드 프레임워크에서 스키마 매칭은 대소문자를 구분한다. 결과적으로 여러분은 항상 스키마를 소문자를 사용해서 지정해야 한다.

도입 :

API 레벨 1

더 보기 :

```
<action> <category>
```

⟨grant-uri-permission⟩ API Level 1

문법 :

```
<grant-uri-permission android:path="string"
                      android:pathPattern="string"
                      android:pathPrefix="string" />
```

상위 엘리먼트 :

```
<provider>
```

설명 :

부모parent 컨텐트 프로바이더 퍼미션이 부여될 수 있는 데이터 서브셋subset을 지정한다. 데이터 서브셋subset은 content: URI(URI의 권위(authority) 영역은 컨텐트 프로바이더에 대한 식별자이다)의 경로path의 일부분을 가리킨다. 퍼미션을 부여하는 grant 것은 일반적으로 컨텐트 프로바이더의 데이터에 대한 접근 퍼미션을 가지지 않은 클라이언트에게 한 번은 그 제약을 극복하게 하는 것을 가능하게 하

는 방법이다.

만약 컨텐트 프로바이더의 grantUriPermissions 애트리뷰트가 "참true"이라면, 퍼미션은 프로바이더 영역 안의 어떤 데이터에 대해서도 부여될 수 있다. 하지만 만약 그 애트리뷰트가 "거짓false"이라면, 퍼미션은 이 엘리먼트에 의해 지정된 데이터 서브셋subset에 한해서만 부여될 수 있다. 프로바이더는 임의의 개수의 〈grant-uri-permission〉 엘리먼트를 포함할 수 있다. 각각은 오직 하나의 경로path (세 개의 가능한 애트리뷰트 중 오직 한 개)만 지정할 수 있다.

퍼미션을 부여하는 방법에 관한 정보에 대해서는, 〈intent-filter〉 엘리먼트의 grantUriPermissions 애트리뷰트를 보라.

애트리뷰트 :

android:path
android:pathPrefix
android:pathPattern

퍼미션이 부여될 수 있는 데이터 서브셋subset 또는 서브셋subset들을 구분하는 경로path. 경로path 애트리뷰트는 전체 경로path를 지정한다; 퍼미션은 그 경로path에 의해 구분되는 특정 데이터 서브셋subset에게만 부여될 수 있다. pathPrefix 애트리뷰트는 경로path의 첫 머리 부분을 지정한다; 퍼미션은 그 첫 머리 부분을 공유하는 경로들paths을 가지는 모든 데이터 서브셋subset들에 부여될 수 있다. pathPattern 애트리뷰트는 전체 경로path를 지정하지만, 그것은 다음과 같은 와일드카드를 포함할 수 있다.

- 별표('*')는 바로 앞에 나오는 문자가 0번 이상 발생하는 것을 나타낸다.
- 별표가 마침표 뒤에 나오는 것(".*")은 어떤 문자든 0번 이상 발생하는 것을 나타낸다.

문자열이 XML로부터 읽어질 때(그것이 패턴으로써 분석되기 전에) "\"는 이스케이프escape 문자로써 사용되기 때문에, 여러분은 두 개의 이스케이프escape가 필요할 것이다.: 예를 들어 글자 그대로인 '*'는 "*"로 작성되어질 것이고, 글자 그대로인 '\'는 "\\\\"로 작성되어질 것이다. 이것은 여러분이 만약 Java 코드 안에서 문자열을 만들 때 작성할 필요가 있는 것과 기본적으로 같다.

이러한 세 가지 타입의 패턴에 대한 더 많은 정보에 대해서는, PatternMatcher 클래스 안의 PATTERN_LITERAL, PATTERN_PREFIX, 그리고

PATTERN_SIMPLE_GLOB에 대한 설명을 보라.

도입 :

API 레벨 1

더 보기:

〈provider〉 엘리먼트의 grantUriPermissions 애트리뷰트

〈instrumentation〉 API Level 1

문법 :

```
<instrumentation android:functionalTest=["true" | "false"]
                 android:handleProfiling=["true" | "false"]
                 android:icon="drawable resource"
                 android:label="string resource"
                 android:name="string"
                 android:targetPackage="string" />
```

상위 엘리먼트 :

```
<manifest>
```

설명 :

여러분이 애플리케이션의 시스템과의 상호작용을 감시하는 것을 가능하게 하기 위해 instrumentation(도구) 클래스를 선언한다. Instrumentation 오브젝트는 모든 애플리케이션 컴포넌트 이전에 인스턴스화 된다.

애트리뷰트 :

android:functionalTest

Instrumentation 클래스가 기능적인 테스트를 위해 수행되어야 하는지 여부. 만약 그것을 해야 한다면 "참true" , 아니라면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

android:handleProfiling

Instrumentation 오브젝트가 프로파일링profiling을 온오프onoff할 것인지 여부 – 만약 그것이 프로파일링을 시작할 때와 멈출 때를 결정한다면 "참true" 그리고 프로파일링이 그것이 실행되는 모든 시간 동안 계속된다면 "거짓false"이다. "참true" 값은 해당 오브젝트가 특정 오퍼레이션 집합set에 대한 타겟 프로파일링을 가능하게 한다. 디폴트 값은 "거짓false"이다.

android:icon

Instrumentation 클래스를 표현하는 아이콘. 이 애트리뷰트는 드로어블 drawable 리소스에 대한 레퍼런스로써 설정되어야 한다.

android:label

Instrumentation 클래스를 대해 사용자가 읽은 수 있는 라벨. 라벨은 원시 raw 문자열 또는 문자열 리소스에 대한 레퍼런스로써 설정될 수 있다.

android:name

Instrumentation 서브클래스의 이름. 이것은 ("com.example.project.StringInstrumentation" 와 같이) 전체 클래스 이름이 되어야 한다. 하지만 단축형으로 그 이름의 첫 번째 문자가 마침표라면, 그것은 〈manifest〉 엘리먼트에서 지정된 package 이름 끝에 덧붙어진다.

디폴트는 없다. 이름은 지정되어야 한다.

android:targetPackage

Instrumentation 오브젝트를 실행할 대상 애플리케이션. 애플리케이션은 〈manifest〉 엘리먼트에 의해 그것의 매니페스트 파일 내에서 할당된 package 이름에 의해 구분된다.

도입 :

API 레벨 1

〈intent-filter〉 API Level 1

문법 :

```
<intent-filter android:icon="drawable resource"
               android:label="string resource"
               android:priority="integer" >
    . . .
</intent-filter>
```

상위 엘리먼트 :

```
<activity>
<activity-alias>
<service>
```

```
<receiver>
```

필수 하위 엘리먼트 :

```
<action>
```

하위 엘리먼트 :

```
<category>
<data>
```

설명 :

액티비티, 서비스, 또는 브로드캐스트 리시버가 응답할 수 있는 인텐트의 타입을 명시한다. 인텐트 필터는 그것의 부모parent 컴포넌트의 기능을 선언한다 – 액티비티 또는 서비스가 할 수 있는 것이 무엇인지, 그리고 브로드캐스트 리시버가 처리할 수 있는 타입이 무엇인지. 그것은 컴포넌트에 무의미한 인텐트들을 필터링함으로써, 공시된 타입의 인텐트를 수신하도록 컴포넌트를 개방한다.

필터의 대부분의 컨텐트는 그것의 〈action〉, 〈category〉, 그리고 〈data〉 서브엘리먼트에 의해 기술된다.

필터의 더 상세한 논의에 대해서는, 앞에 나왔던 인텐트 필터 섹션과 5장. "인텐트와 인텐트 필터"를 보라.

애트리뷰트 :

android:icon

그 컴포넌트가 필터에 의해 기술된 기능을 가지는 것과 같이 사용자에게 표시될 때, 그것의 부모 액티비티, 서비스, 또는 브로드캐스트 리시버를 표시하는 아이콘.

이 애트리뷰트는 이미지 정의를 포함하는 드로어블drawable 리소스에 대한 레퍼런스로써 설정되어야 한다. 디폴트 값은 부모 컴포넌트의 icon 애트리뷰트에 의해 설정된 아이콘이다. 만약 부모가 icon 설정하지 않는다면, 디폴트는 〈application〉 엘리먼트에 의해 설정된 아이콘이다.

인텐트 필터 아이콘에 대한 보다 많은 것에 대해서는, 앞에 나왔던 아이콘과 라벨을 보라.

android:label

부모 컴포넌트에 대한 사용자가 읽을 수 있는 라벨. 이 라벨은 부모 컴포넌트에 의해 설정된 것보다는 컴포넌트가 필터에 의해 기술된 기능을 가지는 것과 같이 사용자에게 보여질 때 사용되어진다.

라벨은 사용자 인터페이스의 다른 문자열과 같이 로컬라이즈 될 수 있도록, 문자열 리소스에 대한 참조로써 설정되어야만 한다. 하지만 여러분이 애플리케이션을 개발하는 동안 편리하도록, 그것은 또한 원시 문자열로써 설정될 수 있다.

디폴트 값은 부모 컴포넌트에 의해 설정된 라벨이다. 만약 부모가 라벨을 설정하지 않으면, 디폴트는 〈application〉 엘리먼트의 label 애트리뷰트에 의해 설정된 라벨이다.

인텐트 필터 라벨에 관한 더 많은 것에 대해서는, 앞에 나왔던 아이콘과 라벨을 보라.

android:priority

필터에 의해 기술된 타입의 인텐트를 처리하는 것에 대해 부모 컴포넌트에게 주어져야만 하는 우선순위priority. 이 애트리뷰트는 액티비티와 브로드캐스트 리시버에 의미가 있다.

- 그것은 필터와 일치하는 인텐트에 대해, 그 인텐트에 또한 응답할 수 있는 다른 액티비티들과 비교하여 임의의 액티비티가 응답할 수 있는 방법에 대한 정보를 제공한다. 인텐트가 서로 다른 우선순위를 가지는 여러 액티비티에 의해 처리될 수 있을 때, 안드로이드는 인텐트에 대한 잠재적 타겟으로써 더 높은 우선순위 값을 가지는 액티비티만을 고려할 것이다.
- 그것은 브로드캐스트 메시지를 수신하기 위해 실행되는 브로드캐스트 리시버의 순서를 제어한다. 더 높은 우선순위 값을 가지는 것이 더 낮은 값을 갖는 것보다 먼저 호출된다(순서는 동기적 메시지들에게만 적용된다; 그것은 비동기적 메시지에 대해서는 무시된다).

만약 여러분이 정말로 브로드캐스트를 특정 순서대로 수신하도록 하는 것이 필요하거나, 또는 안드로이드가 다른 것들 보다 특정 액티비티를 선호하도록 강제하고 싶으면 이 애트리뷰트를 사용하라.

값은 “100”와 같은 정수integer이어야 한다. 더 높은 숫자는 더 높은 우선순위를 가진다.

도입 :

API 레벨 1

더 보기 :

```
<action> <category> <data>
```

〈manifest〉 API Level 1

문법 :

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="string"
          android:sharedUserId="string"
          android:sharedUserLabel="string resource"
          android:versionCode="integer"
          android:versionName="string" >
    . . .
</manifest>
```

상위 엘리먼트 :

없음.

필수 하위 엘리먼트 :

```
<application>
```

하위 엘리먼트 :

```
<instrumentation>
<permission>
<permission-group>
<permission-tree>
<uses-configuration>
<uses-permission>
```

설명 :

AndroidManifest.xml 파일의 루트 엘리먼트. 그것은 〈application〉 엘리먼트를 포함해야 하며, 그리고 xlmns:android 와 package 애트리뷰트를 지정해야 한다.

애트리뷰트 :

xmlns:android

안드로이드 네임스페이스를 정의한다. 이 애트리뷰트는 항상 "http://schemas.android.com/apk/res/android"로 설정되어야 한다.

package

애플리케이션에 대한 전체 Java 패키지 이름. 그 이름은 고유해야 한다. 그 이름은 ('A'부터 'Z'까지의) 대문자 또는 소문자, 숫자, 그리고 언더스코어('_')를 포함할 수 있다. 하지만, 패키지 이름의 각 영역은 문자로만 시작한다. 예를 들어 구글에 의해 배포된 애플리케이션은 com.google.app.application_name 형식으로 된 이름을 가질 수 있다.

패키지 이름은 애플리케이션에 대한 고유한 식별자 역할을 한다. 그것은 또한 애플리케이션 프로세스에 대한 디폴트 이름이고 액티비티의 디폴트 친화력 affinity 이름이다(〈application〉 엘리먼트의 process 애트리뷰트를 보라)와 액티비티(〈activity〉 엘리먼트의 taskAffinity 애트리뷰트를 보라).

android:sharedUserId

다른 애플리케이션과 공유될 리눅스 유저user ID의 이름. 디폴트로, 안드로이드는 각각의 애플리케이션에게 그것 자신의 고유한 유저user ID를 할당한다. 하지만 만약 이 애트리뷰트가 두 개 또는 그 이상의 애플리케이션에 대해 동일한 값으로 설정된다면, 그들은 동일한 ID를 모두 공유할 것이다 – 그들이 또한 동일한 인증서에 의해 사인된다는 조건 하에서. 동일 유저user ID를 가진 애플리케이션은 각각 서로의 데이터에 접근할 수 있고, 원한다면 동일 프로세스에서 실행될 수 있다.

android:sharedUserLabel API Level 3

공유 유저user ID 에 대한 사용자가 읽을 수 있는 라벨. 그 라벨은 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다; 그것은 원시raw 문자열이 될 수는 없다.

이것은 API 레벨 3에서 도입되었다. 그것은 shardUserId 또한 설정되었을 때에만 유의미하다.

android:versionCode

내부적인 버전 숫자. 이 숫자는 더 최근 버전을 가리키는 더 높은 숫자를 가

지고, 하나의 버전이 다른 버전보다 더 최근 것인지 여부를 결정하기 위해서만 사용된다. 이것은 사용자에게 보여지는 버전 숫자는 아니다; 그 숫자는 versionName 애트리뷰트에 의해 설정된다.

그 값은 "100"와 같은 정수integer로 설정되어야 한다. 여러분은 각각의 다음 버전이 더 높은 숫자를 가지기만 한다면, 여러분이 원하는 어떤 것으로든 그것을 정의할 수 있다. 예를 들어 그것은 빌드 숫자가 될 수도 있으며, 또는 여러분은 "x.y" 포맷의 버전 숫자를 "x"와 "y"를 각각 상위, 그리고 하위 16비트로 인코딩한 정수integer로 변환할 수도 있고, 또는 여러분은 새로운 버전이 릴리즈될 때마다 간단하게 숫자를 하나씩 증가할 수도 있다.

android:versionName

사용자에게 보여지게 되는 버전 숫자. 이 애트리뷰트는 원시 문자열 또는 문자열 리소스에 대한 레퍼런스로써 설정될 수 있다. 문자열은 사용자에게 표시되는 것 말고 다른 목적이 없다. 그 versionCode 애트리뷰트가 내부적으로 사용되는 의미있는 버전 숫자를 보유한다.

도입 :

API 레벨 3에서 추가되었던 sharedUserLabel을 제외한 모드 애트리뷰트는 API 레벨 1에서 도입되었다.

더 보기 :

```
<application>
```

〈meta-data〉 API Level 1

문법 :

```
<meta-data android:name="string"
           android:resource="resource specification"
           android:value="string" />
```

상위 엘리먼트 :

```
<activity>
<activity-alias>
<service>
<receiver>
```

설명 :

부모 컴포넌트에 제공될 수 있고 추가적이고 임의의 데이터 아이템에 대한 이름-값 한 쌍name-value pair. 컴포넌트 엘리먼트는 임의의 개수의 〈meta-data〉 서브엘리먼트를 포함할 수 있다. 그것들 모두의 값들은 하나의 번들Bundle 오브젝트로 모아지고, PackageItemInfo.metaData 필드로써 컴포넌트에서 사용가능해진다.

보통 값들은 value 애트리뷰트를 통해 지정되어진다. 하지만 리소스 ID에 값을 할당하기 위해 리소스 애트리뷰트를 대신 사용하라. 예를 들어 다음의 코드는 "zoo" 이름에 @string/kangaroo 리소스 안에서 저장된 임의의 값을 지정한다.

```
<meta-data android:name="zoo"
    android:value="@string/kangaroo" />
```

반면에, 리소스 애트리뷰트를 사용하는 것은 "zoo" 에 리소스 안에 저장된 값이 아니라 리소스의 숫자 ID를 할당할 것이다.

```
<meta-data android:name="zoo"
    android:resource="@string/kangaroo" />
```

여러분이 여러 개의 별도의 〈meta-data〉 엔트리로써 관련 데이터를 제공하지 않는 것을 적극 권장한다. 대신에, 만약 여러분이 컴포넌트와 관련된 복잡한 데이터를 가지고 있다면, 리소스로 그것을 저장하고 그 리소스 ID의 컴포넌트를 알려주기 위해 resource 애트리뷰트를 사용하라.

애트리뷰트 :

android:name

아이템에 대한 고유한 이름. 이름을 유일하게 하기 위해, 예를 들어 "com.example.project.activity.fred" Java 스타일 네이밍naming 규약을 사용하라.

android:resource

리소스에 대한 레퍼런스. 리소스의 ID는 아이템에 할당된 값이다. ID는 Bundle.getInt() 메쏘드에 의해 meta-data Bundle로부터 얻어질 수 있다.

android:value

아이템에 할당된 값. 값으로 할당될 수 있는 데이터 타입과 그것들의 값을 얻기 위해 사용하는 컴포넌트의 번들Bundle 메쏘드들이 아래의 테이블에 나열되

어 있다.

타입	Bundle 메쏘드
유니코드 문자를 위한 "\\n" 와 "\\uxxxxx" 같은 이스케이프(escape) 캐릭터에 더블 백슬래시(\\)를 사용하는 문자열 값.	getString()
"100"같은 정수(integer) 값.	getInt()
"참(true)" 또는 "거짓(false)" 중 하나인 불리언 값.	getBoolean()
"#rgb"/"#argb"/"#rrggbb" 또는 "#aarrggbb"형식의 컬러 값.	getString()
"1.23" 같은 부동소수점 값,	getFloat()

도입 :

API 레벨 1

〈path-permission〉 API Level 4

문법 :

```
<path-permission android:path="string"
                 android:pathPrefix="string"
                 android:pathPattern="string"
                 android:permission="string"
                 android:readPermission="string"
                 android:writePermission="string" />
```

상위 엘리먼트 :

```
<provider>
```

설명 :

컨텐트 프로바이더내에 있는 데이터의 특정 부분 집합에 대한 경로path와 요구되는 퍼미션을 정의한다. 이 엘리먼트는 여러 경로를 지원하기 위해서 여러번 지정될 수 있다.

애트리뷰트 :

android:path

컨텐트 프로바이더 데이터의 부분 집합에 대한 완전한 URI 경로. 퍼미션은 이 경로에 의해서 식별된 특정 데이터에만 허용될 수 있다. 검색 제안 컨텐트 search suggestion content을 제공하기 위해서 이 경로가 사용될 때 이 경로의 끝에 "/search_suggest_query" 가 붙여져야 한다.

android:pathPrefix

컨텐트 프로바이더 데이터의 부분 집합에 대한 완전한 URI 경로의 시작 부분. 퍼미션은 이 시작 부분을 공유하는 경로를 가지는 모든 데이터 부분 집합에 허용될 수 있다.

android:pathPattern

컨텐트 프로바이더 데이터의 부분 집합에 대한 완전한 URI 경로인데 다음과 같은 와일드 카드를 사용할 수 있는 경로.

- 별표('*')는 바로 앞에 나오는 문자가 0번 이상 발생하는 것을 나타낸다.
- 마침표 뒤에 별표가 나오는 것(".*")은 어떤 문자든 0번 이상 발생하는 것을 나타낸다.

문자열이 XML로부터 읽어질 때(그것이 패턴으로써 분석되기 전에) "\"는 이스케이프escape 문자로써 사용되기 때문에, 여러분은 두 개의 이스케이프escape가 필요할 것이다.: 예를 들어 글자 그대로인 '*'는 "*"로 작성되어질 것이고, 글자 그대로인 '\'는 "\\\\"로 작성되어질 것이다. 이것은 여러분이 만약 Java 코드 안에서 문자열을 만들 때 작성할 필요가 있는 것과 기본적으로 같다.

이러한 세 가지 타입의 패턴에 대한 더 많은 정보에 대해서는, PatternMatcher 클래스 안의 PATTERN_LITERAL, PATTERN_PREFIX, 그리고 PATTERN_SIMPLE_GLOB에 대한 설명을 보라.

android:permission

클라이언트가 컨텐트 프로바이더 데이터를 읽거나 쓰기 위해서 가져야 하는 퍼미션 이름. 이 애트리뷰트는 읽고 쓰기 모두에 대한 하나의 퍼미션을 설정하는 편리한 방법이다. 하지만 readPermission과 writePermission 애트리뷰트가 이 애트리뷰트보다 우선 순위를 가진다.

android:readPermission

클라이언트가 컨텐트 프로바이더를 쿼리query하기 위해서 가져야 하는 퍼미션.

android:writePermission

클라이언트가 컨텐트 프로바이더를에 의해 제어되는 데이터를 수정하기 위해 가져야 하는 퍼미션.

도입 :

API 레벨 4

더 보기 :

SearchManager
Manifest.permission
Security and Permissions

⟨permission⟩ API Level 1

문법 :

```
<permission android:description="string resource"
            android:icon="drawable resource"
            android:label="string resource"
            android:name="string"
            android:permissionGroup="string"
            android:protectionLevel=["normal" | "dangerous" |
                                     "signature" | "signatureOrSystem"]
/>
```

상위 엘리먼트 :

```
<manifest>
```

설명 :

이것 또는 다른 애플리케이션의 특정 컴포넌트 또는 기능에 대한 접근을 제한하기 위해 사용될 수 있는 보안 퍼미션을 선언한다. 퍼미션의 동작 방식에 대한 더 많은 정보에 대해서는 앞에 나왔던 퍼미션 섹션과 8장. "보안과 퍼미션"을 보라.

애트리뷰트 :

android:description

라벨보다 더 길고 더 많은 정보인 퍼미션에 대해 사용자가 읽을 수 있는 설명. 그것은 사용자에게 퍼미션을 설명하기 위해 표시될 수도 있다 – 예를 들어 다른 애플리케이션에게 퍼미션을 부여할 지 여부를 사용자에게 요청할 때.

이 애트리뷰트는 문자열 리소스에 대한 레퍼런스로 설정되어야 한다; label 애트리뷰트와는 달리 그것은 원시 문자열이 될 수 없다.

android:icon

퍼미션을 의미하는 아이콘에 대한 드로어블(drawable) 리소스에 대한 레퍼

런스.

android:label

사용자에게 보여질 수 있는 것으로 퍼미션에 대한 이름.

편의상, 여러분이 애플리케이션을 개발하는 동안은 라벨이 원시 문자열로 직접 설정될 수 있다. 하지만 애플리케이션이 배포될 준비가 되어 있을 때, 그것은 사용자 인터페이스의 다른 문자열과 같이 로컬라이즈될 수 있도록 문자열 리소스에 대한 레퍼런스로 설정되어야 한다.

android:name

퍼미션의 이름. 이것은 퍼미션에 대한 참조를 위해 코드에서 사용되는 이름이다 – 예를 들어 〈uses-permission〉 엘리먼트와 애플리케이션 컴포넌트의 permission 애트리뷰트에서 사용된다.

이름은 고유해야 한다. 그래서 그것은 Java 스타일 범위 지정을 사용해야 한다 – 예를 들어 "com.example.project.PERMITTED_ACTION"이다.

android:permissionGroup

하나의 그룹에 퍼미션을 할당한다. 이 애트리뷰트의 값은, 이것 또는 다른 애플리케이션의 〈permission-group〉 엘리먼트를 사용해서 선언되어야 하는 그룹의 이름이다. 만약 이 애트리뷰트가 설정되지 않으면, 퍼미션은 그룹에 속하지 않는다.

android:protectionLevel

퍼미션에 내재된 잠재적 위험을 나타내며, 퍼미션을 요청하는 애플리케이션에게 퍼미션 부여 여부를 결정할 때 시스템이 따라야 할 절차를 가리킨다. 이 값은 다음 문자열 중의 하나로 설정되어야 한다.

값	의미하는 바
"normal"	디폴트 값임. 낮은 수위의 위험을 가지는 퍼미션으로, 이것은 다른 애플리케이션, 시스템, 또는 사용자에게 최소한의 위험 요소를 가진 상태에서 퍼미션을 요청하는 애플리케이션이 그것과 분리된 애플리케이션 레벨(application-level)의 기능들에 대해 접근하는 것을 부여한다. 시스템은 설치시점에서 사용자에게 명시적인 승인을 요청하지 않고 그 퍼미션을 요청하는 애플리케이션에게 해당 유형의 퍼미션을 자동으로 부여한다(비록 사용자는 항상 설치하기 전에 이런 퍼미션들을 검토하는 옵션을 가질 수 있다 할지라도).

“dangerous”	높은 수위의 위험을 가지는 퍼미션으로, 이것은 퍼미션을 요청하는 애플리케이션이 사용자의 사적 데이터에 접근하게 하거나 또는 사용자에게 부정적 영향을 미칠 수 있는 디바이스에 대한 제어를 허용할 수 있다. 해당 유형의 퍼미션은 잠재적인 위험을 초래하기 때문에, 시스템은 해당 퍼미션을 요청하는 애플리케이션에게 자동으로 퍼미션을 부여하지 않는다. 예를 들어 애플리케이션에 의해 요청된 임의의 “dangerous” 퍼미션들은 사용자에게 보여질 것이며, 처리에 앞서 사용자의 확인을 요구하거나 또는 해당 기능에 대한 사용을 사용자에게 자동으로 허용하는 것을 피하기 위한 약간의 다른 접근법들이 부여될 수도 있다.
“signature”	이 퍼미션은 퍼미션을 요청하는 애플리케이션이 해당 퍼미션을 정의한 애플리케이션과 같은 인증서를 가지고 사인되었을 때에 한하여 시스템이 퍼미션을 부여한다. 만약 인증서가 일치한다면, 시스템은 사용자에게 알리는 과정없이 또는 사용자에게 명시적인 승인을 요청하지 않고 해당 퍼미션을 자동으로 부여한다.
“signatureOrSystem”	이 퍼미션은, 시스템이 안드로이드 시스템 이미지 안에 있거나, 또는 시스템 이미지 안에 있는 것들과 같은 인증서로 사인된 애플리케이션에 한하여 부여하는 것이다. 정확하게 애플리케이션이 어디에 설치되는 지 관계없이 대부분의 필요와 작업을 위해 “signature” 보호(protection) 레벨로 충분하기 때문에, 이 옵션을 사용하는 것은 피하기 바란다. "signatureOrSystem" 퍼미션은, 여러 업체들이 하나의 시스템 이미지내에 애플리케이션들을 내장시키고, 그 애플리케이션들이 함께 만들어졌기 때문에 명시적으로 특정 기능들을 공유해야 할 필요가 있는 어떤 특별 상황을 위해 사용된다.

도입 :

API 레벨 1

더 보기 :

```
<uses-permission>
<permission-tree>
<permission-group>
```

〈permission-group〉 API Level 1

문법 :

```
<permission-group android:description="string resource"
                  android:icon="drawable resource"
                  android:label="string resource"
                  android:name="string" />
```

상위 엘리먼트 :

```
<manifest>
```

설명 :

관련된 퍼미션에 대한 논리적 그룹화를 위한 이름을 선언한다. 각각의 퍼미션

은 〈permission〉 엘리먼트의 permissionGroup 애트리뷰트를 통해 그룹과 결합된다. 그룹의 멤버들은 사용자 인터페이스에서 함께 표시된다.

이 엘리먼트는 퍼미션 그 자체를 선언하지 않으며, 퍼미션이 놓일 수 있는 하나의 카테고리만을 선언한다는 것에 주의하라. 퍼미션을 선언하고, 그룹에 그것을 할당하는 것에 것에 대한 더 많은 정보에 대해서는 〈permission〉 엘리먼트를 보라.

애트리뷰트 :

android:description

그룹을 설명하는 사용자가 읽을 수 있는 텍스트. 그 텍스트는 라벨보다 더 길고 더 설명적이어야 한다. 이 애트리뷰트는 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다. label 애트리뷰트와 달리, 그것은 원시 문자열이 될 수 없다.

android:icon

퍼미션을 나타내는 아이콘. 이 애트리뷰트는 이미지 정의를 포함하고 있는 드로어블drawable 리소스에 대한 레퍼런스로 설정되어야 한다.

android:label

그룹에 대한 사용자가 읽을 수 있는 이름. 편의상, 여러분이 애플리케이션을 개발하는 동안 라벨은 원시 문자열로 직접 설정될 수 있다. 하지만 애플리케이션이 배포될 준비가 되어 있을 때, 그것은 사용자 인터페이스의 다른 문자열처럼 로컬라이즈될 수 있도록 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다.

android:name

그룹의 이름. 이것은 〈permission〉 엘리먼트의 permissionGroup 애트리뷰트에 할당될 수 있는 이름이다.

도입 :

API 레벨 1

더 보기 :

```
<permission>
<permission-tree>
```

```
<uses-permission>
```

〈permission-tree〉 API Level 1

문법 :

```
<permission-tree android:icon="drawable resource"
                 android:label="string resource"
                 android:name="string" />
```

상위 엘리먼트 :

```
<manifest>
```

설명 :

퍼미션 트리에 대한 기본적인 이름을 선언한다. 그 애플리케이션은 트리 내의 모든 이름들에 대한 소유권을 갖는다. 그것은 PackageManager.addPermission()을 호출해서 트리에 동적으로 신규 퍼미션을 추가할 수 있다. 트리 내의 이름들은 마침표(.)에 의해 분리된다. 예를 들어 만약 기본적인 이름이 com.example.project.taxes라면, 다음과 같은 퍼미션들이 추가될 수도 있다.

```
com.example.project.taxes.CALCULATE
com.example.project.taxes.deductions.MAKE_SOME_UP
com.example.project.taxes.deductions.EXAGGERATE
```

이 엘리먼트는 퍼미션 자체를 선언하지는 않으며, 오직 그것에 놓일 수 있는 퍼미션들의 네임스페이스만을 정의한다는 것에 주의하라. 퍼미션을 선언하는 정보에 대해서는 〈permission〉 엘리먼트를 보라.

애트리뷰트 :

android:icon

트리에 있는 모든 퍼미션들을 나타내는 아이콘. 이 애트리뷰트는 이미지 정의를 포함하는 드로어블drawable 리소스에 대한 레퍼런스로써 설정되어야 한다.

android:label

그룹에 대한 사용자가 읽을 수 있는 이름. 편의상, 라벨은 빠르고 지저분한 프로그래밍을 위해 원시 문자열로 직접적으로 설정될 수 있다. 하지만 애플리케이션이 배포될 준비가 되었을 때, 그것은 사용자 인터페이스의 다른 문자열처럼 로컬라이즈localize 될 수 있도록 문자열 리소스에 대한 레퍼런스로써 설정

되어야 한다.

android:name

퍼미션 트리의 기본에 있는 이름. 그것은 트리에 있는 모든 퍼미션 이름들에 대한 접두어로써 사용된다. 그 이름이 고유할 수 있도록 Java 스타일 범위 지정이 사용된다. 그 이름은 그것의 경로path에 있는 두 개의 마침표로 분리된 세그먼트 이상을 가져야 한다 – 예를 들어 com.example.base는 되지만, com.example는 안된다.

도입 :

API 레벨 1

더 보기 :

```
<permission>
<permission-group>
<uses-permission>
```

〈provider〉 API Level 1

문법 :

```
<provider android:authorities="list"
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:grantUriPermissions=["true" | "false"]
          android:icon="drawable resource"
          android:initOrder="integer"
          android:label="string resource"
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:permission="string"
          android:process="string"
          android:readPermission="string"
          android:syncable=["true" | "false"]
          android:writePermission="string" >
    . . .
</provider>
```

상위 엘리먼트 :

```
<application>
```

하위 엘리먼트 :

```
<meta-data>
<grant-uri-permission>
```

설명 :

컨텐트 프로바이더 – ContentProvider의 서브클래스 – 를 선언한다. 그것은 애플리케이션에 의해 관리되는 데이터에 대한 구조적인 접근을 제공한다. 애플리케이션의 일부가 되는 모든 컨텐트 프로바이더는 매니페스트 파일에 있는 〈provider〉 엘리먼트에 의해 표시되어야 한다. 선언되지 않는 어떤 것도 시스템에서 볼 수 없으며, 그러므로 실행하지 않을 것이다(여러분 애플리케이션의 일부로써 개발하는 컨텐트 프로바이더만을 선언할 필요가 있으며, 여러분의 애플리케이션이 사용하는 다른 애플리케이션에 의해 개발되는 것들에 대해서는 선언할 필요가 없다)

안드로이드 시스템은 content: URI의 권위authority 영역으로 컨텐트 프로바이더를 구분한다. 예를 들어 다음에 나오는 URI가 ContentResolver.query()에 전달된다고 가정하자.

```
content://com.example.project.healthcareprovider/nurses/rn
```

content: 스키마scheme는 그 데이터가 컨텐트 프로바이더에 속하는 것으로 간주하며, 권위authority(com.example.project.healthcareprovider)는 특정 프로바이더를 구분한다. 그러므로 권위authority는 고유해야 한다. 일반적으로 이 예제에서처럼, 권위authority는 ContentProvider 서브클래스에 대한 전체 이름이다. URI의 경로path 영역은 특정 데이터 서브셋subset을 구분하기 위해 컨텐트 프로바이더에 의해 사용될 수도 있다. 하지만 그 경로path는 매니페스트 안에 선언되지는 않는다.

컨텐트 프로바이더 사용과 개발에 관한 정보에 대해서는 7장. "컨텐트 프로바이더"를 보라.

애트리뷰트 :

android:authorities

컨텐트 프로바이더의 영역 아래에 있는 데이터를 구분하는 하나 또는 그 이상의 URI 권위authority 목록. 복수의 권위authority는 세미콜론으로 그 이름들을 구분함으로써 나열된다. 충돌을 회피하기 위해, 권위authority 이름은 Java 스타일 네이밍naming 규약을 사용해야 한다(com.example.provider.

cartoonprovider와 같이). 일반적으로 그것은 ContentProvider 서브클래스의 이름이다.

디폴트는 없다. 적어도 하나의 권위authority가 지정되어야 한다.

android:enabled

컨텐트 프로바이더가 시스템에 의해 인스턴스화될 수 있는 지 여부 – 만약 될 수 있다면 "참true" 아니라면 "거짓false" 이다. 디폴트 값은 "참true"이다.

〈application〉 엘리먼트는 컨텐트 프로바이더를 포함하여, 모든 애플리케이션 컴포넌트에 적용되는 그것 자신의 enabled 애트리뷰트를 가진다. 컨텐트 프로바이더가 활성화될 수 있게 하기 위해서는 〈application〉과 〈provider〉 애트리뷰트 둘 다 "참true"(디폴트로 그것들 모두가 그렇듯이)이 되어야 한다. 만약 둘 중 어느 하나가 "거짓false"이라면, 프로바이더는 비활성화된다; 그것은 인스턴스화 될 수 없다.

android:exported

컨텐트 프로바이더가 다른 애플리케이션의 컴포넌트에 의해 사용될 수 있는 지 여부 – 만약 사용될 수 있다면 "참true" 아니라면 "거짓false"이다. 만약 "거짓false"이라면, 프로바이더는 동일 애플리케이션의 컴포넌트 또는 동일 유저user ID를 가진 애플리케이션에게만 유용하다. 디폴트 값은 "참true"이다.

여러분은 컨텐트 프로바이더를 제공할 수 있지만 여전히 퍼미션 애트리뷰트를 가지고 그것에 대한 접근을 제한할 수 있다.

android:grantUriPermissions

보통 컨텐트 프로바이더의 데이터에 접근할 수 있는 퍼미션을 가지지 않은 것이 일시적으로 readPermission, writePermission, 그리고, permission 애트리뷰트에 의해 강제된 제약을 극복하고 그 데이터 접근을 할 수 있는 퍼미션을 부여받을 수 있는 지 여부 – 만약 퍼미션이 부여될 수 있다면 "참true" 아니라면 "거짓false"이다. "참true"이라면 컨텐트 프로바이더의 모든 데이터에 대해 퍼미션이 부여될 수 있다. "거짓false"이라면 퍼미션은 〈grant-uri-permission〉 서브엘리먼트 안에서 나열된 데이터 서브셋subset에 한해서만 부여될 수 있다.

퍼미션을 부여하는 것은 퍼미션에 의해 보호된 데이터에 대해 애플리케이션 컴

포넌트에게 한 번의 접근을 제공하는 방법이다. 예를 들어 이메일 메시지가 첨부파일을 포함할 때, 메일 애플리케이션은 그것을 열기 위한 적절한 뷰어가 해당 컨텐트 프로바이더의 모든 데이터를 볼 수 있는 일반적인 퍼미션을 가지고 있지 않을 지라도 그 뷰어를 호출할 지 모른다.

그런 경우에, 그 컴포넌트를 활성화하는 인텐트 오브젝트 내의 FLAG_GRANT_READ_URI_PERMISSION과 FLAG_GRANT_WRITE_URI_PERMISSION 플래그에 의해 퍼미션이 부여된다. 예를 들어 메일 애플리케이션은 Context.startActivity에 전달되는 인텐트 내에 FLAG_GRANT_READ_URI_PERMISSION를 넣을 수 있다. 그 퍼미션은 인텐트에 있는 URI에 국한된다.

만약 여러분이 이 애트리뷰트를 "참true"로 설정하거나 또는 〈grant-uri-permission〉 서브엘리먼트를 정의함으로써 이 기능을 활성화한다면, 여러분은 부여한 URI가 프로바이더로부터 지워지게 될 때, Context.revokeUriPermission()을 호출해야 한다.

또한 〈grant-uri-permission〉 엘리먼트를 보라.

android:icon

컨텐트 프로바이더를 나타내는 아이콘. 이 애트리뷰트는 이미지 정의를 포함하는 드로어블drawable 리소스에 대한 레퍼런스로써 설정되어야 한다. 만약 그것이 설정되지 않으면, 전체 애플리케이션을 위해 지정된 아이콘이 대신 사용된다(〈application〉 엘리먼트의 icon 애트리뷰트를 보라).

android:initOrder

동일 프로세스가 보유하고 있는 다른 컨텐트 프로바이더에 상대적으로, 컨텐트 프로바이더가 인스턴스화되어야 하는 순서. 컨텐트 프로바이더들 사이에 의존성이 있을 때, 그것들 각각에 대해 이 애트리뷰트를 설정하는 것은 그 의존성이 요구되는 순서대로 그것들을 생성하는 것을 보장한다. 그 값은 간단한 정수integer 값이며, 더 높은 숫자들이 먼저 초기화된다.

android:label

컨텐트 프로바이더에 대한 사용자가 읽을 수 있는 라벨. 만약 이 애트리뷰트가 설정되지 않으면, 전체 애플리케이션에 대한 라벨 설정이 대신 사용된다(〈application〉 엘리먼트의 label 애트리뷰트를 보라).

라벨은 사용자 인터페이스의 다른 문자열처럼 로컬라이즈 될 수 있도록 문자열 리소스에 대한 레퍼런스로 설정되어야 한다. 하지만 편의상 여러분이 애플리케이션을 개발하는 동안, 그것은 또한 원시 문자열로써 설정될 수 있다.

android:multiprocess

컨텐트 프로바이더의 인스턴스가 모든 클라이언트 프로세스에서 생성될 수 있는 지 여부 – 만약 인스턴스가 여러 프로세스에서 실행될 수 있다면 "참true" 아니라면 "거짓false"이다. 디폴트 값은 "거짓false"이다.

보통 컨텐트 프로바이더는 그것을 정의했던 애플리케이션 프로세스 안에서 인스턴스화 된다. 하지만 이 플래그가 "참true"으로 설정되면, 시스템은 그것과 상호작용하길 원하는 클라이언트가 있는 모든 프로세스 내에서 인스턴스를 생성할 수 있다. 따라서 프로세스 간 통신 오버헤드를 방지한다.

android:name

컨텐트 프로바이더, 즉 ContentProvider의 서브클래스를 구현하는 클래스의 이름. 이것은 전체 클래스 이름이어야 한다("com.example.project.TransportationProvider" 같이). 하지만 단축형으로 이름의 첫 번째 문자가 마침표라면 그것은 〈manifest〉 엘리먼트에 지정된 패키지 이름뒤에 덧붙여진다.

디폴트는 없다. 이름은 반드시 지정되어야 한다.

android:permission

클라이언트가 컨텐트 프로바이더의 데이터를 읽거나 쓰기 위해 가져야 하는 퍼미션의 이름. 이 애트리뷰트는 읽기와 쓰기 모두에 대해, 하나의 퍼미션을 설정하는 편리한 방법이다. 하지만 readPermission과 writePermission 애트리뷰트가 이것보다 우선권을 갖는다. 만약 readPermission 애트리뷰트 또한 설정된다면, 그것은 컨텐트 프로바이더에 쿼리하는 것에 대한 접근을 통제한다. 그리고 만약 writePermission 애트리뷰트가 설정된다면, 그것은 프로바이더의 데이터를 수정하는 것에 대한 접근을 통제한다.

퍼미션에 관한 더 많은 정보에 대해서는, 앞에 나왔던 퍼미션 섹션과 8장. "보안과 퍼미션"을 보라.

android:process

컨텐트 프로바이더가 실행되어야 하는 프로세스 이름. 보통 애플리케이션의 모든 컴포넌트는 애플리케이션을 위해 만들어진 디폴트 프로세스에

서 실행된다. 그 프로세스는 애플리케이션 패키지과 같은 이름을 가진다. 〈application〉 엘리먼트의 프로세스 애트리뷰트는 모든 컴포넌트들을 위한 다른 디폴트를 설정할 수 있다. 그러나 각각의 컴포넌트는 그것 자신의 process 애트리뷰트를 사용해서 디폴트를 오버라이드 할 수 있으며, 이것은 여러분으로 하여금 애플리케이션을 여러 프로세스에 걸쳐 분포시키게 하는 것을 가능하게 한다.

만약 이 애트리뷰트에 할당된 이름이 콜론(":")으로 시작된다면, 애플리케이션에 사적인private 신규 프로세스가 필요할 때 생성되며, 액티비티는 그 프로세스에서 실행된다. 만약 프로세스 이름이 소문자로 시작하면, 액티비티는 그 이름을 가지는 글로벌 프로세스 안에서 실행될 것이며 이는 그렇게 하도록 하는 퍼미션을 가지는 경우에 가능하다. 이것은 다른 애플리케이션에 있는 컴포넌트들이 프로세스를 공유하는 것을 가능하게 하며, 리소스 사용을 감소시킨다.

android:readPermission

클라이언트가 컨텐트 프로바이더에 쿼리하기 위해 가져야 하는 퍼미션. 또한 permission 과 writePermission 애트리뷰트를 보라.

android:syncable

컨텐트 프로바이더의 제어하에 있는 데이터가 서버 상의 데이터와 동기화되어야 하는 지 여부 – 만약 동기화되어져야 한다면 "참true" 그리고 아니면 "거짓false"이다.

android:writePermission

클라이언트가 컨텐트 프로바이더에 의해 제어되는 데이터를 변경하기 위해 가져야 하는 퍼미션. 또한 permission과 readPermission 애트리뷰트를 보라.

도입 :

API 레벨 1

더 보기 :

Content Providers

〈receiver〉 API Level 1

문법 :

```
<receiver android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >
    . . .
</receiver>
```

상위 엘리먼트 :

```
<application>
```

하위 엘리먼트 :

```
<intent-filer>
<meta-data>
```

설명 :

애플리케이션 컴포넌트 중 하나인 브로드캐스트 리시버(BroadcastReceiver 서브클래스)를 선언한다. 브로드캐스트 리시버는 애플리케이션의 다른 컴포넌트들이 실행되지 않을 때 조차도 다른 애플리케이션 또는 시스템에 의해 브로드캐스트되는 인텐트를 받기 위해 애플리케이션을 활성화한다.

시스템에게 브로드캐스트 리시버를 알리기 위한 두 가지 방법이 있다: 하나는 이 엘리먼트를 가지고 매니페스트 파일 안에 그것을 선언하는 것이다. 다른 하나는 코드 안에서 동적으로 리시버를 만들어서 Context.registerReceiver() 메쏘드를 사용해서 그것을 등록하는 것이다. 동적으로 만들어진 리시버에 대한 더 많은 것은 BroadcastReceiver 클래스 설명을 보라.

애트리뷰트 :

android:enabled

브로드캐스트 리시버가 시스템에 의해 인스턴스화 될 수 있는 지 여부 – 만약 될 수 있다면 "참true" 아니라면 "거짓false"이다. 디폴트 값은 "참true"이다.

〈application〉 엘리먼트는 브로드캐스트 리시버를 포함하는 모든 애플리케이션 컴포넌트들에게 적용되는 그것 자신의 enabled 애트리뷰트를 갖는다. 브로드캐스트 리시버가 활성화될 수 있게 하기 위해서는 〈application〉과 〈receiver〉 애트리뷰트 모두가 "참true"이 되어야 한다. 만약 어느 것 하나가 "거짓false"이라면, 그것은 비활성화된다; 그것은 인스턴스화될 수 없다.

android:exported

브로드캐스트 리시버가 그것의 애플리케이션 외부의 소스로부터 메시지를 받을 수 있는지 여부 – 만약 할 수 있으면 "참true" 아니라면 "거짓false"이다. 만약 "거짓false"이라면 브로드캐스트 리시버가 받을 수 있는 유일한 메시지는 동일 애플리케이션의 또는 동일 유저user ID를 가지는 애플리케이션의 컴포넌트에 의해 보내진 것들이다.

디폴트 값은 브로드캐스트 리시버가 인텐트 필터를 포함하는 지 여부에 따라 다르다. 아무 필터도 없다는 것은 그 리시버가 그것의 정확한 클래스 이름을 지정하는 인텐트 오브젝트에 의해서만 호출될 수 있다는 것을 의미한다. 이것은 (다른 것들이 일반적으로 클래스 이름을 모를 것이기 때문에) 리시버가 애플리케이션 내부 용도를 위해서만 의도되었다는 것을 의미한다. 그러므로 이런 경우에, 디폴트 값은 "거짓false"이다. 반면에, 최소한 하나의 필터가 있다는 것은 브로드캐스트 리시버가 시스템 또는 다른 애플리케이션의 브로드캐스트된 인텐트를 받도록 의도되었다는 것을 의미한다. 그러므로 디폴트 값은 "참true"이다.

이 애트리뷰트가 브로드캐스트 리시버의 외부 노출을 제한하는 유일한 방법은 아니다. 여러분은 또한 그 브로드캐스트리시버에 메시지를 보낼 수 있는 외부 엔티티를 제한하는 퍼미션을 사용할 수 있다(permission 애트리뷰트를 보라).

android:icon

브로드캐스트 리시버를 나타내는 아이콘. 이 애트리뷰트는 이미지 정의를 포함하는 드로어블drawable 리소스에 대한 레퍼런스로써 설정되어야 한다. 만약 설정되지 않으면, 전체 애플리케이션을 위해 지정된 아이콘이 대신 사용된다(〈application〉 엘리먼트의 icon 애트리뷰트를 보라).

브로드캐스트 리시버의 아이콘은 – 여기 또는 〈application〉 엘리먼트에서 설정되던 상관없이 – 모든 리시버의 인텐트 필터에 대한 디폴트 아이콘이다(〈intent-filter〉 엘리먼트의 icon 애트리뷰트를 보라).

android:label

브로드캐스트 리시버에 대한 사용자가 읽을 수 있는 라벨. 만약 이 애트리뷰트가 설정되지 않으면, 전체 애플리케이션을 위한 라벨 설정이 대신 사용된다. (〈application〉 엘리먼트의 label 애트리뷰트를 보라).

브로드캐스트 리시버의 라벨은 – 여기 또는 〈application〉 엘리먼트에서 설정되던 상관없이 – 또한 모든 리시버의 인텐트 필터에 대한 디폴트 라벨이다. (〈intent-filter〉 엘리먼트의 label 애트리뷰트를 보라).

라벨은 사용자 인터페이스의 다른 문자열처럼 로컬라이즈 될 수 있도록 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다. 하지만 편의상 여러분이 애플리케이션을 개발하는 동안 그것은 또한 원시 문자열로써 설정될 수 있다.

android:name

브로드캐스트 리시버, 즉 BroadcastReceiver의 서브클래스를 구현하는 클래스의 이름. 이것은 정확한 전체 클래스 이름이어야 한다("com.example.project.ReportReceiver" 와 같이). 하지만 단축형으로 이름의 첫 번째 문자가 마침표라면(예를 들어 ". ReportReceiver"), 그것은 〈manifest〉 엘리먼트에 지정된 패키지 이름 끝에 붙여진다.

디폴트는 없다. 이름은 지정되어야 한다.

android:permission

브로드캐스터가 브로드캐스트 리시버에게 메시지를 보내기 위해 가져야 하는 퍼미션의 이름. 만약 이 애트리뷰트가 설정되지 않으면, 〈application〉 엘리먼트의 permission 애트리뷰트에 의해 설정된 퍼미션이 브로드캐스트 리시버에 적용된다. 만약 둘 모두의 애트리뷰트가 설정되지 않으면, 그 리시버는 퍼미션에 의해 보호되지 않는다.

퍼미션에 관한 더 많은 정보에 대해서, 앞에 나왔던 퍼미션 섹션과 8장. 보안과 퍼미션을 보라.

android:process

브로드캐스트 리시버가 실행되어야 하는 프로세스의 이름. 보통, 애플리케이션의 모든 컴포넌트는 애플리케이션을 위해 생성된 디폴트 프로세스에서 실행된다. 그것은 애플리케이션 패키지와 같은 이름을 갖는다. 〈application〉 엘리먼트의 process 애트리뷰트는 모든 컴포넌트에 대해서 다른 디폴트를 설정

할 수 있다. 하지만 각각의 컴포넌트는 그것 자신의 process 애트리뷰트를 사용해서 디폴트를 오버라이드 할 수 있다. 그것은 다양한 프로세스에 걸쳐 여러분의 애플리케이션을 분포시키는 것을 가능하게 한다.

만약 이 애트리뷰트에 할당된 이름이 콜론(":")으로 시작된다면, 애플리케이션에 사적private인 신규 프로세스가 필요할 때 생성되며, 브로드캐스트 리시버는 그 프로세스에서 실행된다. 만약 프로세스 이름이 소문자로 시작된다면, 리시버는 그 이름을 가지는 글로벌 프로세스에서 실행되며, 이것은 그렇게 하도록 하는 퍼미션을 가지는 경우에 가능하다. 이것은 다른 애플리케이션의 컴포넌트들이 하나의 프로세스를 공유하게 하며, 리소스 사용을 감소시킨다.

도입 :

API 레벨 1

⟨service⟩ API Level 1

문법 :

```
<service android:enabled=["true" | "false"]
         android:exported[="true" | "false"]
         android:icon="drawable resource"
         android:label="string resource"
         android:name="string"
         android:permission="string"
         android:process="string" >
    . . .
</service>
```

상위 엘리먼트 :

```
<application>
```

하위 엘리먼트 :

```
<intent-filer>
<mea-data>
```

설명 :

애플리케이션의 컴포넌트들 중 하나로써, 서비스(Sevice 서브클래스)를 선언한다. 액티비티들과 달리, 서비스는 비쥬얼한 사용자 인터페이스가 없다. 그것은 오랫동안 실행되는 백그라운드 오퍼레이션이나, 또는 다른 애플리케이션에 의해

호출될 수 있는 풍부한 커뮤니케이션 API를 구현하기 위해 사용된다. 모든 서비스는 매니페스트 파일의 〈service〉 엘리먼트에 의해 표시되어야 한다. 그곳에 선언되지 않는 것은 시스템에게 보여지지 않으며, 결코 실행되지 않을 것이다.

애트리뷰트 :

android:enabled

서비스가 시스템에 의해 인스턴스화 될 수 있는 지 여부 – 그것이 될 수 있다면 "참true" 아니라면 "거짓false"이다. 디폴트 값은 "참true"이다.

〈application〉 엘리먼트는, 서비스를 포함하는 모든 애플리케이션 컴포넌트에 적용되는 그것 소유의 enabled 애트리뷰트를 가진다. 서비스가 활성화 되게하기 위해서는 〈application〉 과 〈service〉 애트리뷰트 모두 (디폴트로 그것들 모두가 그렇듯이) "참true"이어야 한다. 만일 어느 것 하나라도 "거짓false"이라면, 서비스는 비활성화 된다; 그것은 인스턴스화 될 수 없다.

android:exported

다른 애플리케이션의 컴포넌트가 서비스를 호출하거나 또는 그것과 상호작용할 수 있는 지 여부 – 만약 그것들이 할 수 있으면 "참true" 아니라면 "거짓false"이다. 값이 "거짓false" 일 때, 동일 애플리케이션 또는 동일 유저user ID를 갖는 애플리케이션의 컴포넌트들만이 서비스를 시작하거나 또는 그것이 바인드 할 수 있다.

디폴트 값은 서비스가 인텐트 필터를 포함하는 지 여부에 따라 다르다. 아무런 필터도 없다는 것은 그것의 정확한 클래스 이름을 지정하는 것에 의해서만 호출될 수 있다는 것을 의미한다. 이것은 서비스가 애플리케이션 내부 용도를 위해서만 의도되었다는 것을 의미한다(다른 것들이 클래스 이름을 모를 것이기 때문에). 그러므로 이런 경우에 디폴트 값은 "거짓false"이다. 반면에, 최소한 하나의 필터라도 있다는 것은 서비스가 외부 사용을 위해 의도되었다는 것을 의미한다. 그러므로 디폴트 값은 "참true"이다.

이 애트리뷰트가 다른 애플리케이션에게 서비스의 노출을 제한하는 유일한 방법은 아니다. 여러분은 또한 서비스와 상호작용할 수 있는 외부 엔티티를 제한하기 위해서 퍼미션을 사용할 수 있다(permission 애트리뷰트를 보라).

android:icon

서비스를 나타내는 아이콘. 이 애트리뷰트는 이미지 정의를 포함하는 드로어블drawable 리소스에 대한 레퍼런스로써 설정되어야 한다. 그것이 설정되지 않으면, 전체 애플리케이션에 대해 지정된 아이콘이 대신 사용된다(〈application〉 엘리먼트의 icon 애트리뷰트를 보라).

서비스의 아이콘은 – 여기서 설정되던 또는 〈application〉 엘리먼트에서 설정되던 상관없이 – 또한 모든 서비스의 인텐트 필터에 대한 디폴트 아이콘이다(〈intent-filter〉 엘리먼트의 icon 애트리뷰트를 보라).

android:label

사용자에게 보여질 수 있는 서비스의 이름. 이 애트리뷰트가 설정되지 않으면, 전체 애플리케이션에 대한 라벨 설정이 대신 사용된다(〈application〉 엘리먼트의 label 애트리뷰트를 보라).

서비스의 라벨은 – 여기서 설정되던 또는 〈application〉 엘리먼트에서 설정되던 상관없이– 또한 모든 서비스의 인텐트 필터에 대한 디폴트 라벨이다(〈intent-filter〉 엘리먼트의 label 애트리뷰트를 보라).

라벨은 사용자 인터페이스의 다른 문자열처럼 로컬라이즈될 수 있도록 문자열 리소스에 대한 레퍼런스로써 설정되어야 한다. 하지만 편의상 여러분이 애플리케이션을 개발하고 있는 동안 그것은 또한 원시 문자열로 설정될 수 있다.

android:name

서비스를 구현하는 Service 서브클래스의 이름. 이것은 전체 클래스 이름이어야 한다("com.example.project.RoomService"처럼). 하지만 단축형으로 이름의 첫 번째 캐릭터가 마침표인 경우(예를 들어 ".RoomService"), 〈manifest〉 엘리먼트에 지정된 패키지 이름 끝에 붙여진다.

디폴트는 없다. 이름은 지정되어야만 한다.

android:permission

서비스를 런치하거나 그것에 바인드하기 위해 어떤 엔티티가 가져야 하는 퍼미션의 이름. 만약 startsService() , bindService() , 또는 stopService()의 호출자가 이 퍼미션을 부여받지 않는다면, 그 메쏘드는 작동하지 않을 것이며 인텐트 오브젝트는 서비스에 전달되지 않을 것이다.

만약 이 애트리뷰트가 설정되지 않으면, 〈application〉 엘리먼트의

permission 애트리뷰트에 의해 설정된 퍼미션이 서비스에 적용된다. 만약 이 두개의 애트리뷰트가 설정되지 않는다면, 서비스는 퍼미션에 의해 보호되지 않는다.

퍼미션에 관한 더 많은 정보에 대해서, 앞에 나왔던 퍼미션 섹션과 8장. "보안과 퍼미션"을 보라.

android:process

서비스가 실행되는 프로세스 이름. 보통, 애플리케이션의 모든 컴포넌트는 애플리케이션을 위해 생성된 디폴트 프로세스에서 실행된다. 그것은 애플리케이션 패키지와 같은 이름을 갖는다. 〈application〉 엘리먼트의 process 애트리뷰트는 모든 컴포넌트에 대한 다른 디폴트를 설정할 수 있다. 하지만 컴포넌트는 그것 자신의 process 애트리뷰트를 사용해서 디폴트를 오버라이드 할 수 있으며, 이것은 다양한 프로세스에 걸쳐 여러분의 애플리케이션이 분포되는 것을 가능하게 한다.

만약 이 애트리뷰트에 할당된 이름이 콜론(":")으로 시작한다면, 애플리케이션에 사적private인 신규 프로세스가 필요할 때 생성되며, 서비스는 그 프로세스에서 실행된다. 만약 프로세스 이름이 소문자로 시작한다면, 서비스는 그 이름을 가지는 글로벌 프로세스에서 실행되며 이는 그렇게 하도록 하는 퍼미션을 가진 경우 가능하다. 이것은 다른 애플리케이션에 있는 컴포넌트가 하나의 프로세스를 공유하는 것을 가능하게 하며, 리소스 사용을 줄인다.

도입 :

API 레벨 1

더 보기 :

```
<application>
<activity>
```

〈supports-screens〉 API Level 4

문법 :

```
<supports-screens android:smallScreens=["true" | "false"]
                  android:normalScreens=["true" | "false"]
                  android:largeScreens=["true" | "false"]
                  android:anyDensity=["true" | "false"] />
```

상위 엘리먼트 :

<manifest>

설명 :

여러분으로 하여금 애플리케이션이 지원하는 스크린 크기를 지정하게 한다. 디폴트로 최신 애플리케이션(API 레벨 4 또는 그 이상을 사용하는)은 모든 스크린 크기를 지원하므로 여기서 명시적으로 특정 스크린 크기를 비활성화해야 한다; 오래된 애플리케이션은 "normal" 스크린 크기만을 지원하는 것으로 가정된다. 스크린 크기는 밀도density와 분리된 축이다. 스크린 크기는 밀도 조정이 적용된 후에 애플리케이션에 이용가능한 픽셀들로 결정된다.

타겟이 되는 다바이스 스크린 밀도에 기반하여, 안드로이드 프레임워크는 에셋을 0.75 비율 만큼 크기 축소(낮은 dpi 스크린)하거나 1.6 비율 만큼 크기를 확대(높은 dpi 스크린)할 것이다. 스크린 밀도는 인치당 도트 개수dots-per-inch (dpi)로 표시된다.

더 많은 정보에 대해서는 Multiple Screens Supprt(http://developer.android.com/guide/practices/screens_support.html)를 보라.

애트리뷰트 :

android:smallScreens

애플리케이션이 더 작은 스크린 외형form-factors을 지원하는 지 여부를 가리킨다. small 스크린은 "normal"(전형적인 HVGA) 스크린 보다 더 작은 가로 세로 비율을 가지는 스크린으로 정의된다. small 스크린을 지원하지 않는 애플리케이션은 작은 스크린 디바이스에서는 이용가능하지 않다. 왜냐하면, 그 디바이스 상의 플랫폼은 그런 애플리케이션이 더 작은 스크린에 대해서 거의 동작할 수 없게 하기 때문이다. API 레벨 4 또는 그 이상을 사용하는 애플리케이션은 디폴트로 이 값을 "참true"으로 설정하고 다른 애플리케이션은 "거짓false"로 설정한다.

android:normalScreens

애플리케이션이 "normal" 스크린 외형form-factors을 지원하는 지 여부를 가리킨다. 전통적으로 "normal" 스크린은 HVGA 중간 밀도 스크린이지만, WQVGA 낮은 밀도와 WVGA 높은 밀도 또한 "normal" 크기로 간주된다.

이 애트리뷰트는 디폴트로 "참true"이며, 애플리케이션들은 현재 그 방식대로 그대로 나두어야 한다.

android:largeScreens

애플리케이션이 더 큰 스크린 외형form-factors을 지원할 수 있는지 여부를 가리킨다. large 스크린은 "normal" 폰 스크린 보다 훨씬 더 큰 스크린으로 정의된다. 따라서 이것은 large 스크린을 잘 사용하기 위해 애플리케이션 영역에서의 특별한 주의를 요구할 지도 모른다. large 스크린을 지원하지 않는 애플리케이션은 최초의 디자인되었던 크기를 유지하기 위해 그러한 스크린 상에서는 "우표postage stamp"처럼 놓이게 될 것이다. API 레벨 4 또는 그 이상을 사용하는 애플리케이션은 디폴트로 이 값이 "참true"이고, 다른 애플리케이션은 "거짓false"이다.

android:anyDensity

애플리케이션이 임의의 스크린 밀도를 수용할 수 있는 지를 가리킨다. 더 오래된 애플리케이션(API 레벨 4 이전)은 모든 밀도를 지원할 수 없는 것으로 가정한다. 따라서, 이 애트리뷰트 값은 디폴트로 "거짓false"이다. API 레벨 4 또는 그 이상을 사용하는 애플리케이션은 모든 밀도를 지원할 수 있는 것으로 가정하고 이 애트리뷰트 값은 디폴트로 "참true"이다. 여러분은 여기서 명시적으로 여러분의 지원 능력을 지원할 수 있다.

도입 :

API 레벨 4

더 보기 :

Multiple Screens Support
DisplayMetrics

〈uses-configuration〉 API Level 3

문법 :

```
<uses-configuration android:reqFiveWayNav=["true" | "false"]
                    android:reqHardKeyboard=["true" | "false"]
                    android:reqKeyboardType=["undefined" | "nokeys"
                                        | "qwerty" | "twelvekey"]
                    android:reqNavigation=["undefined" | "nonav"
                                | "dpad" | "trackball" | "wheel"]
```

```
                    android:reqTouchScreen=["undefined" | "notouch"
                                | "stylus" | "finger"]  />
```

상위 엘리먼트 :

```
<manifest>
```

설명 :

애플리케이션이 필요로 하는 하드웨어와 소프트웨어의 기능을 가리킨다. 예를 들어 애플리케이션이 물리적 키보드 또는 트랙볼 같은 특정 네비게이션 디바이스를 요구한다는 것을 지정할 수도 있다. 그 명세는 애플리케이션이 작동되지 않을 디바이스 상에 그것을 설치하는 것을 피하기 위해 사용된다.

만약 애플리케이션이 서로 다른 디바이스 설정들을 가지고 작업할 수 있다면, 그것은 각각에 대한 별도의 〈uses-configuration〉 선언을 포함해야 한다. 각각의 선언은 완전해야 한다. 예를 들어 만약 애플리케이션이 다섯 방향의 네비게이션 컨트롤과, 손가락을 사용해서 작동될 수 있는 터치 스크린, 그리고 표준 QWERTY키보드나 대부분의 폰에서 발견되는 것과 같은 12-key 숫자 키패드 중 하나를 요구한다면, 그것은 다음과 같은 두 개의 〈uses-configuration〉 엘리먼트를 사용해서 이러한 요구조건을 지정할 할 것이다.

```
<uses-configuration android:reqFiveWayNav="true"
                    android:reqTouchScreen="finger"
                    android:reqKeyboardType="qwerty" />
<uses-configuration android:reqFiveWayNav="true"
                    android:reqTouchScreen="finger"
                    android:reqKeyboardType="twelvekey" />
```

애트리뷰트 :

android:reqFiveWayNav

애플리케이션이 다섯 방향의 네비게이션 컨트롤을 요구하는 지 여부 – 만약 그렇다면 "참true" 아니라면 "거짓flase"이다. 다섯 방향 컨드롤은 선택을 위, 아래, 우측, 또는 좌측, 그리고 또한 현재 선택을 부르는 방법을 제공하는 것이다. 그것은 D-pad(방향 패드), 트랙볼, 또는 다른 디바이스가 될 수 있다.

만약 애플리케이션이 방향 컨트롤을 요구하지만 특정 타입의 컨트롤을 요구하지 않는다면, 그것은 이 애트리뷰트를 "참true"로 설정해서 reqNavigation 애트리뷰트를 무시할 수 있다. 하지만 만약 그것이 특정 타입의 방향 컨트롤을

요구한다면, 그것은 이 애트리뷰트를 무시하고 대신 reqNavigation 을 설정할 수 있다.

android:reqHardKeyboard

애플리케이션이 하드웨어 키보드를 요구하는 지 여부 – 만약 그렇다면 "참true" 아니라면 "거짓flase"이다.

android:reqKeyboardType

만약 있다면, 애플리케이션이 요구하는 키보드 타입. 이 애트리뷰트는 하드웨어와 소프트웨어 키보드를 구별하지 않는다. 만약 특정 타입의 하드웨어 키보드가 요구된다면, 그 타입을 이곳에 지정하고 또한 reqHardKeyboard 애트리뷰트를 "참true"로 설정하라.

그 값은 아래의 문자열 중 하나가 되어야 한다.

값	설명
"undefined"	애플리케이션이 키보드를 요구하지 않는다.(키보드 요구조건이 정의되지 않다.) 이것이 디폴트 값이다.
"nokeys"	애플리케이션이 키보드를 요구하지 않는다.
"qwerty"	애플리케이션이 표준 QWERTY 키보드를 요구한다.
"twelvekey"	애플리케이션이 대부분의 폰에 있는 것과 비슷한 0부터 9까지의 숫자키에 더해 별표(*)와 파운드(#) 키를 가진12-key 키패드를 요구한다.

android:reqNavigation

만약 있다면, 애플리케이션에 의해 요구되는 네비게이션 디바이스. 그 값은 아래의 문자열 중 하나가 되어야 한다.

값	설명
"undefined"	애플리케이션이 어떤 타입의 네비게이션 컨트롤도 요구하지 않는다.(그 네비게이션 요구조건이 정의되지 않다.) 이것이 디폴트 값이다.
"nonav"	애플리케이션이 네비게이션 컨트롤을 요구하지 않는다.
"dpad"	애플리케이션이 네비게이션을 위한 D-pad(방향 패드)를 요구한다.
"trackball"	애플리케이션이 네비게이션을 위한 트랙볼을 요구한다.
"wheel"	애플리케이션이 네비게이션 휠을 요구한다.

만약 애플리케이션이 네비게이션 컨트롤을 요구하지만, 정확한 컨트롤 타입은 중요하지 않다면, 이 값을 설정하기 보다는 차라리 reqFiveWayNav애트리뷰트를 "참true"으로 설정할 수 있다.

android:reqTouchScreen

만약 있다면, 애플리케이션이 요구하는 터치 스크린 타입. 그 값은 아래의 문

자열 중 하나가 되어야 한다.

값	설명
"undefined"	애플리케이션이 터치 스크린을 요구하지 않는다. (터치 스크린 요구조건이 정의되지 않다.) 이것이 디폴트 값이다.
"notouch"	애플리케이션이 터치 스크린을 요구하지 않는다.
"stylus"	애플리케이션이 스타일러스(stylus)를 사용해서 동작되는 터치 스크린을 요구한다.
"finger"	애플리케이션이 손가락을 사용해서 동작될 수 있는 터치 스크린을 요구한다.

도입 :

API 레벨 3

더보기 :

〈activity〉 엘리먼트의 configChanges 애트리뷰트

ConfigurationInfo

〈uses-feature〉 API Level 4

문법 :

```
<uses-feature android:glEsVersion="integer"
              android:name="string" />
```

상위 엘리먼트 :

```
<manifest>
```

설명 :

이 엘리먼트는 애플리케이션에 의해서 사용되는 특정 기능을 선언한다. 안드로이드는 모든 안드로이드 디바이스에 의해서 동등하게 지원되지 않을 수 있는 몇가지 기능을 제공한다. 〈uses-sdk〉 엘리먼트와 비슷한 방법으로, 이 엘리먼트는 애플리케이션으로 하여금 그것이 사용하는 디바이스에 따라 변하는 기능을 지정하도록 허용한다. 이런 방법으로, 애플리케이션은 그 기능을 지원하지 않는 디바이스 상에서는 설치되지 않을 것이다.

예를 들어, 애플리케이션은 자동auto 포커스 능력을 가진 카메라를 필요로 한다는 것을 지정할 지도 모른다. 만약, 디바이스가 자동 포커스를 가지는 카메라를 지원하지 않는다면 그 디바이스는 애플리케이션의 설치를 허용하지 않을 것이다.

엄격한 디바이스 호환성을 유지하기 위해서, 여러분의 애플리케이션이 사용하는 모든 기능을 선언하기 위해서 여러분이 이 엘리먼트를 사용하는 것은 매우 중요하다. 기능을 선언하지 않게 되면 결과적으로 여러분의 애플리케이션은 그 기능을 지원하지 않는 디바이스 상에서 설치되어서 여러분의 애플리케이션은 작동하지 않을 수도 있다.

특정 기능에 대해서, 여러분으로 하여금 사용된 OpenGL 버전(glEsVersion을 사용하여 선언됨)과 같이 기능의 버전을 정의하도록 허용하는 특정 애트리뷰트가 존재할 수도 있다. 카메라 자동 포커스와 같이, 디바이스에 존재하거나 또는 존재하지 않는 다른 기능은 name 애트리뷰트를 사용해서 선언된다.

안드로이드 디바이스들 사이에서 달라질 수 있는 소프트웨어 또는 하드웨어 기능들은 이 페이지에서 나열될 것이다. 만약 여러분이 여러분의 애플리케이션에서 사용하는 어떤 기능들을 본다면, 여러분은 각각에 대해서 〈uses-feature〉를 포함시켜야 한다. 예를 들어 여러분의 애플리케이션이 디바이스 카메라를 사용한다면, 여러분은 AndroidManifest.xml에 다음을 포함시켜야 한다.

```
<uses-feature android:name="android.hardware.camera" />
```

만약 여러분이 "android.hardware.camera"를 선언한다면, 자동 포커스를 이용할 수 있는 지 여부와 상관없이 여러분의 애플리케이션은 카메라를 포함하는 모든 디바이스와 호환가능한 것으로 간주된다. 만약 여러분이 또한 자동 포커스 기능(카메라 API를 통해 이용가능)을 사용한다면, 여러분은 "android.hardware.camera.autofocus" 기능을 선언하는 추가적인 〈uses-feature〉 엘리먼트를 포함시킬 필요가 있다. 또한 여러분은 여전히 CAMERA 퍼미션을 요구해야 한다는 것에 주의하라. 여러분의 애플리케이션에 의해 사용되는 기능을 선언하는 것은 적합한 디바이스 호환성을 보장하는 반면에, 퍼미션을 요구하는 것은 여러분의 애플리케이션이 적합한 하드웨어와 소프트웨어에 접근하는 것을 허용한다.

비록 〈uses-feature〉 엘리먼트는 단지 API 레벨 4 또는 그 이상을 실행하는 디바이스에 대해서 활성화된다 할 지라도, "3" 또는 그 이하의 minSdkVersion을 선언하는 애플리케이션에 대해서 이것을 포함하는 것이 안전하다. 낮은 버전의 플랫폼을 실행하는 디바이스는 간단히 이 엘리먼트를 무시할 것이다. 하지만, 더 새로운 디바이스는 이 엘리먼트를 인식하고 디바이

스가 그 기능을 지원하는 지 여부에 근거하여 설치 제약을 강제할 것이다.

> **Note**
> 여러분의 애플리케이션에 의해서 요구되는 각 기능에 대해서, 여러분은 새로운 〈uses-feature〉 엘리먼트를 포함시켜야 한다. 여러 기능들이 이 엘리먼트의 한 인스턴스에서 선언될 수 없다.

애트리뷰트 :

android:glEsVersion

애플리케이션이 필요로 하는 GLES 버전. 상위 16 비트는 메이저 숫자를 나타내고 하위 16 비트는 마이너 숫자를 나타낸다. 예를 들어 0x00000102를 가리키는 GL 1.2에 대해서, 실제 값은 0x00010002로 설정되어야 한다.

android:name

애플리케이션이 필요로 하는 기능의 이름. 값은 다음에 나오는 허용되는 문자열들 중의 하나여야 한다.

기능	값	설명
Camera	"android.hardware.camera"	애플리케이션이 카메라를 필요로 한다.
	"android.hardware.camera.autofocus"	애플리케이션이 자동 포커스 기능을 가진 카메라를 필요로 한다. 필요 조건으로, "android.hardware.camera"가 별도의 〈uses-feature〉 엘리먼트에 선언되어야 한다.
	주의 : 카메라 퍼미션을 요구하지만 〈uses-feature〉 엘리먼트를 사용하여 카메라 기능을 선언하지 않는 애플리케이션은 (자동 포커스와 같은) 모든 카메라 기능을 사용하기 위한 것으로 가정할 것이다. 따라서, 애플리케이션은 모든 기능을 지원하지 않는 디바이스와 호환이 불가능할 것이다. 여러분의 애플리케이션이 필요로 하는 카메라 기능만을 선언하기 위해 〈uses-feature〉를 사용하길 부탁한다.	

도입 :

API 레벨 4

더보기 :

ConfigurationInfo

〈uses-library〉 API Level 1

문법 :

```
<uses-library android:name="string" />
```

상위 엘리먼트 :

```
<application>
```

설명 :

애플리케이션에 링크되어야 하는 공유 라이브러리를 지정한다. 이 엘리먼트는 시스템에게 패키지를 위한 클래스 로더에 라이브러리 코드를 포함할 것을 지시한다.

(android.app, android.content, android.view, 그리고 android.widget과 같은) 모든 안드로이드 패키지는 모든 애플리케이션이 자동으로 링크하는 디폴트 라이브러리안에 있다. 하지만 (맵과 awt와 같은) 몇몇 패키지는 자동으로 링크되지 않는 별도의 라이브러리에 있다. 어떤 라이브러리가 해당 패키지 코드를 포함하는 지를 결정하기 위해서 여러분이 사용하는 패키지에 대한 문서를 참고하라.

애트리뷰트 :

android:name

라이브러리의 이름.

도입 :

API 레벨 1

〈uses-permission〉 API Level 1

문법 :

```
<uses-permission android:name="string" />
```

상위 엘리먼트 :

```
<manifest>
```

설명 :

애플리케이션이 그것이 정확하게 동작하도록 하기 위해 부여받아야 하는 퍼미션을 요청한다. 퍼미션은 애플리케이션이 실행되는 동안 부여되는 것이 아니

라, 그것이 설치될 때 부여된다.

퍼미션에 관한 더 많은 정보에 대해서, 앞에 나왔던 퍼미션 섹션과 8장. "보안과 퍼미션"을 보라. 기본 플랫폼에 의해 정의된 퍼미션 리스트는 android.Manifest.permission 에서 찾을 수 있다.

애트리뷰트 :

android:name

퍼미션의 이름. 그것은 〈permission〉 엘리먼트를 사용해서 애플리케이션에서 정의된 퍼미션, 다른 애플리케이션에 의해 정의된 퍼미션, 또는 "android.permission.CAMERA"나 "android.permission.READ_CONTACTS"와 같은 표준 시스템 퍼미션 중의 하나가 될 수 있다. 이들 예제가 보여주듯이, 퍼미션 이름은 일반적으로 접두어로 패키지 이름을 포함한다.

도입 :

API 레벨 1

더 보기 :

<permission>

〈uses-sdk〉 API Level 1

문법 :

```
<uses-sdk android:minSdkVersion="integer"
          android:maxSdkVersion="integer"
          amdroid:targetSdkVersion="integer" />
```

상위 엘리먼트 :

<manifest>

설명 :

여러분으로 하여금 API 레벨 숫자에 의해 하나 이상의 버전의 플랫폼과 애플리케이션의 호환성을 표현하게 한다. 애플리케이션에 의해서 표현되는 API 레벨은 주어진 안드로이드 시스템의 API 레벨과 비교될 것이며, 다른 안드로이드 디바이스들간에 다를 수 있다.

이 애트리뷰트의 이름에도 불구하고 이 애트리뷰트는 SDK 버전 숫자가 아니

라 API 레벨을 지정하기 위해서 사용된다. API 레벨은 항상 정수integer이다; SDK 버전은 (1.5와 같이) 메이저 부분과 마이너 부분으로 나뉠 수도 있다. 여러분은 SDK 버전 숫자로부터 API 레벨을 얻을 수 없다(예를 들어 API 레벨은 메이저 버전 또는 메이저 버전과 마이너 버전의 합과 같지 않다).

더 많은 정보에 대해서, 안드로이드 API 레벨(http://developer.android.com/guide/appendix/api-levels.html)과 17장. "애플리케이션 버전 부여"를 읽어라.

안드로이드 마켓과 〈uses-sdk〉 애트리뷰트

안드로이드 마켓은 사용자가 자신의 디바이스와 호환되는 애플리케이션만을 보고 다운로드할 수 있도록 사용자에게 보여지는 애플리케이션들을 필터한다. 마켓이 애플리케이션을 필터하는 방법 중 하나가 안드로이드 버전 호환성에 의한 것이다. 이것을 하기 위해, 마켓은 애플리케이션의 버전 호환성 범위를 확정하기 위해 각각의 애플리케이션의 매니페스트에 있는 〈uses-sdk〉 애트리뷰트를 검사하며, 그런 다음에 사용자의 안드로이드 시스템 버전에 API 레벨과의 비교에 기초하여 애플리케이션을 보여주거나 숨긴다.

애트리뷰트 :

android:minSdkVersion

애플리케이션이 실행되기 위해 요구되는 안드로이드 API의 최소 레벨을 지정하는 정수integer. 안드로이드 시스템은 시스템의 API 레벨이 이 애트리뷰트에 지정된 값보다 더 낮다면, 사용자로 하여금 애플리케이션을 설치하는 것을 막을 것이다. 여러분은 항상 이 애트리뷰트를 선언해야 한다.

Caution

여러분이 이 애트리뷰트를 선언하지 않으면 값이 "1"이 가정되며 이는 여러분의 애플리케이션이 모든 버전의 안드로이드와 호환된다는 것을 의미한다. 여러분의 애플리케이션이 모든 버전과 호환되지 않는다면 (예를 들어 그 애플리케이션이 안드로이드 API 레벨 3에서 도입된 API들을 사용한다면) 그리고 여러분이 적절한 minSdkVersion을 선언하지 않았다면, 그것이 더 낮은 API 레벨을 가진 시스템 상에서 설치될 때 실행되는 동안 망가질 것이다. 이러한 이유로 minSdkVersion 애트리뷰트에 적절한 API 레벨을 선언하는 것을 확인하라.

android:targetSdkVersion API Level 4

애플리케이션이 타겟으로 하고 있는 API 레벨을 지정하는 정수integer.

이 애트리뷰트가 설정되면, 애플리케이션은 더 낮은 버전(minSdkVersion까지) 상에서 실행될 수 있지만, 여기서 명시된 버전에서 명시적으로 테스트되었음을 의미한다. 이 타겟 버전을 지정하는 것은 플랫폼으로 하여금 타겟 버전을 위해서 필요하지 않은 (포워드-호환성을 유지하기 위해 켜질 수 있는) 호환성 설정을 비활성화 하도록 허용하고 또는 더 오래된 애플리케이션에 이용가능하지 않은 더 새로운 기능을 활성화 하도록 허용한다. 이것은 여러분이 다른 버전의 플랫폼에 대한 다른 기능을 프로그램할 수 있다는 것을 의미하지 않는다 - 이것은 단지 플랫폼에게 여러분이 타겟 버전에 대해서 테스트하였고 플랫폼은 타겟 버전과 포워드-호환성을 유지하기 위해 추가적인 작업을 수행해서는 안된다는 것을 알려야 한다.

도입 : API 레벨 4

android:maxSdkVersion API Level 4

애플리케이션이 실행되도록 설계된 최대 API 레벨을 지정하는 정수integer.

안드로이드 1.5, 1.6, 2.0, 그리고 2.0.1에서, 시스템은 애플리케이션을 설치할 때, 그리고 시스템을 업데이트한 후 애플리케이션을 재검증할 때 이 애트리뷰트의 값을 검사한다. 두 가지 경우에서, 만약 애플리케이션의 android:maxSdkVersion 애트리뷰트가 시스템 자신에 의해 사용된 API 레벨보다 낮다면, 시스템은 애플리케이션이 설치되는 것을 허용하지 않을 것이다. 시스템을 업데이트한 후 재검증하는 경우에, 이것은 효과적으로 디바이스에서 여러분의 애플리케이션들을 제거한다.

시스템 업데이트 이후 여러분의 애플리케이션에 이 애트리뷰트가 어떻게 영향을 미칠 수 있는가를 분명히 하기 위해, 다음과 같은 예제를 고려하라.

매니페스트에 android:maxSdkVersion="5"라고 선언하고 있는 하나의 애플리케이션이 안드로이드 마켓에 배포되었다. 안드로이드 1.6(API 레벨 4)이 실행되는 디바이스의 사용자가 그 애플리케이션을 다운로드하고 설치한다. 몇 주 후, 사용자는 안드로이드 2.0(API 레벨 5)으로 OTA(over-the-air) 시스템 업데이트를 수신한다. 그 업데이트가 설치된 후, 시스템은 애플리케이션의 android:maxSdkVersion을 검사하고, 그것을 성공적으로 재검증한다.

애플리케이션은 제 기능을 한다. 하지만, 얼마 후 디바이스는 다른 시스템 업데이트를 수신한다. 이번에는 안드로이드 2.0.1(API 레벨 6)이다. 업데이트 후, 시스템은 애플리케이션을 다시 유효화할 수 없다. 왜냐하면 시스템 자신의 API 레벨(6)이 애플리케이션에 의해 지원되는 최대치(5)보다 이제 더 크기 때문이다. 시스템은 디바이스에서 그것을 효과적으로 제거함으로써, 사용자에게 애플리케이션이 보여지는 것을 차단한다.

> **Warning:** 이 애트리뷰트를 선언하는 것은 권장되지 않는다. 첫째로, 애플리케이션이 릴리즈되었기 때문에 신규 버전의 안드로이드 플랫폼에 여러분의 애플리케이션의 배치를 차단하기 위한 수단으로써 이 애트리뷰트를 지정하는 것은 의미가 없다. 계획적으로, 신규 플랫폼 버전은 완전하게 이전 버전과 호환된다. 여러분의 애플리케이션이 오직 표준 API들만을 사용하고 개발 모범 사례를 따른다면, 그것은 신규 버전에서 제대로 작동해야 한다. 둘째로, 몇 가지 경우에, 이 애트리뷰트를 선언하는 것은 더 높은 API 레벨로 시스템이 업데이트된 이후, 사용자의 디바이스에서 여러분의 애플리케이션이 제거되는 결과를 초래할 수 있다. 여러분의 애플리케이션이 설치될 가능성이 있는 대부분의 디바이스는 OTA(over the air)로 주기적인 시스템 업데이트를 수신할 것이다. 그러므로 이 애트리뷰트를 셋팅하기 전에 여러분의 애플리케이션에 대한 그것의 영향을 고려해야 한다.

도입 : API 레벨 4

(안드로이드 2.0.1 을 넘어선) 미래의 안드로이드 버전은 설치 또는 개검증하는 동안 android:maxSdkVersion 애트리뷰트를 더아상 검사하거나 또는 강제하지 않을 것이다. 하지만, 안드로이드 마켓은 사용자에게 다운로드 가능한 애플리케이션을 보여줄 때, 하나의 필터로써 이 애트리뷰트를 계속 사용할 것이다.

도입 :

API 레벨 1

Section | 10

그래픽

10.1. 개요

10.2. 2차원 그래픽

10.3. OpenGL을 사용하는 3차원 그래픽

10. 그래픽

안드로이드 그래픽은 커스텀 2D 그래픽 라이브러리와 고사양의 3D 그래픽을 위한 OpenGL ES 1.0에 의해 구동된다. 가장 일반적인 2D 그래픽 API들은 drawable 패키지에서 찾을 수 있다. OpenGL API는 Khronos OpenGL ES 패키지로부터 유용하게 되었으며, 더불어 약간의 안드로이드 OpenGL 유틸리티들이 추가되었다.

프로젝트를 시작할 때, 여러분의 그래픽에 대한 요구demand가 무엇인지를 정확하게 고려하는 것이 중요하다. 많은 그래픽 작업들은 많은 기술들을 사용하여 이루어진다. 예를 들어 다소 정적인 애플리케이션에 대한 그래픽과 애니메이션은 인터랙티브 게임이나 3D 렌더링에 대한 그래픽과 애니메이션과는 매우 다르게 구현되어야 한다.

여기서 우리는 여러분이 안드로이드상에서 그래픽을 그리기 위해 가질 수 있는 몇 가지 옵션과 그것들이 어떤 작업에 가장 잘 맞는지에 대해 논의할 것이다.

만약 여러분이 특별히 3D 그래픽을 그리는 것에 대한 정보를 찾고 있다면, 이 페이지는 그다지 도움을 주지 못할 것이다. 하지만 캔버스를 사용해서 "드로잉하기(on Drawing with a Canvas)"에 있는 정보는 여러분에게 뷰 계층구조에 드로잉drawing 하는 방법에 대한 빠른 아이디어를 제공할 것이다. OpenGL ES API에서 제공되는 안드로이드 3D 그래픽 유틸리티에 대한 더 많은 정보에 대해서는 "OpenGL을 사용하는 3D(3D with OpenGL)"를 읽고, 다른 OpenGL 문서들을 참고하라.

개요

여러분의 옵션을 고려하라

2D 그래픽을 그릴 때, 여러분은 보통 다음의 두 가지 방법 중 하나로 그것을 할 것이다.

a. 여러분 레이아웃의 뷰 오브젝트에 여러분의 그래픽 또는 애니메이션을 그리기. 이런 방식으로 여러분의 그래픽을 그리는 것은 시스템의 일반적인 뷰 계층구조의 드로잉 절차에 의해 처리된다 – 여러분은 간단히 그 뷰 안으로 그래픽이 들어가도록 정의한다.

b. 캔버스Canvas에 직접적으로 여러분의 그래픽을 그리기. 이 방법은 여러분이 개인적으로 적절한 클래스의 draw() 메쏘드(그것에 여러분의 캔버스(Canvas)를 전달함), 또는 (drawPicture()와 같은) 캔버스 draw...() 메쏘드들 중의 하나를 호출하는 것이다. 그렇게 함으로써 여러분은 임의의 애니메이션 또한 제어하게 된다.

옵션 "a"의 뷰에 그리기는 여러분이 동적으로 바꿀 필요가 없고 성능 집약적인 게임에 속하지 않는 간단한 그래픽을 그릴 때 최선의 선택이다. 예를 들어 다른 정적 애플리케이션 안에서 정적 그래픽이나 미리 정의된 애니메이션을 표현하길 원할 때, 여러분은 뷰안에서 여러분의 그래픽을 드로잉해야 한다. "뷰 안에서의 간단한 그래픽Simple Graphics Inside a View"를 읽어보기 바란다.

옵션 "b"의 캔버스에 그리기는, 여러분의 애플리케이션이 규칙적으로 스스로를 다시 그려야 할 필요가 있을 때 더 좋다. 기본적으로 모든 비디오 게임은 그것 자신의 캔버스Canvas에 드로잉되어야 한다. 하지만 이것을 하는 데에는 한 가지 이상의 방법이 있다.

- 여러분의 액티비티와 동일한 쓰레드에서 여러분의 레이아웃 내에 커스텀 뷰를 생성하고, invalidate()를 호출하고, 그런 다음 onDraw() 콜백을 처리한다.
- 또는 별도의 쓰레드에서 거기에서 여러분은 서피스뷰SurfaceViw를 관리하고, 여러분의 쓰레드가 수용할 수 있는 한 빠르게 캔버스에 그리기를 수행한다.

"캔버스를 사용해서 그리기Draw with a Canvas" 를 읽는 것에서 시작하라.

뷰 안에서의 간단한 그래픽

만약 여러분이 약간의 간단한 그래픽(이미지, 모양(shape), 컬러, 미리 정의된 애니메이션 등)을 드로잉하게 된다면, 여러분은 아마도 단지 뷰의 백그라운드에 그리거나, 또는 여러분의 레이아웃 내의 이미지뷰ImageView의 컨텐트에 그릴 것이다. 이런 경우, 여러분은 이 문서의 나머지 부분을 건너 뛰고, 2D 그래픽 문서에 있는 그래픽과 애

니메이션을 그리는 방법을 배워라.

캔버스를 사용해서 그리기

여러분이 전문화된 드로잉 및/또는 그래픽 애니메이션에 대한 제어를 수행하는 애플리케이션을 작성할 때, 여러분은 캔버스를 통해 드로잉함으로써 그것을 해야 한다. 캔버스는 여러분의 그래픽이 그려지는 실제 표면에 대한 하나의 프리텐스pretense 또는 인터페이스로써, 여러분을 위한 작업을 수행한다 – 그것은 여러분이 사용할 모든 "draw" 호출을 보유한다. 캔버스Canvas를 통해 여러분이 드로잉하는 것은 그 아래 놓여 있는 비트맵(Bitmap) 위에서 실제로 수행되며, 그 비트맵은 윈도우 안에 위치하게 된다.

여러분이 만약 onDraw() 콜백 메쏘드 내에서 드로잉을 할 경우, 여러분을 위한 캔버스가 제공되며 여러분은 그 캔버스 위에 여러분의 드로잉 호출들을 놓으면 된다. 여러분이 서피스뷰SurfaceView 오브젝트를 다룰 때, SurfaceHolder.lockCanvas()로부터 캔버스를 또한 얻을 수 있다(이러한 모든 시나리오는 이후 섹션에서 논의된다). 하지만 여러분이 새로운 캔버스를 만들어야 할 필요가 있다면, 드로잉이 그 위에서 실제로 수행되는 비트맵을 정의해야 한다. 비트맵은 항상 캔버스를 위해 필요하다. 다음과 같이 새로운 캔버스를 만들 수도 있다.

```
Bitmap b = Bitmap.createBitmap(100, 100, Bitmap.Config.ARGB_8888);
Canvas c = new Canvas(b);
```

이제 여러분의 캔버스는 정의된 비트맵 위에 그릴 것이다. 그 캔버스를 사용해서 그 위에 드로잉을 한 이후, 여러분은 Canvas.drawBitmap(Bitmap,…) 메쏘드들 중 하나를 사용해서 여러분의 비트맵을 또 다른 캔버스로 이동시킬 수 있다. View.onDraw()나 SurfaceHolder.lockCanvas()에 의해 여러분에게 제공되는 캔버스를 통해 최종 그래픽을 드로잉하는 것이 권장된다(이후의 섹션들을 보라).

캔버스 클래스는 drawBitmap(…), drawRect(…), drawText(…) 등과 같이 여러분이 사용할 수 있는 그것 자신의 드로잉 메쏘드 집합을 가지고 있다. 여러분이 사용할 지도 모를 다른 클래스들 역시 draw() 메쏘드들을 가지고 있다. 예를 들어 아마도 여러분은 캔버스 위에 놓고 싶은 약간의 드로어블Drawable 오브젝트

를 가질 수 있다. 드로어블Drawable은 아규먼트로 여러분의 캔버스를 가지는 그것 자신의 draw() 메쏘드를 가지고 있다.

뷰에서(On a View)

만약 여러분의 애플리케이션이 많은 양의 프로세싱이나 프레임-레이트frame-rate 속도(아마도 체스 게임, 스네이크(snake) 게임, 또는 느리게 애니메이션되는 애플리케이션에 적합한)를 요구하지 않는다면, 여러분은 커스텀 뷰 컴포넌트를 생성하고 View.onDraw()에서 캔버스를 사용해 드로잉하는 것을 고려해야 한다. 그렇게 하는 것의 가장 편리한 측면은 안드로이드 프레임워크가 여러분의 드로잉 호출을 위치시키는 미리 정의된 캔버스를 여러분에게 제공할 것이라는데 있다.

이를 위해 우선, 뷰 클래스(또는 그것으로부터 파생된 것)를 확장extend하고 onDraw() 콜백 메쏘드를 정의하라. 이 메쏘드는 여러분의 뷰가 스스로를 그리도록 하기 위해 안드로이드 프레임워크에 의해 호출될 것이다. 이곳은 캔버스를 통해 그리기 위해 여러분이 모든 호출을 수행할 곳이며, 캔버스는 onDraw() 콜백을 통해 여러분에게 파라미터로 전달된다.

안드로이드 프레임워크는 필요할 때만 onDraw()를 호출할 것이다. 여러분의 애플리케이션이 그려질 준비가 될 때마다, 여러분은 invalidate()를 호출함으로써 여러분의 뷰가 무효화되도록 요청해야 한다. 이것은 여러분이 뷰가 그려지길 원한다는 것을 나타내며, 그러면 안드로이드는 여러분의 onDraw() 메쏘드를 호출할 것이다(콜백이 즉시 될 것이라고 보장되지는 않더라도).

여러분의 뷰 컴포넌트의 onDraw()안에서, 여러분의 모든 드로잉을 위해 여러분에게 제공되는 캔버스를 사용하라. 여러분은 다양한 Canvas.draw...()메쏘드들 또는 아규먼트로 여러분의 캔버스를 가지는 다른 클래스의 draw() 메쏘드들을 사용한다. 일단 여러분의 onDraw()가 완료되면, 안드로이드 프레임워크는 시스템에 의해 취급되는 비트맵을 그리기 위해 여러분의 캔버스를 사용할 것이다.

여러분이 메인 액티비티의 쓰레드가 아닌 다른 쓰레드로부터 무효화 invalidate를 요청하기 위해서는 postInvalidate()를 호출해야 한다.

또한 뷰 클래스를 확장하는 것에 대한 가이드에 대해서는 "커스텀 컴포넌트 만들기"를 읽어라. 그리고 여러분의 리소스와 다른 기본 형상primitive shapes에 이미지와 같은 드로어블Drawable 오브젝트를 사용하는 정보에 대해서는 "2차원 그래픽"의 "드로어블Drawables"을 읽어라.

샘플 애플리케이션에 대해서는, SDK 샘플 폴더(<your-sdk-directory>/samples/Snake/)에 있는 스네이크 게임을 보라.

서피스뷰에서(On a SurfaceView)

서피스뷰SurfaceView는 뷰 계층구조 내에서 드로잉 전용 공간surface를 제공하는 뷰에 대한 특수한 서브클래스이다. 그 목적은 시스템의 뷰 계층구조가 그릴 준비가 될 때까지 애플리케이션이 기다리지 않도록 하기 위해, 애플리케이션의 보조secondary 쓰레드에게 이러한 드로잉 공간을 제공하는데 있다. 서피스뷰SurfaceView에 레퍼런스를 가지고 있는 보조 쓰레드는 자신의 속도로 자신의 캔버스에 드로잉을 할 수 있다.

이를 위해 우선, 여러분은 서피스뷰SurfaceView를 확장하는 새로운 클래스를 생성하는 것이 필요하다. 이 클래스는 또한 SurfaceHolder.Callback을 구현해야만 한다. 이 서브클래스는 기반이 되는 서피스surface가 생성되고, 변경되고, 소멸될 때 기반이 되는 서피스surface에 대한 정보를 여러분에게 알려주는 인터페이스이다. 이러한 이벤트들은 여러분이 언제 드로잉을 시작할 것인지, 여러분이 새로운 서피스 속성들에 기반한 조정을 해야할 필요가 있는지 여부, 그리고 여러분이 언제 드로잉을 중단할 지와 잠재적으로 태스크를 강제 종료시킬 지를 여러분이 알 수 있도록 한다는 점에서 중요하다. 여러분의 서피스뷰SurfaceView 클래스 내부는 또한 여러분이 보조 쓰레드 클래스를 정의하기 좋은 장소이다. 그것은 여러분의 캔버스에 모든 드로잉 절차를 수행할 것이다.

서피스Surface 오브젝트를 직접적으로 처리하는 대신에, 여러분은 SurfaceHolder를 통해 그것을 처리해야만 한다. 그러므로 여러분의 서피스뷰SurfaceView가 초기화 될 때, getHolder()를 호출해서 SurfaceHolder를 얻어라. 그리고 나서 여러분은 addCallback()(this를 파라미터로 전달하라)을 호출해서(SurfaceHolder.Callback으로부터) SurfaceHolder 콜백을 수신하고자 한다는 것을

SurfaceHolder에 알려줘야 한다. 그리고 여러분의 서피스뷰SurfaceView 클래스 내의 각각의 SurfaceHolder.Callback 메쏘드들을 오버라이드하라.

여러분의 보조 쓰레드 내에서 서피스Surface 캔버스에 그리기 위해, 그 쓰레드에 SurfaceHandler를 전달하고 lockCanvas()를 사용해서 캔버스를 가져와야 한다. 이제 여러분은 SurfaceHolder에 의해 제공된 캔버스를 보유할 수 있으며, 그 위에 여러분에게 필요한 드로잉을 할 수 있다. 일단 여러분이 캔버스를 사용해서 드로잉하는 것을 마쳤다면, unlockCanvasAndPost()를 호출하고 그것에 여러분의 캔버스 오브젝트를 파라미터로 전달하라. 서피스Surface는 이제 여러분이 캔버스에 남겨둔 대로 그 캔버스를 그릴 것이다. 여러분이 다시 그리고 싶을 때마다 캔버스를 lock하고 unlock하는 이 절차를 수행하라.

> **주의** 여러분이 SurfaceHolder로부터 캔버스를 가져오는 각 패스pass에서, 캔버스의 이전 상태는 계속 유지된다. 여러분의 그래픽을 적절하게 애니메이션하기 위해, 여러분은 전체 서피스surface를 다시 그려야 한다. 예를 들어 여러분은 drawColor()를 사용해서 하나의 색으로 채우거나 또는 drawBitmap()을 사용해서 배경 이미지를 설정함으로써, 캔버스의 이전 상태를 깨끗하게 지울 수 있다. 그렇지 않은 경우라면, 여러분은 이전에 수행했던 드로잉 흔적을 볼 것이다.

샘플 애플리케이션에 대해서는, SDK samples 폴더 〈your-sdk-directory〉/samples/LunarLander/ 에 있는 Lunar Landar 게임을 보거나 또는 Sample Code 섹션에 있는 소스를 찾아보라.

2차원 그래픽

안드로이드는 쉐이프shape와 이미지를 드로잉하고 애니메이션시키는 커스텀 2D 그래픽 라이브러리를 제공한다. Android.graphics.drawable과 android.view.animation 패키지에서, 여러분은 2차원 그림을 드로잉하고 애니메이션하는데 사용되는 일반적인 클래스를 발견할 수 있다.

이 문서는 안드로이드 애플리케이션에서 그래픽을 드로잉하는 것을 소개한다. 우리는 그래픽을 드로잉하기 위해 드로어블Drawable 오브젝트를 사용하는 것에 대한 기본basic과 드로어블Drawable 클래스의 두개의 서브클래스를 사용하는 방법, 그

리고 하나의 그래픽을 트윈tween(이동, 확대, 회전)하거나 일련의 그래픽(필름의 롤처럼)에 움직임을 주는 애니메이션을 생성하는 방법에 대해서 논의할 것이다.

드로어블(Drawables)

드로어블Drawable은 "그려질 수 있는 어떤 것"에 대한 일반적인 추상화abstraction이다. 여러분은 BitmapDrawable, ShapeDrawable, PictureDrawable, LayerDrawable, 그리고 더 다양한 것들을 포함하여, 특정 종류의 드로어블drawable 그래픽을 정의하기 위해 드로어블Drawable 클래스가 확장된다는 것을 발견할 것이다. 물론 여러분은 고유한 방식으로 동작하는 여러분 자신의 커스텀 드로어블Drawable 오브젝트를 정의하기 위해 이러한 드로어블들을 확장할 수 있다.

드로어블Drawable을 정의하고 인스턴스화하는 데에는 세 가지 방법이 있다. 여러분의 프로젝트 리소스에 저장된 이미지를 사용하는 방법, 드로어블Drawable 속성properties을 정의하고 있는 XML 파일 사용는 방법, 또는 일반적인 클래스 생성자를 사용하는 방법이 있다. 아래에서 처음 두 가지 방법 각각에 대해 논의할 것이다(숙련된 개발자에게 있어서 생성자를 사용하는 것은 새로울 것이 없다).

리소스 이미지에서 생성하기

여러분의 애플리케이션에 그래픽을 추가하는 간단한 방법은 프로젝트 리소스에 있는 이미지 파일을 레퍼런스하는 것이다. 지원되는 파일 타입들은 PNG(선호되는), JPG(수용가능한), 그리고 GIF(좋지않은)이다. 이 기법technique은 애플리케이션 아이콘, 로고, 또는 게임에서 사용되는 것과 같은 다른 그래픽들에서 매우 선호된다.

이미지 리소스를 사용하려면 여러분의 파일을 프로젝트의 res/drawable/ 디렉토리에 추가하기만 하면 된다. 그곳으로부터, 여러분은 코드 또는 XML 레이아웃에서 그것을 레퍼런스할 수 있다. 어떤 방법이든, 그것은 파일 타입 확장자를 제외한 파일 이름으로 된 리소스 ID를 사용하여 레퍼런스된다(예를 들어 my_image.png는 my_image로 레퍼런스 된다).

NOTE

res/drawable/ 에 위치하는 이미지 리소스들은 자동으로 aapt 도구에 의해 무손실 이미지로 압축되어 최적화 된다. 예를 들어 256 컬러 이상을 요구하지 않는 트루컬러 PNG는 컬러 팔레트를 가지는 8-비트 PNG로 변환될 수도 있다. 이것은 동일한 품질의 이미지를 결과로 갖지만, 더 작은 메모리를 요구한다. 그러므로 이 디렉토리에 위치하는 이미지 바이너리들은 빌드하는 동안 변경될 수 있다는 것을 알아라. 만약 여러분이 이미지를 비트맵으로 변환하기 위해 비트 스트림으로 읽을 예정이라면, res/drawable 대신에 여러분의 이미지를 res/raw/ 폴더 안에 놓아라. 그 폴더 안의 이미지들은 최적화되지 않을 것이다.

예제 코드(Example code)

아래의 코드 단편snippet은 드로어블drawable 리소스로부터 이미지를 사용하여 이미지뷰를 만들고, 그것을 레이아웃에 추가하는 방법을 보여준다.

```
LinearLayout mLinearLayout;
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Create a LinearLayout in which to add the ImageView
    mLinearLayout = new LinearLayout(this);
    // Instantiate an ImageView and define its properties
    ImageView i = new ImageView(this);
    i.setImageResource(R.drawable.my_image);
    i.setAdjustViewBounds(true); // set the ImageView bounds to match the Drawable's dimensions
     i.setLayoutParams(new Gallery.LayoutParams(LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CON-
TENT));

    // Add the ImageView to the layout and set the layout as the content view
    mLinearLayout.addView(i);
    setContentView(mLinearLayout);
}
```

다른 경우로, 여러분은 이미지 리소스를 드로어블Drawable 오브젝트로 취급하길 원할 수 있다. 그렇게 하기 위해서는, 다음과 같이 리소스로부터 드로어블Drawable을 생성하라.

```
Resources res = mContext.getResources();
Drawable myImage = Resources.getDrawable(R.drawable.my_image);
```

NOTE

여러분의 프로젝트에 있는 각각의 고유한 리소스는 여러분이 그것에 대해 얼마나 많은 오브젝트를 인스턴스화 할 지 상관없이, 오직 하나의 상태만을 유지할 수 있다. 예를 들어 여러분이 동일한 이미지 리소스로부터 두 개의 드로어블Drawable 오브젝트를 인스턴스화하고, 그런 다음에 그 드로어블Drawable들 중 하나에 대해 (알파와 같은) 속성을 변경한다면, 그것은 또한 다른 것에도 영향을 미칠 것이다. 그러므로 하나의 리소스로부터 복수개의 인스턴스를 취급할 때, 여러분은 드로어블Drawable을 직접 변형시키는 것 대신에 트윈 애니메이션을 수행해야 한다.

예제 XML(Example XML)

아래의 XML 단편은 리소스 드로어블Drawable을 XML 레이아웃에 있는 ImageView에 추가하는 방법을 보여준다(단지 재미를 위해 약간의 빨간 tint를 가짐).

```
<ImageView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:tint="#55ff0000"
  android:src="@drawable/my_image"/>
```

프로젝트 리소스들을 사용하는 것에 대한 더 많은 정보는 4장. "리소스와 에셋"을 보라.

리소스 XML에서 생성하기

여러분은 지금쯤 이미 사용자 인터페이스를 개발하는 안드로이드의 원리에 대해 친숙해 있어야 한다. 따라서, 여러분은 XML로 오브젝트를 정의하는 것에 대한 그 고유한 힘과 유연성을 이해하고 있다. 이 철학은 뷰View에서부터 드로어블Drawable에까지 걸쳐서 전해진다. 만약 그곳에 여러분이 생성하고자 하는 드로어블 오브젝트가 있고, 그것이 여러분의 애플리케이션 코드 또는 사용자 상호작용에 의해 정의되는 변수에 초기에 의존적이지 않다면, XML로 드로어블을 정의하는 것은 좋은 선택이다. 비록, 여러분의 애플리케이션을 사용자가 경험하는 동안에, 여러분의 드로어블이 그것의 속성을 변경해야 할 필요가 있을 지라도 여러분은 XML로 오브젝트를 정의하는 것을 고려해야 한다. 왜냐하면, 일단 오브젝트가 한번 인스턴스화되면 여러분이 속성들을 항상 수정할 수 있기 때문이다.

일단 여러분이 XML로 드로어블을 정의했다면 그 파일을 프로젝트의 res/drawable/ 디렉토리에 저장하라. 그런 다음에 Resources.getDrawables()을 호출하고, 그것에 여러분의 XML 파일에 대한 리소스 ID를 전달함으로써 오브젝트를 가져와서 인스턴스화하라(아래의 예제를 보라).

Inflate() 메쏘드를 지원하는 어떠한 드로어블 서브클래스라도 XML로 정의되고 여러분의 애플리케이션에 의해 인스턴스화될 수 있다. XML 인플레이션inflation을 지원하는 각각의 드로어블은 오브젝트 속성 정의를 돕는 특정 XML 애트리뷰트들을 이용한다(이러한 것들에 대한 것을 보려면 해당 클래스 레퍼런스를 참조하라). XML로 그것을 정의하는 방법에 대한 정보에 대해서는 각각의 드로어블 서브클래스의 클래스 문서를 보라.

예제(Example)

여기에는 TrasitionDrawable을 정의하는 약간의 XML이 있다.

```
<transition xmlns:android="http://schemas.android.com/apk/res/an-
droid">
  <item android:drawable="@drawable/image_expand">
  <item android:drawable="@drawable/image_collapse">
</transition>
```

res/drawable/expand_collapse.xml 파일로 저장된 이 XML을 사용해서 아래의 코드는 TrasitionDrawable을 인스턴스화 할 것이며, 그것을 이미지뷰의 컨텐트로 지정한다.

```
Resources res = mContext.getResources();
TransitionDrawable transition = (TransitionDrawable) res.getDrawable(R.
drawable.expand_collapse);
ImageView image = (ImageView) findViewById(R.id.toggle_image);
image.setImageDrawable(transition);
```

그런 다음에 이 변화trasition가 (1초 동안) 아래와 같이 앞으로 실행될 수 있다.

```
transition.startTransition(1000);
```

각각에 의해 지원하는 XML 애트리뷰트에 대한 더 많은 정보에 대해서는 위에 나열된 드로어블Drawable 클래스들을 참조하라.

쉐이프 드로어블(ShapeDrawable)

여러분이 동적으로 어떤 2차원 그래픽을 그리기를 원할 때, 쉐이프 드로어블ShapeDrawable 오브젝트는 여러분의 필요에 아마도 잘 맞을 것이다. 쉐이프 드로어블을 사용해서 여러분은 상상할 수 있는 어떠한 방법으로든 프로그램적으로 기본 쉐이프primitive shapes를 그리고, 그것들을 스타일링할 수 있다.

쉐이프 드로어블ShapeDrawable은 드로어블의 확장이다. 그러므로 여러분은 드로어블이 있을 수 있는 어디에서든 그것을 사용할 수 있다 – 아마도 뷰의 백그라운드를 위해서는, setBackgroundDrawable()를 사용해서 설정할 수 있다. 물론 여러분은 또한 여러분이 원하는대로 여러분의 레이아웃에 추가되도록 하기 위해 자신만의 커스텀 뷰로써 쉐이프shape를 그릴 수 있다. 쉐이프 드로어블은 자신만의 draw() 메쏘드를 가지고 있기 때문에, 여러분은 View.onDraw() 메쏘드 도중에 쉐이프 드로어블을 그리는 뷰의 서브클래스를 생성할 수 있다. 여기에 하나의 뷰로써 쉐이프 드로어블을 그리기 위해, 단지 아래와 같은 것을 하는 뷰 클래스의 기본적 확장이 있다.

```
public class CustomDrawableView extends View {
    private ShapeDrawable mDrawable;

    public CustomDrawableView(Context context) {
        super(context);

        int x = 10;
        int y = 10;
        int width = 300;
        int height = 50;

        mDrawable = new ShapeDrawable(new OvalShape());
        mDrawable.getPaint().setColor(0xff74AC23);
        mDrawable.setBounds(x, y, x + width, y + height);
    }

    protected void onDraw(Canvas canvas) {
```

```
            mDrawable.draw(canvas);
        }
    }
```

생성자에서 쉐이프 드로어블은 타원모양OvalShape으로 정의된다. 그리고 컬러가 주어지고 쉐이프shape의 범위가 설정된다. 여러분이 범위를 설정하지 않으면 그 쉐이프는 그려지지 않을 것이다. 그러나 컬러를 설정하지 않으면 디폴트로 검정색이 지정된다.

정의된 그 커스텀 뷰를 사용해서, 쉐이프는 여러분이 원하는 어떤 방법으로도 그려질 수 있다. 위의 샘플을 사용해서, 우리는 액티비티에서 프로그램적으로 쉐이프shape를 그릴 수 있다.

```
CustomDrawableView mCustomDrawableView;

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mCustomDrawableView = new CustomDrawableView(this);

    setContentView(mCustomDrawableView);
}
```

만약 여러분이 액티비티에서가 아니라 XML 레이아웃으로부터 이 커스텀 드로어블을 그리고자 한다면, View(Context, AttributeSet) 생성자를 오버라이드해야 하며, View 생성자는 XML로부터 인플레이션inflation을 통해 하나의 뷰를 인스턴스화할 때 호출된다. 그 다음에 CustomDrawable 엘리먼트를 다음과 같이 XML에 추가하라.

```
<com.example.shapedrawable.CustomDrawableView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
```

(android.graphics.drawable 패키지에 있는 다른 많은 Drawable 타입들과 같은) 쉐이프 드로어블 ShapeDrawable 클래스는 여러분으로 하여금 public 메쏘드들을 사용해서 드로어블의 다양한 속성들을 정의하는 것을 허용한다. 여러분이 조정하고자 하는 몇몇

의 속성값으로는 알파 투명도alpha transparency, 컬러 필터color filter, 떨림dither, 불투명opacity, 그리고 컬러color가 있다.

나인패치 드로어블(NinePatchDrawable)

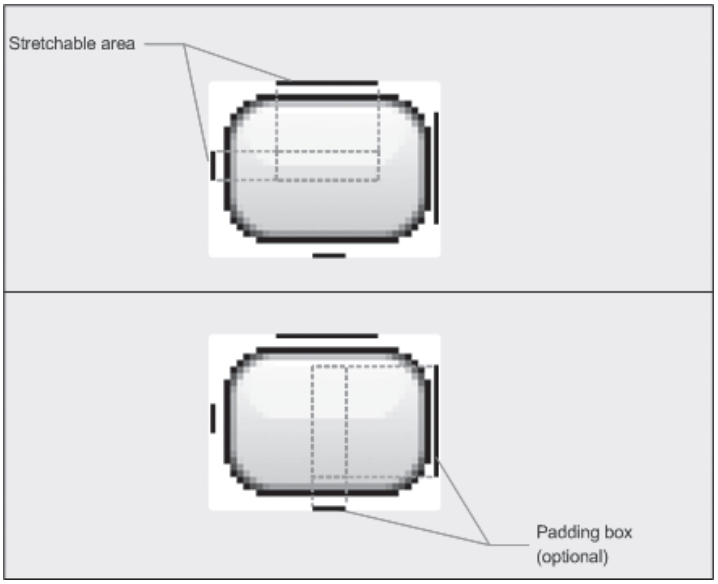
나인패치 드로어블NinePatchDrawble 그래픽은 확대 가능한 비트맵 이미지로, 안드로이드는 여러분이 그것을 배경으로 놓은 뷰의 컨텐츠에 적합하도록 자동으로 크기를 조정할 것이다. 나인패치NinePatch가 사용된 예제는 표준 안드로이드 버튼에서 사용되는 백그라운드들이다. 버튼은 문자열의 가변적 길이에 적합하도록 확대되어야 한다. 나인패치NinePatch 드로어블은 추가적인 1 픽셀 두께의 선border을 포함하는 표준 PNG 이미지이다. 이것은 .9.png 확장자로 저장되어야 하고 프로젝트의 res/drawable 디렉토리에 저장되어야 한다.

선border은 이미지의 확대 가능 영역 및 고정 영역을 정의하기 위해 사용된다. 여러분은 왼쪽과 상단 영역의 선border에 하나 (또는 그 이상의) 1픽셀 두께의 검정색 라인(들)을 그려서 확대 가능한 섹션을 가리킨다(여러분이 원하는 만큼의 많은 확대가능한 섹션을 가질 수 있다). 확대 가능한 섹션들의 상대적 크기는 동일하게 유지된다. 그러므로 가장 큰 섹션들은 항상 가장 크게 남게 된다.

여러분은 또한 우측과 하단 라인에 선을 그려서 선택사항인 이미지의 드로어블 섹션(실제로, 패딩 라인)을 정의할 수도 있다. 만약 뷰 오브젝트가 그것의 백그라운드로 나인패치NinePatch를 설정하고, 그런 다음에 뷰의 텍스트를 지정한다면, 그것은 모든 텍스트가 우측과 하단 라인(만약 있다면)에 의해 지정된 영역 내에 맞도록 자기 자신을 확대할 것이다. 만약 패딩 라인이 포함되지 않았다면, 안드로이드는 드로어블 영역을 정의하기 위해 왼쪽과 상단 라인을 사용한다.

다른 선들 간의 차이를 명확히 하기 위해서, 왼쪽과 상단 선은 이미지를 확대하기 위해 이미지의 어떤 픽셀들이 복제가 허용되는지를 정의한다. 하단과 우측 선은 뷰의 컨텐츠가 그 내부에 놓이도록 허용되는 이미지 내부의 상대적 영역을 정의한다.

여기에 버튼을 정의하기 위해 사용된 나인패치NinePatch 샘플이 있다.



나인패치NinePatch는 좌측과 상단 선으로 확대 가능한 영역을 그리고, 하단과 우측 선으로 드로어블 영역을 정의한다. 위의 이미지에서 점으로 표현된 회색 선들은 이미지를 확대하기 위해 복제될 이미지의 영역을 나타내고 있다. 아래쪽 이미지의 핑크색 직사각형은 뷰의 컨텐츠가 허용된 영역을 나타낸다. 만약 컨텐츠가 이 영역에 맞지 않는다면, 이미지는 맞게 되도록 확대될 것이다.

draw9patch 툴tool은 WYSIWYG 그래픽 편집기를 사용해서 여러분의 나인패치NinePatch 이미지들을 생성하는 매우 편리한 방법을 제공한다. 이 도구는 심지어 만약 여러분이 확대 가능한 영역으로 정의한 구역이 픽셀 복사 결과로써 인위적 그림을 만들어내는 위험이 있다면 경고를 나타낸다.

예제 XML(Example XML)

여기에 나인패치NinePatch 이미지를 두개의 버튼에 추가하는 방법을 보여주는 샘플 레이아웃 XML이 있다(나인패치(NinePatch) 이미지는 res/drawable/my_button_background.9.png로 저장되어 있다).

```
<Button id="@+id/tiny"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerInParent="true"
        android:text="Tiny"
        android:textSize="8sp"
        android:background="@drawable/my_button_background"/>

<Button id="@+id/big"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerInParent="true"
        android:text="Biiiiiiig text!"
        android:textSize="30sp"
        android:background="@drawable/my_button_background"/>
```

버튼이 텍스트 둘레에 깨끗하게 맞도록 하기 위해 버튼의 너비와 높이는 "wrap_content"로 설정되어 있다는 점에 유의하라.

아래는 위에서 보여진 XML과 나인패치NinePatch 이미지로부터 그려진 두 개의 버튼이다. 버튼의 너비와 높이가 텍스트와 함께 어떻게 바뀌었고, 배경 이미지가 그것에 맞춰지도록 어떻게 확대되었는지를 주목하라.

트윈 애니메이션(Tween Animation)

트윈Tween 애니메이션은 뷰 오브젝트의 컨텐츠에 대한 일련의 간단한 변환(위치, 크기, 회전, 그리고 투명도)을 수행할 수 있다. 그러므로 만약 여러분이 텍스트뷰TextView 오브젝트를 가지고 있다면, 여러분은 텍스트를 이동하고, 회전하고, 키우거나 줄일 수 있다. 만약 백그라운드 이미지가 있다면, 백그라운드 이미지는 텍스트에 따라 변환될 것이다. animation 패키지는 트윈 애니메이션에서 사용되는 모든 클래스를 제공한다.

일련의 애니메이션 명령어들은 XML이나 안드로이드 코드에 의해 정의된 트윈 애니메이션을 정의한다. 레이아웃을 정의하는 것과 같이, XML 파일이 애니메이션을 하드 코딩하는 것보다 더 읽기 쉽고, 재사용 가능하고, 그리고 교체 가능하기 때문에 XML 파일이 권장된다. 아래의 예제에서 우리는 XML을 사용한다(XML 대신에 애플리케이션 코드에서 애니메이션을 정의하는 것에 대해 더 알고자 하면 AnimationSet 클래스와 다른 Animation 서브클래스를 참조하라).

애니메이션 명령어들은 여러분이 발생하길 원하는 변환이 언제 일어날 지, 얼마나 오랫동안 적용되어야 하는지를 정의한다. 변환은 순차적 또는 동시에 일어날 수 있다. 예를 들어 여러분은 텍스트뷰 컨텐츠를 왼쪽에서 오른쪽으로 이동한 다음에 180도 회전할 수 있으며, 또한 여러분은 텍스트의 이동과 회전을 동시에 가질 수 있다. 각각의 변환은 그 변환에 대한 특정 파라미터(크기 변경을 위한 시작 크기와 끝 크기, 회전을 위한 시작 각도와 끝 각도, 그리고 기타 등등)의 집합을 가진다. 그리고 또한 일반적인 파라미터의 집합(예를 들어, 시작 시간과 기간)이 있다. 여러 개의 변환이 동시에 일어나도록 하려면, 그것들에게 동일한 시작 시간을 부여하라. 순차적으로 일어나게 하려면 시작 시간에 선행하는 변환의 기간을 더해서 계산하라.

애니메이션 XML 파일은 안드로이드 프로젝트의 res/anim/ 디렉토리에 위치한다. 그 파일은 오직 하나의 루트 엘리먼트를 가져야 한다. 이것은 하나의 〈alpha〉, 〈scale〉, 〈translate〉, 〈rotate〉, interpolator 엘리먼트, 또는 이러한 엘리먼트들의 그룹들을 가지는 〈set〉 엘리먼트(이것은 또 다른 〈set〉을 포함할 지도 모른다) 중 하나가 될 것이다. 디폴트로 모든 애니메이션 명령어는 동시에 적용된다. 순차적으로 일어나도록 하려면 아래 예제에서 보여지는 것처럼 startOffset 애트리뷰트를 지정해야만 한다.

ApiDemos 중 하나에 있는 다음의 XML은 확대하기 위해 사용되며, 그런 다음에 뷰 오브젝트를 동시에 돌리고 회전시킨다.

```
<set android:shareInterpolator="false">
   <scale
       android:interpolator
               ="@android:anim/accelerate_decelerate_interpolator"
          android:fromXScale="1.0"
          android:toXScale="1.4"
          android:fromYScale="1.0"
```

```
            android:toYScale="0.6"
            android:pivotX="50%"
            android:pivotY="50%"
            android:fillAfter="false"
            android:duration="700" />
      <set android:interpolator="@android:anim/decelerate_interpolator">
         <scale
                android:fromXScale="1.4"
                android:toXScale="0.0"
                android:fromYScale="0.6"
                android:toYScale="0.0"
                android:pivotX="50%"
                android:pivotY="50%"
                android:startOffset="700"
                android:duration="400"
                android:fillBefore="false" />
         <rotate
                android:fromDegrees="0"
                android:toDegrees="-45"
                android:toYScale="0.0"
                android:pivotX="50%"
                android:pivotY="50%"
                android:startOffset="700"
                android:duration="400" />
      </set>
   </set>
```

(이 예제에서 사용되지 않은) 스크린 좌표는 왼쪽 상단 쪽 구석에 있는 (0,0)이고, 여러분이 아래와 오른쪽으로 가는 만큼 증가한다.

pivotX와 같은 몇 개의 값들은 오브젝트 그 자신에 상대적으로 또는 부모parent에 상대적으로 지정될 수 있다. 여러분이 원하는 것에 대한 적절한 포맷을 사용하고 있는 지를 확인하라(부모(parent)에 상대적인 50%는 "50"이고 오브젝트 자기 자신에 상대적인 50%는 "50%"이다).

여러분은 인터폴레이터Interpolator를 지정해서 시간의 경과에 따라 변환이 적용되는 방법을 결정할 수 있다. 안드로이드는 다양한 속도 곡선을 지정하는 여러 개의 인터폴레이터Interpolator 서브클래스를 포함한다. 예를 들어 AccelerateInterpolator는 천천히 시작해서 속도를 높이는 변환을 가리킨다. 각각은 XML에서 적용될 수 있는 하나의 애트리뷰트 값을 가진다.

프로젝트의 res/anim/ 디렉토리에 hiperspace_jump.xml로 저장된 이 XML을 사용해서, 다음의 Java 코드는 그것을 레퍼런스할 것이며, 그리고 레이아웃의 ImageView 오브젝트에 적용할 것이다.

```
ImageView spaceshipImage
      = (ImageView) findViewById(R.id.spaceshipImage);
Animation hyperspaceJumpAnimation
      = AnimationUtils.loadAnimation(this, R.anim.hyperspace_jump);
spaceshipImage.startAnimation(hyperspaceJumpAnimation);
```

startAnimation()에 대한 대체물로써, 여러분은 Animation.setStartTime()을 사용해서 애니메이션 시작 시각을 정의할 수 있으며, 그런 다음에 View.setAnimation()으로 View에 애니메이션을 지정할 수 있다.

사용가능한 태크와 애트리뷰트에 대한 XML 구문에 대한 더 많은 정보에 대해서는 4장. "리소스와 에셋"의 "사용가능한 리소스 타입"에 있는 애니메이션에 대한 논의를 참고하라.

> **NOTE**
>
> 여러분의 애니메이션이 얼마나 움직이고 크기가 바뀌었는지와 무관하게, 여러분의 애니메이션을 보유하는 뷰의 경계는 자동으로 그것을 맞추도록 조절되지 않을 것이다. 심지어 그 애니메이션은 여전히 뷰의 경계를 넘어서서 그려질 것이고 잘려지지도 않을 것이다. 하지만 만약 애니메이션이 부모parent 뷰의 경계를 넘어서면 클리핑이 일어날 것이다.

프레임 애니메이션(Frame Animation)

이것은 필름의 롤처럼 순서대로 재생되는 일련의 서로 다른 이미지들로 만들어졌다는 점에서 전통적인 애니메이션이다. 애니메이션 드로어블AnimationDrawable 클래스는 프레임 애니메이션의 기본이다.

애니메이션 드로어블AnimationDrawable 클래스 API를 사용해서 여러분이 코드에서 애니메이션의 프레임을 정의할 수 있을지라도, 애니메이션을 구성하는 프레임을 나열하는 XML 파일을 사용해서 더 쉽게 이루어진다. 위에서의 트윈 애니메이션처럼, 이러한 종류의 애니메이션에 대한 XML 파일은 안드로이드 프로젝트의

res/anim/ 디렉토리에 위치한다. 이 경우, 명령어들은 애니메이션 각 프레임에 대한 순서와 기간이다.

XML 파일은 루트 노드로써 〈animation-list〉 엘리먼트와 각각이 프레임(프레임과 프레임 기간에 대한 드로어블 리소스)을 정의하는 일련의 자식child 〈item〉 노드로 구성된다. 여기에 frame-by-frame 애니메이션에 대한 XML 파일 예제가 있다.

```
<animation-list
       xmlns:android="http://schemas.android.com/apk/res/android"
       android:oneshot="true">
  <item android:drawable="@drawable/rocket_thrust1"
         android:duration="200" />
  <item android:drawable="@drawable/rocket_thrust2"
         android:duration="200" />
  <item android:drawable="@drawable/rocket_thrust3"
         android:duration="200" />
</animation-list>
```

이 애니메이션은 단 세 개의 프레임으로 실행된다. 리스트의 android:oneshot 애트리뷰트를 참true으로 설정하면, 그것은 단지 한 번만 순환하고 정지해서 마지막 프레임에 멈춰있다. 만약 그것이 거짓false으로 설정된다면, 애니메이션은 반복될 것이다. 프로젝트의 res/anim/ 디렉토리에 rocket_thrust.xml로 저장된 이 XML을 가지고, 그것은 뷰의 백그라운드 이미지에 추가될 수 있으며 그리고 실행되도록 호출될 수 있다. 여기에 애니메이션이 이미지뷰에 추가되어서 스크린이 터치될 때 움직이는 액티비티 예제가 있다.

```
AnimationDrawable rocketAnimation;

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

  ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
  rocketImage.setBackgroundResource(R.anim.rocket_thrust);
  rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
}

public boolean onTouchEvent(MotionEvent event) {
```

```
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
      rocketAnimation.start();
      return true;
    }
    return super.onTouchEvent(event);
  }
```

애니메이션 드로어블AnimationDrawable에 대해서 호출되는 start() 메쏘드는 여러분의 액티비티의 onCreate() 메쏘드안에서 호출될 수 없다는 것을 알고 있는 것이 중요하다. 왜냐하면, 애니메이션 드로어블AnimationDrawable이 아직 완전하게 윈도우와 결합되지 않았기 때문이다. 만약 여러분이 상호작용을 요구하지 않고 애니메이션을 즉시 재생하고자 한다면, 여러분 액티비티의 onWindowFocusChanged() 메쏘드에서 그것을 호출할 수도 있다. onWindowFocusChanged() 메쏘드는 안드로이드가 여러분의 윈도우에 포커스를 줄 때 호출될 것이다.

OpenGL을 사용하는 3차원 그래픽

안드로이드는 OpenGL API, 보다 명확하게 OpenGL ES API를 통해 고성능의 3차원 그래픽에 대한 지원을 포함하고 있다.

OpenGL ES는 임베디드 디바이스를 고려한 OpenGL 규격specification의 변형이다. OpenGL ES 버전은 기본적인 OpenGL 표준 버전에 정확하게 대응하지 않는다. 안드로이드는 현재 OpenGL 1.3에 대응되는 OpenGL ES 1.0을 지원한다. 그러므로 만약 여러분이 생각하고 있는 애플리케이션이 데스크탑 시스템 상의 OpenGL 1.3을 사용하는 것이 가능하다면, 그것이 안드로이드에서도 가능해야 한다.

안드로이드에 의해 제공되는 특정 API는 J2ME JSR239 OpenGL ES API와 유사하다. 하지만 그것이 동일하지는 않을 지도 모른다. 그러므로 차이들에 주의하라.

API 사용하기(Using the API)

여기에 매우 높은 수준에서 바라본 API 사용법이 있다.

1. 커스텀 뷰 서브클래스를 작성하라.
2. OpenGL 기능에 대한 접근을 제공하는 OpenGLContext에 대한 핸들handle을 얻어라.
3. 여러분 뷰의 onDraw() 메쏘드에서, GL 오브젝트에 대한 핸들handle을 얻어라. 그리고 GL 오퍼레이션을 수행하기 위해 그 GL 오브젝트의 메쏘드를 사용하라.

(전통적인 GL ColorCube에 기반한) 이런 사용 모델에 대한 예제에 대해서는, Api Demos 샘플 코드 프로젝트에 있는 com.android.samples.graphics.GLSurfaceView.java를 참조하라. 쓰레드를 사용해서 이것을 사용하는 방법을 보여주는 다소 더 세련된 버전을 com.android.samples.graphics.GLSurfaceViewActivity.java에서 찾을 수 있다.

OpenGL을 사용해서 실제로 3차원 애플리케이션을 작성하는 방법에 대한 개요를 작성하는 것은 이 텍스트의 범위를 넘어서는 것이다. 그래서 그것은 독자의 과제로 남겨 놓는다.

추가 정보에 대한 링크

OpenGL ES에 대한 정보는 http://www.khronos.org/opengles/ 에서 찾을 수 있다.

(자세한 구격을 포함해서) OpenGL ES 1.0에 대한 정보는 http://www.khronos.org/opengles/1_X/ 에서 찾을 수 있다.

안드로이드 OpenGL ES 구현에 대한 문서 또한 아래의 링크에서 입수할 수 있다.

```
http://developer.android.com/reference/javax/microedition/khronos/
opengles/package-summary.html
```

마지막으로 안드로이드가 OpenGL ES 1.1에 대한 몇 가지 기본적 지원을 포함하고 있음에도 불구하고 그 지원은 완전하지 않으며, 이 시점에서는 신뢰해서는 안된다.

Section | 11

오디오 및 비디오

11.1. 오디오 및 비디오 재생

11.2. 오디오 캡쳐

11.3. 안드로이드 지원 미디어 포맷

11. 오디오 및 비디오

안드로이드 플랫폼은 다양한 일반적인 미디어 타입에 대한 내장된built-in 인코딩/디코딩을 제공하며, 이로써 여러분은 쉽게 오디오, 비디오, 이미지를 여러분의 애플리케이션에 통합할 수 있다. 플랫폼의 미디어 기능에 액세스하는 것은 꽤 수월하다 – 여러분은 인텐트와 액티비티 메커니즘을 사용해서 그렇게 할 수 있으며, 이 메커니즘은 안드로이드의 나머지 부분들에서도 사용된다.

안드로이드는 여러분으로 하여금 여러 유형의 데이터 소스로부터 오디오와 비디오를 재생할 수 있도록 한다. 여러분은 애플리케이션 리소스들(raw resources)에 저장된 미디어 파일들, 파일 시스템에 있는 독립된 파일들, 혹은 네트워크 커넥션을 통해 도착하는 데이터 스트림으로부터 오디오와 비디오를 재생할 수 있다. 애플리케이션에서 오디오와 비디오를 재생하려면 미디어플레이어MediaPlayer 클래스를 사용하라.

플랫폼은 여러분으로 하여금 오디오와 비디오를 녹음(녹화)할 수 있도록 하며, 이것은 모바일 디바이스 하드웨어에 의해 지원된다. 오디오와 비디오를 녹음(녹화)하려면 미디어레코더MediaRecorder 클래스를 사용하라. 에뮬레이터는 오디오와 비디오를 캡쳐하는 하드웨어를 가지고 있지 않지 않다는 것에 유의하라. 하지만 실제 모바일 디바이스는 미디어레코더MediaRecorder를 통해 접근할 수 있는 이러한 기능들을 여러분에게 제공할 것이다.

안드로이드가 내장된built-in 지원을 제공하는 미디어 포맷 리스트에 대해서는 이후에 나오는 안드로이드 지원 미디어 포맷을 보라.

오디오 및 비디오 재생

미디어는 원시 리소스, 시스템 내의 파일, 또는 사용 가능한 네트워크(URL) 어느 곳으로부터든 재생될 수 있다.

여러분은 표준 출력 디바이스로만 오디오 데이터를 재생할 수 있다; 현재 그 디바이스는 모바일 디바이스 스피커 또는 블루투스 헤드셋이다. 여러분은 대화형 오디오conversation audio에서 사운드 파일을 재생할 수 없다.

원시 리소스에서 재생하기

아마도 수행하길 원하는 가장 일반적으로 것은 여러분의 애플리케이션에서 미디어(특히 사운드)를 재생하는 것일 것이다. 이것을 하는 것은 쉽다.

1. 사운드(또는 다른 미디어 리소스) 파일을 여러분의 프로젝트의 res/raw 폴더에 넣어라. 이클립스 플러그인(또는 aapt)은 그곳에 있는 파일을 찾아서 그 파일이 여러분의 R 클래스에서 레퍼런스될 수 있는 리소스가 되게 한다.
2. MediaPlayer.create()를 사용해서 리소스를 레퍼런스하는 미디어플레이어 MediaPlayer 인스턴스를 생성하라. 그런 다음에 그 인스턴스에 대한 start()를 호출하라.

```
MediaPlayer mp = MediaPlayer.create(context, R.raw.sound_file_1);
mp.start();
```

재생을 중지하려면 stop()을 호출하라. 만약 여러분이 나중에 그 미디어를 다시 재생하고자 한다면, 여러분은 start()를 다시 호출하기 전에 MediaPlayer 오브젝트를 reset()하고 prepare() 해야만 한다(create()는 최초 시점에 prepare()를 호출한다).

재생을 일시 정지하기 위해서는 pause()를 호출하라. 여러분은 일시 정지한 곳에서 start()를 사용해서 재생을 재개할 수 있다.

파일 또는 스트림에서 재생하기

여러분은 파일 시스템 또는 웹 URL로부터 미디어 파일을 재생할 수 있다.

1. new를 사용하여 미디어플레이어(MediaPlayer) 인스턴스를 생성하라.
2. 재생하고자 하는 파일의 경로(로컬 파일시스템 또는 URL)을 포함하고 있는 문자열을 가지는 setDataSource()를 호출하라.
3. 먼저 prepare()를 하고, 그런 다음에 그 인스턴스에 대한 start()를 하라.

```
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(PATH_TO_FILE);
mp.prepare();
mp.start();
```

stop()과 pause()는 위에서 논의된 것과 같이 동작한다.

mp가 널null 값이 되는 것이 가능하다. 그러므로 좋은 코드는 new 이후에 널null 체크를 해야 한다. 또한 여러분이 레퍼런스하는 파일이 존재하지 않을 수도 있기 때문에, setDataSource()를 사용할 때 IllegalArgumentException과 IOException이 캐치catch되거나 전달될 필요가 있다.

여러분이 온라인 미디어 파일에 대한 URL을 전달한다면, 그 파일은 프로그레시브progressive 다운로드가 지원되어야 한다.

제트 컨텐트 재생하기

안드로이드 플랫폼은 여러분의 애플리케이션에 인터랙티브 제트JET 오디오 컨텐트 재생을 추가하게 해주는 제트JET 엔진을 포함하고 있다. 여러분은 SDK에 함께 포함되어 있는 JetCreator 제작authoring 애플리케이션을 사용해서 인터랙티브 재생을 위한 제트JET 컨텐트를 생성할 수 있다. 여러분의 애플리케이션부터 제트JET 컨텐트를 재생하고 관리하기 위해서는, JetPlayer 클래스를 사용하라.

제트JET의 개념과 JetCreator 제작authoring 도구 사용법에 대한 명령어 설명에 대해서는, JetCreator 사용자 메뉴얼을 보라. JetCreator 도구는 OS X와 윈도우 플랫폼 상에서 모든 기능을 제공하며, 리눅스 버전에서 모든 컨텐트 생성 기능들을 지원하지만 임포드import된 에셋들assets의 오디션닝은 지원하지 않는다.

여기 SD카드에 저장된 .jet 파일로부터 제트JET 재생을 설정하는 방법에 대한 예제가 있다.

```
JetPlayer myJet = JetPlayer.getJetPlayer();
myJet.loadJetFile("/sdcard/level1.jet");
byte segmentId = 0;

// queue segment 5, repeat once, use General MIDI, transpose by -1 oc-
tave
myJet.queueJetSegment(5, -1, 1, -1, 0, segmentId++);
// queue segment 2
myJet.queueJetSegment(2, -1, 0, 0, 0, segmentId++);

myJet.play();
```

SDK는 이것에 대한 예제 애플리케이션, JetBoy를 포함하고 있다. 그것은 여러분의 게임에 인터랙티브 뮤직 사운드트랙을 생성하기 위해 제트플레이어 JetPlayer를 사용하는 방법을 보여준다. 그것은 또한 뮤직과 게임 로직을 동기화하기 위해 제트JET 이벤트를 사용하는 방법을 설명한다. 애플리케이션은 〈sdk〉/platforms/android-1.5/samples/JetBoy에 위치해 있다.

오디오 캡쳐

디바이스로부터 오디오를 캡쳐하는 것은 오디오/비디오 재생보다는 좀 더 복잡하지만, 여전히 꽤 간단하다.

1. new를 사용하여 android.media.MediaRecorder의 새로운 인스턴스를 생성하라.
2. android.content.ContentValues의 새로운 인스턴스를 생성하고, TITLE, TIMESTAMP, 그리고 모든 중요한 MIME_TYPE과 같은 몇 가지 표준 속성들을 채워라.
3. 보내야 하는 데이터에 대한 파일 경로를 생성하라 (여러분은 컨텐트 데이터베이스안에 엔트리를 만들기 위해 android.content.ContentResolver를 사용할 수 있으며, 다음에 여러분이 사용할 수 있는 경로를 자동으로 지정하기 위해 그것을 얻을 수 있다).
4. MediaRecorder.setAudioSource()를 사용해서 오디오 소스를 설정하라. 여러분은 아마도 MediaRecorder.AudioSource.MIC 를 사용하고자 할 것이다.

5. MediaRecorder.setOutputFormat()를 사용해서 출력 파일 포맷을 설정하라.
6. MediaRecorder.setAudioEncoder()를 사용해서 오디오 인코더를 설정하라.
7. MediaRecorder 인스턴스에 대한 prepare()를 호출하라.
8. 오디오 캡쳐를 시작하기 위해서는 start()를 호출하라.
9. 오디오 캡쳐를 멈추기 위해서는 stop()을 호출하라.
10. 여러분이 MediaRecorder 인스턴스를 사용했다면, 그것에 release()를 호출하라.

예제: 오디오 캡쳐 설정 및 시작

아래의 예제는 오디오 캡쳐를 설정하고 시작하는 방법을 보여준다.

```
recorder = new MediaRecorder();
ContentValues values = new ContentValues(3);

values.put(MediaStore.MediaColumns.TITLE, SOME_NAME_HERE);
values.put(MediaStore.MediaColumns.TIMESTAMP, System.currentTimeMillis());
values.put(MediaStore.MediaColumns.MIME_TYPE, recorder.getMimeContentType());

ContentResolver contentResolver = new ContentResolver();

Uri base = MediaStore.Audio.INTERNAL_CONTENT_URI;
Uri newUri = contentResolver.insert(base, values);

if (newUri == null) {
   // need to handle exception here - we were not able to create a new
   // content entry
}

String path = contentResolver.getDataFilePath(newUri);

// could use setPreviewDisplay() to display a preview to suitable View here

recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

```
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
recorder.setOutputFile(path);

recorder.prepare();
recorder.start();
```

레코딩 중지

위의 예제에 기반하여, 여기에 여러분이 오디오 캡쳐를 중지하는 방법이 있다.

```
recorder.stop();
recorder.release();
```

안드로이드 지원 미디어 포맷

아래의 코어 미디어 포맷 테이블은 안드로이드 플랫폼안에서 지원되는 미디어 포맷을 설명한다. 어떤 주어진 모바일 디바이스는 여기에 나열되어 있지 않은 포맷이나 파일 타입을 추가로 지원할 수도 있다는 것에 주의하라.

편의를 위해, T-Mobile G1 미디어 포맷 테이블은 T-Mobile G1 디바이스에 의해 제공되는 추가적인 미디어 포맷과 특성들을 설명한다. 다른 디바이스들은 이 페이지에서 나열되지 않은 추가적인 포맷을 지원할 수도 있다.

애플리케이션 개발자로써, 어떤 안드로이드 디바이스 상에서 사용가능한 특정 미디어 코덱을 사용하는 것은 여러분의 자유이다. 그 코덱은 안드로이드 플랫폼에 의해 제공되는 것들과 디바이스 한정적specific인 것들을 포함한다.

코어 미디어 포맷(Core Media Formats)

타입	포맷	인코더	디코더	상세	지원되는 파일 타입
오디오	AAC LC/LTP		O	Mono/Stereo content in any combination of standard bit rates up to 160 kbps and sampling rates from 8 to 48kHz	3GPP (.3gp) and MPEG-4 (.mp4, .m4a). No support for raw AAC (.aac)
	HE-AACv1 (AAC+)		O		
	HE-AACv2 (enhanced AAC+)		O		
	AMR-NB	O	O	4.75 to 12.2 kbps sampled @ 8kHz	3GPP (.3gp)
	AMR-WB		O	9 rates from 6.60 kbit/s to 23.85 kbit/s sampled @ 16kHz 9 레이트	3GPP (.3gp)
	MP3		O	Mono/Stereo 8-320Kbps constant (CBR) or variable bit-rate (VBR)	MP3 (.mp3)
	MIDI		O	MIDI Type 0 and 1. DLS Version 1 and 2. XMF and Mobile XMF. Support for ringtone formats RTTTL/RTX, OTA, and iMelody	타입 0 그리고 1 (.mid, .xmf, .mxmf). Also RTTTL/RTX (.rtttl, .rtx), OTA (.ota), and iMelody (.imy)
	Ogg Vorbis		O		Ogg (.ogg)
오디오	PCM/WAVE		O	8- and 16-bit linear PCM (rates up to limit of hardware)	WAVE (.wav)
이미지	JPEG	O	O	base+progressive	JPEG (.jpg)
	GIF		O		GIF (.gif)
	PNG		O		PNG (.png)
	BMP		O		BMP (.bmp)
비디오	H.263	O	O		3GPP (.3gp)
	H.264	O	O		3GPP (.3gp) and MPEG-4 (.mp4)
	MPEG4 SP		O		3GPP (.3gp)

T-Mobile G1 미디어 포맷

아래 테이블은 안드로이드 플랫폼에서 지원되는 코어 미디어 포맷에 추가해서, T-Mobile G1에 의해 지원되는 미디어 포맷을 나열한다.

이 테이블은 또한 몇가지 안드로이드 코어 미디어 포맷에 대한 G1의 성능 특징을 나열한다.

타입	포맷	인코더	디코더	상세	지원되는 파일 타입
Audio	WMA		O	Supports WMA standard L1-L3: L1: 64 kbps - 161 kbps @ 44.1kHz L2: <=161 kbps <=48 kHz L3: <385 kbps <=48 kHz Mono and stereo profiles with 16-bits per sample. Decoder does not support WMA Pro, Lossless, or Speech codecs.	Windows Media Audio (.wma)
Video	WMV		O	Video Versions 7, 8 and 9. Simple profile only	Windows Media Video (.wmv)
	H.264 AVC		O	On the G1, this decoder is limited to baseline profile up to 480x320, and 600 kbps average bitrate for the video stream.	3GPP (.3gp) and MPEG-4 (.mp4)

Windows Media 코덱은 안드로이드 플랫폼의 일부가 아니고, Microsoft로부터 특별한 라이센싱을 요구하거나 Packet Video와 같은 인증된 개발자를 필요로 한다.

Section | 12

로케이션과 맵

12.1. 로케이션 서비스

12.2. 구글 맵 외부 라이브러리

12. 로케이션과 맵

로케이션location기반 또는 맵map기반 애플리케이션과 서비스들은 모바일 디바이스 사용자에게 매우 중요한 영향력을 갖는다. 여러분은 이 기능을 android.location 패키지와 구글 맵 외부 라이브러리를 사용해서 여러분의 애플리케이션에 만들 수 있다. 아래의 섹션은 그것에 대한 상세한 설명을 제공한다.

로케이션 서비스

안드로이드는 여러분의 애플리케이션이 android.location 패키지내에 있는 클래스를 통해 디바이스에 의해 지원되는 로케이션 서비스에 접근할 수 있도록 한다. 로케이션location 프레임워크의 중심 컴포넌트는 로케이션 매니저LocationManager 시스템 서비스이다. 그것은 아래에 있는 디바이스가 (만약 로케이션 기능을 지원한다면) 위치와 방향각을 결정하기 위한 API를 제공한다.

다른 시스템 서비스들과 같이, 여러분은 로케이션 매니저LocationManager를 직접적으로 인스턴스화해서는 안된다. 대신, getSystemService(Context.LOCATION_SERVICE)를 호출함으로써 시스템으로부터 로케이션 매니저LocationManager 인스턴스를 요청해야 한다.

일단 여러분의 애플리케이션이 로케이션 매니저LocationManager에 대한 핸들(handle)을 가진다면, 여러분의 애플리케이션은 다음의 세 가지를 할 수 있다.

- 마지막으로 알고 있는 위치에 대하여, 로케이션 매니저LocationManager가 알고 있는 모든 로케이션 프로바이더LocationProviders 리스트에게 쿼리를 하는 것.
- (규칙 또는 이름으로 지정되는) 로케이션 프로바이더LocationProvider에 현재 위치에 대한 주기적인 업데이트를 등록하거나 등록 해제하는 것.
- 만약 디바이스가 주어진 위도/경도의 주어진 근접 범위(미터로 된 반경으로 지정됨)내에 들어오면, 주어진 인텐트가 시작되게 하는 것을 등록하거나 등록 해제하는 것.

하지만 처음에 에뮬레이터에서 개발하는 동안 여러분은 실제 로케이션 프로바이더(네트워크 또는 GPS)로부터의 실제 데이터에 접근할 수 없을 수도 있다. 그런 경우에, 여러분은 모의mock 로케이션 프로바이더를 사용함으로써, 애플리케이션을 위한 약간의 데이터를 속이는 것이spoof 필요할 수도 있다.

> **주의** 만약 여러분이 SDK의 이전 버전에 있는 모의mock 로케이션 프로바이더 LocationProviders를 사용했다면, 여러분은 /system/etc/location 디렉토리에 있는 미리 준비된 로케이션 프로바이더LocationProviders를 더 이상 제공받지 못할 것이다. 이 디렉토리은 부팅을 하면서 없어질 것이다. 아래 요약된 새로운 절차를 따라하길 바란다.

모의 로케이션 데이터 제공하기

안드로이드 에뮬레이터에서 여러분의 애플리케이션을 테스트할 때, 모의 로케이션 데이터를 보내기 위한 두가지 서로 다른 방법이 있다. 여러분은 DDMS 도구를 사용하거나, 또는 에뮬레이터 콘솔에서 "geo" 명령 옵션을 사용할 수 있다.

DDMS 사용하기(Using DDMS)

DDMS 툴tool을 사용해서, 여러분은 몇 가지 서로 다른 방법으로 로케이션 데이터를 시뮬레이션 할 수 있다.

- 디바이스에 개별적인 위도/경도 좌표를 수작업으로 전송하는 것.
- 디바이스에 재생을 위한 경로(route)를 설명하는 GPX 파일을 사용하는 것.
- 디바이스에 순차적인 재생을 위해 개별적 장소표시placemark를 설명하는 KML 파일을 사용하는 것.

위치 데이터를 속이기 위해 DDMS를 사용하는 것에 대한 더 많은 정보에 대해서는 아래의 URL을 참조하라.

http://developer.android.com/guide/developing/tools/ddms.html#emulator-control

"geo" 명령 사용하기(Using the "geo" command)

안드로이드 에뮬레이터 안에 있는 여러분의 애플리케이션을 시작시키고, SDK의 /tools 디렉토리에서 터미널/콘솔(역주: adb shell)을 열어라. 그리고 에뮬레이터 콘솔에 연결시켜라. 이제 여러분은 다음과 같은 것을 사용할 수 있다.

- geo fix : 고정된 지리적 위치를 전송하기 위한 것.

  이 커맨드는 십진수 각도인 경도longitude와 위도latitude, 그리고 선택 사항으로 미터로 된 고도altitude를 받아들인다. 예를 들어 다음과 같다.

  geo fix -121.45356 46.51119 4392

- geo nmea : NMEA0183 구문을 전송하기 위한 것.

  이 커맨드는 '$GPGGA'(고정 데이터) 타입 또는 '$GPRMC'(추이(transit, 변화) 데이터) 타입의 하나의 NEMA 구문을 받아들인다. 예를 들어 다음과 같다.

  geo nmea $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62

에뮬레이터 콘솔에 접속하는 방법에 대한 정보에 대해서는 아래의 URL을 참조하라.

http://developer.android.com/guide/developing/tools/emulator.html#console

구글 맵 외부 라이브러리

여러분의 애플리케이션에 강력한 맵 기능을 추가하는 것을 더 쉽게 하기 위해서, 구글은 com.google.android.maps 패키지를 포함하고 있는 맵 외부 라이브러리를 제공한다. com.google.android.maps 패키지의 클래스들은 다양한 디스플레이 옵션과 컨트롤들과 함께 내장되어 있는built-in 다운로딩, 렌더링, 그리고 맵 타일들tiles의 캐싱caching을 제공한다.

맵 패키지에서 핵심 클래스는 뷰그룹의 서브클래스인 com.google.android.maps.MapView이다. 맵뷰는 구글 맵 서비스로부터 가져온 데이터를 사용해서 맵을 보여준다. 맵뷰MapView가 포커스를 가지면, 그것은 추가적인 맵 타일들에 한

네트워크 요청을 처리하는 것을 포함하여하, 맵을 자동으로 움직이고 확대 축소하기 위해 키입력과 터치 제스쳐들을 캡쳐할 것이다. 그것은 또한 맵을 제어하기 위해 사용자에게 필요한 모든 UI 엘리먼트를 제공한다. 여러분의 애플리케이션은 또한 맵뷰MapView를 프로그램적으로 제어해서 그 맵상에 다수의 오버레이overlay 타입들을 그리기 위해 맵뷰MapView 클래스 메쏘드를 사용할 수 있다.

일반적으로, 맵뷰MapView 클래스는 여러분의 애플리케이션으로 하여금 클래스 메쏘드를 통해 구글 맵 데이터를 조작할 수 있게 하는 구글 맵 API를 감싸는 래퍼wrapper를 제공하며, 그것은 여러분으로 하여금 다른 타입의 뷰처럼 맵 데이터를 가지고 작업할 수 있게 한다.

맵 외부 라이브러리는 표준 안드로이드 라이브러리의 부분이 아니다. 그러므로 그것은 어떤 안드로이드 호환 디바이스에서는 존재하지 않을 수 있다. 유사하게, 맵 외부 라이브러리는 SDK에 제공되는 표준 안드로이드 라이브러리에 포함되지 않는다. 여러분이 com.google.android.maps 패키지의 클래스를 사용해서 개발하기 위해서는, 안드로이드 SDK에 대한 구글 API의 추가 항목의 일부인 맵 외부 라이브러리를 여러분이 사용할 수 있어야 한다.

맵 외부 라이브러리와 구글 API 추가분add-on을 다운로드하고 사용하는 방법에 대해서 더 많이 배우기 위해서는 아래의 URL을 방문하라.

```
http://code.google.com/android/add-ons/google-apis
```

여러분의 편의를 위해, 구글 API 추가분add-on 또한 안드로이드 SDK안에 포함되어 있다.

주의 맵뷰MapView에서 구글 맵 데이터를 보여주기 위해, 여러분은 구글 맵 서비스에 등록해야 하며, 맵 API 키key를 얻어야 한다. 맵 API 키를 얻는 방법에 대한 정보에 대해서는 Obtaining a Maps API Key(http://code.google.com/android/add-ons/google-apis/mapkey.html)를 보라.

Section | 13

App 위젯

13.1. 기초

13.2. 매니페스트에 App 위젯 선언하기

13.3. AppWidgetProviderInfo 메타 데이터 추가하기

13.4. App 위젯 레이아웃 생성하기

13.5. AppWidgetProvider 클래스 사용하기

13.6. App 위젯 설정 액티비티 생성하기

13. App 위젯

App 위젯들은 (홈 스크린과 같은) 다른 애플리케이션 내에 임베디드될 수 있는 축소된 애플리케이션 뷰이며, 주기적인 업데이트를 수신한다. 이러한 뷰들은 사용자 인터페이스에서 위젯Widget으로써 불려진다. 그리고 여러분은 App 위젯 프로바이더를 사용해서 그것을 배포할 수 있다. 다른 App 위젯을 보유할 수 있는 애플리케이션 컴포넌트는 App 위젯 호스트라고 불리운다. 아래의 스크린 샷은 뮤직 App 위젯을 보여준다.

이 문서는 App 위젯 프로바이더를 사용하여 App 위젯을 배포하는 방식을 설명한다.

기초

App 위젯을 생성하기 위해서는, 여러분은 다음과 같은 것을 필요로 한다.

AppWidgetProviderInfo 오브젝트

App 위젯의 레이아웃, 업데이트 빈도, 그리고 AppWidgetProvider 클래스와 같은 App 위젯에 대한 메타 데이터를 기술한다. 이것은 XML로 정의되어야 한다.

AppWidgetProvider 클래스 구현

여러분이 브로드캐스트 이벤트에 기초해서 프로그램적으로 App 위젯과 인터페이스하는 것을 가능하게 하는 기본적인 메쏘드를 정의한다. 그것을 통해, 여러분은 App 위젯이 업데이트되고, 활성화되고, 비활성화되고, 그리고 삭제될 때 브로드캐스트를 수신할 것이다.

뷰(View) 레이아웃

XML로 정의된 App 위젯에 대한 초기 레이아웃을 정의한다.

추가적으로, 여러분은 App 위젯 설정configuration 액티비티를 구현할 수 있다. 이 액티비티는 선택사항이며, 그것은 사용자가 여러분의 App 위젯을 추가할 때 실행되며, 그것은 사용자가 생성 시점에서 App 위젯 설정을 수정하는 것을 가능하게 한다.

아래의 섹션들은 각각의 이러한 컴포넌트들을 구성하는 방법을 설명한다.

매니페스트에 App 위젯 선언하기

먼저, 여러분의 애플리케이션의 AndroidManifest.xml 파일에 AppWidget Provider를 선언하라. 예를 들어 다음과 같다.

```
<receiver android:name="ExampleAppWidgetProvider" >
    <intent-filter>
       <action
          android:name="android.appwidget.action.APPWIDGET_UPDATE" />
     </intent-filter>
     <meta-data android:name="android.appwidget.provider"
          android:resource="@xml/example_appwidget_info" />
</receiver>
```

〈receiver〉 엘리먼트는 App 위젯에 의해 사용되는 AppWidgetProvider를 명시하는 android:name 애트리뷰트를 요구한다.

〈intent-filter〉 엘리먼트는 android:name 애트리뷰트를 가지는 〈action〉 엘리먼트를 포함해야 한다. 이 애트리뷰트는 AppWidgetProvider가 ACTION_APPWIDGET_UPDATE 브로드캐스트를 받는다는 것을 지정해야 한다. 이 브로드캐스트는 여러분이 명시적으로 선언해야 하는 유일한 브로드캐스트이다. AppWidgetManager는 필요에 따라 모든 다른 App 위젯 브로드캐스트들을 AppWidgetProvider에게 자동으로 전송한다.

〈meta-data〉 엘리먼트는 AppWidgetProviderInfo 리소스를 지정하며, 다

음과 같은 애트리뷰트들을 요구한다.

- android:name – 데이터 이름을 지정한다. AppWidgetProviderInfo 기술자(descriptor)로써, 데이터를 식별하기 위해 android.appwidget.provider를 사용하라.
- android:resource – AppWidgetProviderInfo 리소스 위치를 지정한다.

AppWidgetProviderInfo 메타 데이터 추가하기

AppWidgetProviderInfo는 App 위젯의 필수적인 특질quality을 정의한다. 그것의 최소 레이아웃 크기, 그것의 초기 레이아웃 리소스, 얼마나 자주 App 위젯을 업데이트 하는지, 그리고 (선택 사항으로) 생성시점에서 실행할 설정configuration 액티비티와 같은 것들이 그것이다. 하나의 〈appwidget-provider〉 엘리먼트를 사용해서 XML 리소스에 AppWidgetProviderInfo를 정의하라. 그리고 그것을 프로젝트의 res/xml/ 폴더에 저장하라.

예를 들어 다음과 같다.

```
<appwidget-provider
       xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="294dp" <!-- density-independent pixels -->
    android:minHeight="72dp"
    android:updatePeriodMillis="86400000" <!-- once per day -->
    android:initialLayout="@layout/example_appwidget"
    android:configure="com.example.android.ExampleAppWidgetConfigure" >
</appwidget-provider>
```

여기에 〈appwidget-provider〉 애트리뷰트들에 대한 요약이 있다.

- minWidth와 minHeight 애트리뷰트 값은 App 위젯의 레이아웃에 의해 요구되는 최소 영역을 지정한다.

  디폴트 홈 스크린은 App 위젯을 정의된 높이와 너비을 가진 그리드grid 셀에 기반하는 자신의 윈도우에 배치한다. 만약 App 위젯의 최소 너비 또는 높이에 대한 값들이 셀의 크기에 맞지 않는다면, App 위젯 크기는 가장 근접한

셀 크기에 라운드 업round up된다(홈 스크린 셀 크기에 대한 더 많은 정보에 대해서는 http://developer.android.com의 App 위젯 디자인 가이드를 보라).

홈 스크린의 레이아웃 오리엔테이션(그리고 그에 따른 셀 크기)이 변경될 수 있기 때문에, 경험적 법칙에 따를 경우, 여러분은 셀의 높이와 너비에 대해 최악의 경우의 셀 크기 74픽셀을 가정해야 한다. 하지만 여러분은 픽셀 수에서 발생하는 임의의 정수integer 라운딩rounding 오류에 대해 고려하기 위해 최종 크기에서 2를 빼야 한다. 밀도 독립적 픽셀(dp)로 여러분의 최소 너비와 높이를 구하기 위해서는 다음의 공식을 사용하라.

```
(number of cells * 74) - 2
```

이 공식에 따라서, 여러분은 하나의 셀의 높이로 72 dp를, 4개의 셀의 너비로는 294 dp를 사용해야 한다.

- updatePeriodMillis 애트리뷰트는 얼마나 자주 App 위젯 프레임워크가 onUpdate() 메쏘드를 호출함으로써 AppWidgetProvider로부터 업데이트를 요청해야 하는가를 정의한다. 실제 업데이트는 이 값으로 지정된 시간에 정확하게 일어나는 것이 보장되지는 않는다. 그리고 우리는 가능한 한 드물게 업데이트할 것을 제안한다. 배터리를 절약하기 위해 한 시간에 많아야 한 번 하는 것이 좋다. 여러분은 또한 설정을 통해 빈도를 사용자가 조절할 수 있도록 허용할 수도 있다. 일부 사람들은 주식 시세 표시를 매 15분마다, 또는 하루에 오직 4번만 업데이트하길 원할 지도 모른다.

> **주의**
> 디바이스가 (updatePeriodMills에 정의된 대로) 절전 상태라면, 그 디바이스는 업데이트를 수행하기 위해 깨어날 것이다. 만약 여러분이 한 시간에 한 번 이상 업데이트하지 않는다면, 이것은 아마 배터리 수명에 대한 심각한 문제를 야기시키지 않을 것이다. 하지만 여러분이 너무 자주 업데이트할 필요가 있고/있거나 디바이스가 절전 상태인 동안 업데이트할 필요가 없다면, 여러분은 대신 그 디바이스를 깨울 알람에 기반하여 업데이트를 수행할 수 있다. 그렇게 하기 위해서, 알람 매니저(AlarmManager)를 이용하여 여러분의 AppWidgetProvider가 수신하는 인텐트를 가지고 하나의 알람을 설정하라. 그리고 그 알람 타입을 ELAPSED_REALTIME 또는 RTC로 설정하라. 이것은 디바이스가 깨어날 때 그 알람을 전달할 것이다. 그리고 나서 updateperioidMills를 "0"으로 설정하라.

- initialLayout 애트리뷰트는 App 위젯 레이아웃을 정의하는 레이아웃 리소스를 가리킨다.
- configure 애트리뷰트는 사용자가 App 위젯을 추가할 때 사용자에게 App 위젯 설정을 요청하기 위해 런치할 액티비티를 정의한다. 이것은 선택 사항이다(아래의 App 위젯 설정 액티비티 생성하기를 읽어라).

〈appwidget-provider〉 엘리먼트에 의해 수용되는 애트리뷰트들에 대한 더 많은 정보에 대해서는 AppWidgetProviderInfo 클래스를 보라.

App 위젯 레이아웃 생성하기

여러분은 XML로 여러분의 App 위젯에 대한 초기 레이아웃을 정의하고, 그것을 프로젝트의 res/layout/ 디렉토리에 저장해야 한다. 여러분은 아래의 리스트된 뷰 오브젝트를 사용해서 여러분의 App 위젯을 디자인 할 수 있다. 그러나 여러분의 App 위젯 디자인을 시작하기에 앞서서 App 위젯 디자인 가이드라인을 이해하라.

App 위젯 레이아웃을 생성하는 것은, 만약 여러분이 XML로 레이아웃을 선언하는 것에 익숙하다면 간단하다. 하지만 여러분은 App 위젯 레이아웃은 리모트뷰 RemoteViews에 기반한다는 것을 알아야 한다. 리모트뷰는 모든 종류의 레이아웃 또는 뷰 위젯을 지원하지 않는다.

리모트뷰RemoteViews 오브젝트 (그리고, 결과적으로 하나의 App 위젯)은 다음과 같은 레이아웃 클래스를 지원할 수 있다.

```
FrameLayout
LinearLayout
RelativeLayout
```

그리고 다음과 같은 위젯 클래스들을 지원한다.

```
AnalogClock
Button
Chronometer
ImageButton
```

```
ImageView
ProgressBar
TextView
```

이러한 클래스들의 자손들descendants은 지원되지 않는다.

AppWidgetProvider 클래스 사용하기

AppWidgetProvider 클래스는 App 위젯 브로드캐스트를 처리하기 위한 편리한 클래스로써 BroadcastReceiver를 확장한다. AppWidgetProvider는 App 위젯이 업데이트, 삭제, 활성화, 비활성화될 때와 같은 App 위젯과 관련된 이벤트 브로드캐스트만을 수신한다. 이러한 브로드캐스트 이벤트가 발생할 때, AppWidgetProvider는 다음과 같은 메쏘드 호출을 수신한다.

> 여러분은 AndroidManifest.xml(앞에서 매니페스트에 App 위젯 선언하기를 보라)의 〈receiver〉 엘리먼트를 사용해서 브로드캐스트 리시버로 여러분의 AppWidgetProvider 클래스 구현을 선언해야 한다.

onUpdate(Context, AppWidgetManager, int[])

이것은 AppWidgetProviderInfo(위의 AppWidgetProviderInfo 메타 데이터 추가하기를 보라)안의 updatePeriodMillis 애트리뷰트에 의해 정의된 간격으로 App 위젯을 업데이트하기 위해 호출된다. 이 메쏘드는 또한 사용자가 App 위젯을 추가할 때 호출된다. 그러므로 그것은 만약 필요하다면, 뷰에 대한 이벤트 핸들러를 정의하고 임시적인 서비스를 시작하는 것과 같은 필수적인 셋업을 수행해야 한다. 하지만 만약 여러분이 설정configuration 액티비티를 선언했다면, 이 메쏘드는 사용자가 App 위젯을 추가할 때 호출되지 않고, 이어지는 업데이트들에 대해서 호출된다. 설정configuration이 완료되었을 때 첫 번째 업데이트를 수행하는 것은, 설정configuration 액티비티의 책임이다(아래의 App 위젯 설정 액티비티 생성하기를 보라).

onDeleted(Context, int[])

이것은 App 위젯이 App 위젯 호스트로부터 삭제될 때마다 호출된다.

onEnabled(Context)

이것은 App 위젯 인스턴스가 최초로 생성될 때 호출된다. 예를 들어 만약 사용자가 여러분의 App 위젯에 대한 두 개의 인스턴스를 추가한다면, 이것은 오직 첫 번째만 호출된다. 만약 여러분이 새로운 데이터베이스를 오픈하거나 모든 App 위젯 인스턴스에 대해 한 번만 발생시켜야 할 필요가 있는 다른 셋업을 수행할 필요가 있다면, 이곳이 그것을 하기 좋은 위치이다.

onDisabled(Context)

이것은 App 위젯 호스트로부터 여러분의 App 위젯에 대한 마지막 인스턴스가 제거될 때 호출된다. 이곳은 여러분이 임시적인 데이터베이스를 삭제하는 것과 같은 onEnabled(Context)에서 했던 어떤 작업들을 일소clean up해야 하는 곳이다.

onReceive(Context, Intent)

이것은 모든 브로드캐스트에 대해 위의 각각의 콜백 메쏘드 이전에 호출된다. 여러분은 보통 이 메쏘드를 구현할 필요가 없다. 왜냐하면, 디폴트 AppWidgetProvider 구현은 모든 App 위젯 브로드캐스트를 여과해서 적절하게 위의 메쏘드를 호출한다.

> **주의** 안드로이드 1.5에서, onDelete() 메쏘드가 호출되어야 할 때 호출되지 않는 이미 알려진 이슈가 있다. 이 이슈를 회피하기 위해서, 여러분은 onDeleted() 콜백을 수신하기 위한 이 그룹 포스트에서 설명된대로 onReceive()를 구현할 수 있다.

가장 중요한 AppWidgetProvider 콜백은 onUpdated()이다. 왜냐하면, 그것은 (만약 여러분이 설정(configuration) 액티비티를 사용하지 않는 한) 각각의 App 위젯이 그것을 보유하는 것에 추가될 때마다 호출되기 때문이다. 만약 여러분의 App 위젯이 어떤 사용자와의 상호작용 이벤트를 받아들인다면, 여러분은 이 콜백에서 이벤트 핸들러를 등록할 필요가 있다. 만약 여러분의 App 위젯이 임시적인 파일 또는 데이터베이스를 생성하지 않거나, 또는 일소clean-up를 요구하는 다른 작업을 수행하지 않는다면, onUpdated()는 여러분이 정의할 필요가 있는 오직 유일한 콜백 메쏘드일 수도 있다. 예를 들어 만약 여러분이 클릭될 때 액티비티를 시작시키는 버튼을 가지고 있는 App 위젯을 원한다면, 여러분은 다음과 같은 AppWidgetProvider 구현을 사용할 수 있다.

```
public class ExampleAppWidgetProvider extends AppWidgetProvider {

    public void onUpdate(Context context,
       AppWidgetManager appWidgetManager, int[] appWidgetIds) {
       final int N = appWidgetIds.length;
        // Perform this loop procedure for each App Widget
       // that belongs to this provider
       for (int i=0; i<N; i++) {
           int appWidgetId = appWidgetIds[i];
           // Create an Intent to launch ExampleActivity
           Intent intent = new Intent(context, ExampleActivity.class);
           PendingIntent pendingIntent
                = PendingIntent.getActivity(context, 0, intent, 0);
          // Get the layout for the App Widget
          // and attach an on-click listener to the button
          RemoteViews views
                = new RemoteViews(context.getPackageName(), R.layout.
                        appwidget_provider_layout);
          views.setOnClickPendingIntent(R.id.button, pendingIntent);
          // Tell the AppWidgetManager to perform an update
          // on the current App Widget
          appWidgetManager.updateAppWidget(appWidgetId, views);
       }
    }
}
```

이 AppWidgetProvider는 액티비티를 런치하는 PendingIntent(계류된 인텐트)를 정의하고 setOnClickPendingIntent(int, PendingIntent)를 가진 App 위젯의 버튼에 그것을 덧붙일 목적을 위해 오직 onUpdated() 메쏘드만을 정의한다. 그것은 appWidgetIds에 있는 각각의 엔트리를 통해 반복되는 루프를 포함하고 있다. appWidgetIds는 AppWidgetProvider에 의해 생성되는 각각의 App 위젯을 식별하는 ID의 배열이다. 이런 방식으로, 사용자가 App 위젯에 대해 하나 이상의 인스턴스를 생성한다면, 그것들은 모두 동시에 업데이트된다. 하지만 오직 하나의 updatePeriodMillis 스케쥴만이 App 위젯의 모든 인스턴스에 대해서 관리될 것이다. 예를 들어 만약 업데이트 스케쥴이 매 두 시간으로 정의되고, App 위젯의 두 번째 인스턴스가 첫 번째 것 뒤에 한 시간 후에 추가된다면, 그것들은 첫 번째 것에 의해 정의된 주기로 둘 다 업데이트될 것이고, 두 번째 업데

이트 주기는 무시될 것이다(그것들은 매 시간이 아닌, 매 두 시간마다 둘 다 업데이트될 것이다).

AppWidgetProvider가 하나의 BroadcastReceiver이기 때문에, 여러분의 프로세스는 콜백 메쏘드가 리턴된 다음에 계속 실행되는 것이 보장되지 않는다(**더 많은 정보에 대해서는, 애플리케이션 기초 〉 브로드캐스드 리시버 생명주기를 보라**). 만약 여러분의 App 위젯 셋업 프로세스가 수 초 정도를 소요한 후(**아마도 웹 요청을 수행하는 동안**) 여러분의 프로세스가 계속되어야 한다면, onUpdated() 메쏘드에서 서비스를 시작시키는 것을 고려하라. 그 서비스 내에서, 여러분은 애플리케이션이 응답하지 않음(Application Not Responding, ANR) 에러 때문에 AppWidgetProvider가 강제 종료되는 것에 대한 경고 없이 App 위젯에 대한 여러분 자신의 업데이트를 수행할 수 있다. App 위젯이 서비스를 실행하는 예제에 대해서는, Wiktionary 샘플의 AppWidgetProvider를 보라.

또한 ExampleAppWidgetProvider.java 샘플 클래스를 보라.

App 위젯 브로드캐스트 인텐트 수신하기

AppWidgetProvider는 꽤 편리한 클래스이다. 만약 여러분이 App 위젯 브로드캐스트를 직접적으로 수신하고자 한다면, 여러분 자신의 Broadcast Receiver를 구현하거나 또는 onReceive(Context, Intent) 콜백을 오버라이드 할 수 있다. 여러분이 관심을 기울일 필요가 있는 4가지 인텐트가 아래에 있다.

- ACTION_APPWIDGET_UPDATE
- ACTION_APPWIDGET_DELETED
- ACTION_APPWIDGET_ENABLED
- ACTION_APPWIDGET_DISABLED

App 위젯 설정 액티비티 생성하기

만약 여러분이 사용자로 하여금 새로운 App 위젯을 추가할 때 셋팅을 설정하게 하고자 한다면, 여러분은 App 위젯 설정configuration 액티비티를 생성할 수 있다. 이 액티비티는 App 위젯 호스트에 의해 자동으로 시작될 것이며, 사용자가 App 위젯 생성 시점에 App 위젯의 색, 크기, 업데이트 주기, 또는 다른 기능 설정들과 같은 App 위젯에 대한 유용한 셋팅들을 설정하는 것을 가능하게 한다.

이 설정configuration 액티비티는 AndroidManifest.xml에 일반적인 액티비티로 정의되어야 한다. 하지만 그것은 ACTION_APPWIDGET_CONFIGURE 액션을 가지고 App 위젯 호스트에 의해 시작될 것이다. 그러므로 그 액티비티는 이 인텐트를 받아들이는 것이 필요하다. 예를 들어 다음과 같다.

```
<activity android:name=".ExampleAppWidgetConfigure">
   <intent-filter>
      <action
        android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
   </intent-filter>
</activity>
```

또한 그 액티비티는 AppWidgetProvider XML 파일에 android:configure 애트리뷰트를 가지고 정의되어야 한다(앞에서 언급된 AppWidgetProviderInfo 메타 데이터 추가하기를 보라). 예를 들어 설정configuration 액티비티는 다음과 같이 선언될 수 있다.

```
<appwidget-provider
       xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    ... >
</appwidget-provider>
```

액티비티가 전체 이름fully-qualified의 네임스페이스를 가지고 정의되는 것에 주의하라. 왜냐하면, 그것은 여러분의 패키지 영역 밖에서 레퍼런스될 것이기 때문이다.

그것이 여러분이 설정 액티비티가 시작되게 하기 위해 필요한 모든 것이다. 이제 여러분이 필요한 모든 것은 실제 액티비티이다. 하지만 여기에 여러분이 액티비티를 구현할 때 기억해야 하는 두 가지 중요한 것이 있다.

- App 위젯 호스트는 설정configuration 액티비티를 호출하며, 설정configuration 액티비티는 항상 결과를 리턴해야 한다. 결과는 액티비티를 시작했던 인텐트에 의해 전달된 (EXTRA_APPWIDGET_ID 로 인텐트 엑스트라에 저장된) App 위젯 ID를 포함해야 한다.

- onUpdate() 메쏘드는 App 위젯이 생성될 때 호출되지 않을 것이다(시스템은 설정(configuration) 액티비티가 시작될 때, ACTION_APPWIDGET_UPDATE 브로드캐스트를 보내지 않을 것이다). App 위젯이 최초로 생성될 때, AppWidgetProvider로부터 업데이트를 요청하는 것은 설정configuration 액티비티의 책임이다. 하지만 onUpdate()는 이어지는 업데이트들에 대해서 호출될 것이다. 그것은 오직 첫 번째에만 건너뛰어진다.

그 설정configuration에서 결과를 리턴하는 방법과 App 위젯을 업데이트 하는 방법의 예제에 대해, 다음 섹션에 있는 코드 단편snippet들을 보라.

설정 Activity를 통한 App 위젯 업데이트

App 위젯이 설정configuration 액티비티를 사용할 때, 설정configuration이 완료될 때 App 위젯을 업데이트하는 것은 그 설정configuration 액티비티의 책임이다. 여러분은 AppWidgetManager로부터 직접적으로 업데이트를 요청함으로써 그렇게 할 수 있다.

여기에 App 위젯을 적절하게 업데이트 하고 설정configuration 액티비티를 종료하는 절차에 대한 요약이 있다.

1. 먼저, 액티비티를 런치했던 인텐트로부터 App 위젯 ID를 얻어라.

```
Intent intent = getIntent();
Bundle extras = intent.getExtras();
if (extras != null) {
    mAppWidgetId = extras.getInt(
    AppWidgetManager.EXTRA_APPWIDGET_ID,
    AppWidgetManager.INVALID_APPWIDGET_ID);
}
```

2. 여러분의 App 위젯 설정configuration을 수행하라.

3. 설정configuration이 완료될 때, getInstance(Context) 를 호출해서 AppWidgetManager의 인스턴스를 얻어라.

```
AppWidgetManager appWidgetManager
    = AppWidgetManager.getInstance(context);
```

4. updateAppWidget(int, RemoteViews)를 호출해서 리모트뷰RemoteViews 레이아웃을 가지고 App 위젯을 업데이트하라.

```
RemoteViews views = new RemoteViews(context.getPackageName(),
R.layout.example_appwidget);
appWidgetManager.updateAppWidget(mAppWidgetId, views);
```

5. 마지막으로, 리턴 인텐트를 생성하고, 그것에 액티비티 결과result를 설정하고, 그리고 액티비티를 종료finish시켜라.

```
Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWid-
getId);
setResult(RESULT_OK, resultValue);
finish();
```

여러분의 설정configuration 액티비티가 처음 오픈될 때 액티비티 결과(result)를 RESULT_CANCELED로 설정하라. 이 방식은, 만약 사용자가 끝에 도달하기 전에 액티비티를 back-out하면, App 위젯 호스트는 그 설정이 취소되었다는 것을 통보받고 App 위젯이 추가되지 않을 것이다.

예제에 대해서는 ApiDemos에 있는 ExampleAppWidgetConfigure.java 샘플 클래스를 보라.

Section | 14

블루투스

14.1. 기초

14.2. 블루투스 퍼미션

14.3. 블루투스 설정(Setting Up)

14.4. 블루투스 검색(Finding)

14.5. 디바이스 연결

14.6. 커넥션 관리

14. 블루투스

안드로이드 플랫폼은 하나의 디바이스가 다른 블루투스 디바이스와 데이터를 교환하는 것을 허용하는 블루투스Bluetooth 네트워크 스택 지원을 포함한다. 애플리케이션 프레임워크는 안드로이드 블루투스 API를 통해 블루투스 기능에 대한 접근을 제공한다. 이 API들은 애플리케이션이 1:1(point-to-point)과 1:N(multipoint) 무선 기능이 가능한 다른 블루투스 디바이스에 무선으로 연결하는 것을 허용한다.

블루투스 API들을 사용하여 애플리케이션은 다음과 같은 것을 수행할 수 있다.

- 다른 블루투스 디바이스에 대한 스캔Scan
- 패어드paird 블루투스 디바이스에 대한 로컬 블루투스 어댑터 쿼리
- RFCOMM 채널 확립
- 서비스 검색을 통한 다른 디바이스 연결
- 다른 디바이스로의 또는 다른 디바이스로부터의 데이터 전송
- 다중 커넥션 관리

기초

이 문서는 우리에게 블루투스를 사용해서 커뮤니케이션하는데 필요한 4가지 주요 태스크task, 즉 블루투스 설정setting up, 로컬 영역내에서 패어드paired 또는 사용 가능한 디바이스의 검색, 디바이스 연결connecting, 그리고 디바이스간의 데이터 전송을 수행하는 안드로이드 블루투스 API 사용법을 설명한다.
모든 블루투스 API들은 android.bluetooth 패키지에서 제공된다. 여기에 블루투스 커넥션을 생성하기 위해 여러분이 필요로 할 클래스에 대한 개요가 있다.

BluetoothAdapter

로컬 블루투스 어댑터(블루투스 라디오)를 나타낸다. BluetoothAdapter는 모든 블루투스 상호작용에 대한 시작점이다. 이것을 사용해서, 여러분

은 다른 블루투스 디바이스들을 검색discover할 수 있고, 결합된bonded(패어드paired) 디바이스 목록을 쿼리하고, 알려진 MAC 어드레스를 사용해서 BluetoothDevice를 인스턴스화하고, 다른 디바이스와의 커뮤니케이션을 청취listen할 BluetoothServerSocket을 생성한다.

BluetoothDevice

원격 블루투스 디바이스를 나타낸다. BluetoothSocket을 통한 원격 디바이스와의 커넥션을 요청하거나 그것의 이름, 주소, 클래스, 그리고 연결 상태와 같은 디바이스에 대한 정보를 쿼리하기 위해 이것을 사용하라.

BluetoothSocket

(TCP Socket과 유사한) 블루투스 소켓에 대한 인터페이스를 나타낸다. 이것은 InputStream과 OutputStream을 통해 애플리케이션에게 또다른 블루투스 디바이스와 데이터를 교환하는 것을 허용하는 연결지점이다.

BluetoothServerSocket

(TCP ServerSocket과 유사하게) 들어오는 요청request들을 청취listen하는 오픈 서버 소켓을 나타낸다. 두개의 안드로이드 디바이스를 연결하기 위해서는 하나의 디바이스는 이 클래스를 사용해서 서버 소켓을 오픈해야 한다. 원격 블루투스 디바이스가 이 디바이스에 연결 요청을 할 때, BluetoothServerSocket은 커넥션이 수락될 때 연결된 BluetoothSocket을 리턴할 것이다.

BluetoothClass

블루투스 디바이스의 일반적인 특징과 기능들을 설명한다. 이것은 디바이스의 메이저major와 마이너minor 디바이스 클래스들과 그것의 서비스들을 정의하는 읽기전용 속성들의 집합이다. 하지만, 이것은 모든 블루투스 프로파일들profiles과 그 디바이스에 의해 지원되는 서비스들을 신뢰할만하게 설명하지는 않지만, 디바이스 타입에 대한 힌트로써는 유용하다.

블루투스 퍼미션

애플리케이션에서 블루투스 기능을 사용하기 위해서는, 여러분은 적어도 BLUETOOTH와 BLUETOOTH_ADMIN이라는 두 개의 블루투스 퍼미션 중에 하나를 선언하는 것이 필요하다.

여러분은 커넥션 요청requesting, 커넥션 수락accepting, 그리고 테이터 전송과 같은 임의의 블루투스 커뮤니케이션을 수행하기 위해 BLUETOOTH 퍼미션을 요청해야 한다.

여러분은 디바이스 검색discovery을 시작하거나 블루투스 설정을 조작manipulate하기 위해서 BLUETOOTH_ADMIN 퍼미션을 요청해야 한다. 대부분의 애플리케이션은 로컬 블루투스 디바이스를 검색하는discovery 기능에 대한 퍼미션만을 필요로 한다. 애플리케이션이 사용자의 요청에 따라 블루투스 설정을 수정할 "전원관리"가 아닌 경우에는, 이 퍼미션에 의해 부여된 나머지 기능들은 사용되어서는 안된다. **주의**: 만약 여러분이 BLUETOOTH_ADMIN 퍼미션을 사용한다면, BLUETOOTH 퍼미션 또한 보유해야 한다.

여러분의 애플리케이션 매니페스트 파일에 블루투스 퍼미션을 선언하라. 예를 들어 다음과 같다.

```
<manifest ... >
      <uses-permission android:name="android.permission.BLUETOOTH" />
      ...
</manifest>
```

애플리케이션 퍼미션을 선언하는 것에 대한 더 많은 정보는 〈uses-permission〉 레퍼런스를 보라.

블루투스 설정(Setting Up)

애플리케이션이 블루투스 너머로 커뮤니케이션할 수 있기 전에, 여러분은 디바이스에서 블루투스가 지원되는지 확인할 필요가 있다. 그리고 만약 그렇다면 그것이 활성화되도록 보장해라.

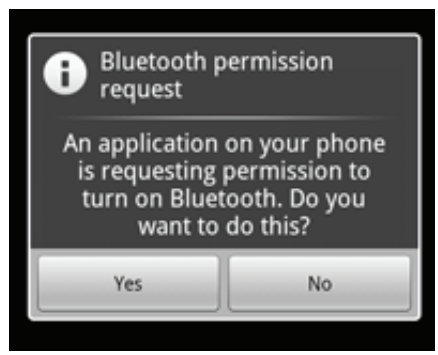


그림 1. 블루투스 활성화 다이얼로그

만약 블루투스가 지원되지 않는다면, 여러분은 적절하게 블루투스 기능을 비활성화해야 한다. 만약 블루투스가 지원되지만 비활성화라면, 여러분은

사용자가 여러분의 애플리케이션을 내에서 블루투스를 활성화하도록 요청할 수 있다. 이 설정은 BluetoothAdapter를 사용해서 두 단계로 이루어진다.

1. BluetoothAdapter 얻기

   BluetoothAdapter는 임의의 그리고 모든 블루투스 액티비티에서 필요하다. BluetoothAdapter를 얻기 위해서는, 전역static getDefaultAdapter() 메쏘드를 호출하라. 이것은 디바이스 자신의 Bluetooth 어댑터(블루투스 라디오)를 나타내는 BluetoothAdapter를 리턴한다. 전체 시스템에 대한 하나의 블루투스 어댑터가 있다. 그리고 여러분의 애플리케이션은 이 오브젝트를 사용해서 그것과 상호작용할 수 있다. 만약 getDefaultAdapter()가 널null을 리턴한다면, 그 디바이스는 블루투스를 지원하지 않으며, 여러분의 이야기는 여기서 끝난다. 예를 들어 다음과 같다.

```
BluetoothAdapter mBluetoothAdapter
    = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
}
```

2. 블루투스 활성화

   다음으로, 여러분은 블루투스가 활성화되는 것을 확실하게 할 필요가 있다. 블루투스의 현재 활성화 여부를 확인하기 위해 isEnabled()를 호출하라. 만약 이 메쏘드가 거짓false을 리턴하면, 블루투스는 비활성화되어 있다. 그 블루투스가 활성화되도록 요청하기 위해, ACTION_REQUEST_ENABLE 액션 인텐트를 가진 startActivityForResult()를 호출하라. 이것은 (여러분 애플리케이션을 종료시키지 않고) 시스템 설정을 통해 블루투스를 활성화하는 요청을 제기issue할 것이다. 예를 들어 다음과 같다.

```
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent
            = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

그림 1에서 보여지듯이, 블루투스를 활성화하기 위해 사용자 퍼미션을 요구하는 하나의 다이얼로그가 나타날 것이다. 만약 사용자가 "Yes"로 응답한다면, 시스템은 블루투스를 활성화하기 시작할 것이고, 그 과정이 완료되거나 또는 실패하자마자 여러분의 애플리케이션에게 포커스를 리턴할 것이다.

만약 블루투스 활성화가 성공하면, 여러분의 액티비티는 onActivityResult() 콜백에서 RESULT_OK 결과 코드를 수신할 것이다. 만약 블루투스가 에러로 인해 활성화되지 않았다면(또는 사용자가 "No"라고 응답했다면) 그 결과 코드는 RESULT_CANCELED가 될 것이다.

또한 여러분의 애플리케이션은 선택적으로 ACTION_STATE_CHANGED 브로드캐스트 인텐트를 수신할 수도 있다. 그것은 블루투스 상태가 변경될 때마다 시스템에 의해 발송된다. 이 브로드캐스트는 엑스트라extra 필드에 각각 신규와 이전 블루투스 상태를 포함하는 EXTRA_STATE와 EXTRA_PREVIOUS_STATE를 포함한다. 이 브로드 캐스트를 수신하는 것은 여러분의 애플리케이션이 실행되는 동안 블루투스 상태에 만들어진 변화를 감지하는데 유용할 수 있다.

TIP

검색 가능(discoverability)을 활성화하는 것은 자동으로 블루투스를 활성화할 것이다. 만약 여러분이 블루투스 액티비티를 수행하기 전에 디바이스 검색 가능(discoverability)을 항상 활성화할 계획이라면, 여러분은 위의 단계 2를 건너뛸 수 있다. 아래의 검색 가능(discoverability) 활성화를 읽어라.

블루투스 검색(Finding)

BluetoothAdapter를 사용해서, 여러분은 디바이스 검색discovery 또는 연결된bonded(패어드paired) 디바이스의 목록을 쿼리하는 것을 통해 원격 블루투스 디바이스를 찾을 수 있다.

디바이스 검색discovery은 블루투스가 활성화된 디바이스를 로컬 영역을 찾아서 각각에 대한 약간의 정보를 요청하는 스캐닝scanning 절차이다(이것은 가끔은 "discovering", "inquiring" or "scanning"으로 언급된다). 하지만, 로컬 영역 내의 하나의 블루투스 디바이스는 그것이 현재 검색 가능하게 활성화되었을 때에만 검색 요청에 응답할 것이다. 만약 디바이스가 검색 가능하다면, 그것

은 디바이스 이름, 클래스, 그리고 고유한 MAC 어드레스와 같은 약간의 정보를 공유하는 것에 의해 검색 요청에 응답할 것이다. 이 정보를 사용해서, 검색을 수행하는 디바이스는 발견된 디바이스와의 연결을 시작하도록 선택할 수 있다.

일단 커넥션이 첫 번째로 원격 디바이스에 만들어진다. 연결 요청이 자동으로 사용자에게 표시된다. 디바이스가 연결될 때, 그 디바이스에 대한 기본정보(디바이스 이름, 클래스, 그리고 MAC 어드레스와 같은)가 저장되며, 블루투스 API를 사용해서 읽혀질 수 있다. 원격 디바이스의 알려진 MAC 어드레스를 사용해서, 어떤 때는 커넥션이 (디바이스가 범위 내에 있다고 가정하고) 검색을 수행하지 않고 만들어 질 수 있다.

패어드paired 와 커넥션이 되는 것 간에는 차이가 있다는 것에 유념하라. 패어드paired가 의미하는 것은 두 개의 디바이스가 각기 다른 존재를 인식하고, 인증을 위해 사용되는 공유된 링크 키를 가지고, 그리고 각기 상대방에 대해 암호화된 커넥션을 확립할 수 있다는 것을 의미한다. 커넥션되었다는 것은 디바이스들이 RFCOMM 채널을 현재 공유하고 각기 상대에게 데이터를 전송할 수 있다는 것을 의미한다. 현재 안드로이드 블루투스 API는 RFCCOMM 커넥션이 확립될 수 있기 전에 디바이스가 패어드paired 될 것을 요구한다.(패어링pairing은 여러분이 블루투스 API를 사용해서 암호화된 커넥션을 만들 때 자동으로 수행된다.)

이어지는 섹션은 패어드paired 된 디바이스를 찾는 방법과 또는 디바이스 검색을 사용해서 신규 디바이스를 찾는것을 설명한다.

안드로이드가 탑재된 디바이스는 디폴트로 검색 가능하게 되어 있지 않다. 사용자는 시스템 설정을 통해 제한된 시간동안 디바이스가 검색 가능해 질 수 있게 만들수 있으며, 또는 애플리케이션은 사용자가 해당 애플리케이션 내에서 검색 가능(discoverability)을 활성화시키도록 요청 할 수 있다. 아래에서 검색 가능(discoverability)을 활성화하는 방법이 설명된다.

패어드(Paired) 디바이스 쿼리

디바이스 검색을 수행하기 전에, 원하는 디바이스가 이미 인식되어 있는 지 보기 위해 패어드paired 디바이스의 집합을 쿼리하는 것은 가치가 있다. 그렇게 하기 이해, getBondedDevices()를 호출하라. 이것은 패어드paired 디바이스를 보여주는 BluetoothDevice들의 집합을 리턴할 것이다. 예를 들어, 여러분은 모든 패어드 디바이스를 쿼리할 수 있다. 그리고 그런 다음에 ArrayAdapter를 사용해서 사용자에게 각각의 디바이스의 이름을 보여줄 수 있다.

```
Set<BluetoothDevice> pairedDevices
       = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {
        // Add the name and address to an array adapter
        // to show in a ListView
        mArrayAdapter.add(device.getName() + "\n"
                                + device.getAddress());
    }
}
```

커넥션은 만들기 위해 BluetoothDevice 오브젝트로부터 필요로 되는 모든 것은 MAC 어드레스이다. 이 예제에서, 사용자에게 보여지는 ArrayAdapter의 일부로써 그것이 저장된다. 그 MAC 어드레스는 나중에 커넥션을 만들기 위해 추출될 수 있다. 여러분은 디바이스 연결에 대해 이 섹션에서 커넥션을 생성하는 것에 대해 더 많이 배울 수 있다.

디바이스 검색(Discovering)

디바이스를 검색discovering하기 위해, 간단하게 startDiscovery()를 호출하라. 그 프로세스는 비동기적이며, 그 메쏘드는 바로 검색이 성공적으로 시작되었는지 여부를 가리키는 불리언boolean을 리턴할 것이다. 검색discovery 프로세스는 통상적으로 대략 12초에 한 번의 검색 스캔inquiry scan을 수반한다. 그리고 발견된 블루투스의 이름을 찾기 위해 각각의 디바이스에 대한 페이지 스캔page scan이 뒤따른다.

여러분의 애플리케이션은 발견된 각각의 디바이스에 대한 정보를 수신하기 위해 ACTION_FOUND 인텐트에 대한 BroadcastReceiver를 등록해야 한다. 각각의 디바이스를 위해, 시스템은 ACTION_FOUND 인텐트를 브로드캐스트할 것이다. 이 인텐트는 엑스트라 필드 EXTRA_DEVICE와 EXTRA_CLASS에 각각에 대응되는 BluetoothDevice와 BluetoochClass를 포함해 나를 것이다. 예를 들어, 여기에 디바이스가 발견되었을 때 브로드캐스트를 처리하기 위해 등록할 수 있는 방법이 있다.

```
// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device
               = intent.getParcelableExtra(
                       BluetoothDevice.EXTRA_DEVICE);
            // Add the name and address
            // to an array adapter to show in a ListView
            mArrayAdapter.add(device.getName() + "\n"
                                + device.getAddress());
        }
    }
};
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister
                                     // during onDestroy
```

커넥션은 만들기 위해 BluetoothDevice 오브젝트로부터 필요로 되는 모든 것은 MAC 어드레스이다. 이 예제에서, 사용자에게 보여지는 ArrayAdapter의 일부로써 그것이 저장된다. 그 MAC 어드레스는 나중에 커넥션을 만들기 위해 추출될 수 있다. 여러분은 디바이스 연결에 대해 이 섹션에서 커넥션을 생성하는 것에 대해 더 많이 배울 수 있다.

주의

디바이스 검색discovery을 수행하는 것은 블루투스 어댑터에게는 무거운 프로시저이며, 그것의 많은 리소스들을 소모할 것이다. 일단 여러분이 연결할 디바이스를 발견했다면, 커넥션을 시도하기 전에 cancelDiscovery()를 사용해서 검색discovery을 항상 중지시키는 것을 확인하라. 또한 만약 여러분이 이미 디바이스와의 커넥션을 보유했다면, 검색discovery을 수행하는 것은 커넥션에 유용한 대여폭을 현격하게 줄일 수 있다. 그러므로 여러분은 연결되어 있는 동안 검색discovery을 수행해서는 안된다.

검색 가능(discoverability) 활성화

만약 여러분이 다른 디바이스에게 로컬 디바이스가 검색가능discoverable하도록 만들고자 한다면, ACTION_REQUEST_DISCOVERABLE 액션 인텐트를 가진 startActivityForResult(Intent, int)를 호출하라. 이것은 (여러분의 애플리케이션을 중지시키지 않고) 시스템 설정을 통해 검색가능discoverable 모드를 활성화하는 요청을 만들 것이다. 디폴트로 그 디바이스는 120초동안 검색가능하게 될 것이다. 여러분은 EXTRA_DISCOVERABLE_DURATION 인텐트 엑스트라를 추가해서 다른 기간을 정의할 수 있다(최대 기간은 300 초이다). 예를 들어, 다음과 같다.

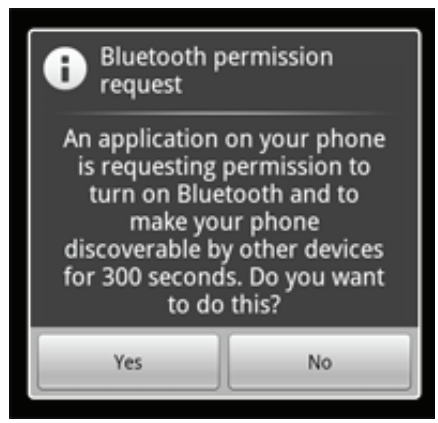


그림 2. 검색가능 활성화 다이얼로그

```
Intent discoverableIntent
       = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(
       BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```

그림 2에서 보여지듯이 디바이스를 검색가능discoverable하게 만들기 위해 사용자의 퍼미션을 요청하는 하나의 다이얼로그가 표시될 것이다. 만약 사용자가 "Yes"라고 응답한다면, 그러면 디바이스는 지정된 시간동안 검색가능하게 될

것이다. 만약 사용자가 "아니오" 응답하거나 또는 오류가 발생한다면, 그 결과 코드는 Activity.RESULT_CANCELLED가 될 것이다.

> **주의** 만약 블루투스가 디바이스에서 활성화되지 않았다면, 디바이스 검색가능 discoverability을 활성화하는 것은 자동으로 블루투스를 활성화할 것이다.

그 디바이스는 조용하게 할당된 시간동안 검색가능discoverable 모드로 남게 될 것이다. 만약 여러분이 그 검색가능한 모드가 변경되었을 때 통보받고자 한다면, 여러분은 ACTION_SCAN_MODE_CHANGED 인텐트에 대한 브로드캐스트리시버를 등록할 수 있다. 이것은 엑스트라 필드 EXTRA_SCAN_MODE와 EXTRA_PREVIOUS_SCAN_MODE를 포함할 것이다. 그것은 여러분에게 각각에 대응되는 신규 및 이전 스캔 모드를 알려준다. 각각에 가능한 값들은 SCAN_MODE_CONNECTABLE_DISCOVERABLE, SCAN_MODE_CONNECTABLE, 또는 SCAN_MODE_NONE이다. 이것은 각각에 따라, 그 디바이스가 둘 다 검색가능discoverable 모드에 있다는 것(역주: inquiry scan과 page scan이 활성화 된 상태), 검색가능 모드는 아니지만 여전히 커넥션을 수신할 수 있다는 것, 또는 검색가능 모드도 아니고 커넥션을 수신할 수도 없다는 것을 가리킨다.

만약 여러분이 원격 디바이스에 연결할 것이라면, 여러분은 디바이스 검색가능 discoverability을 활성화할 필요가 없다. 검색가능discoverability을 활성화하는 것은 오직 여러분의 애플리케이션에 들어오는 커넥션을 수락할 서버 소켓을 보유하길 원할 때 뿐이다. 왜냐하면 원격 디바이스는 그것이 커넥션을 만들 수 있기 전에 디바이스를 검색할 수 있어야 하기 때문이다.

디바이스 연결

두 개의 디바이스에 있는 여러분의 애플리케이션간의 커넥션을 생성하기 위해, 여러분은 서버쪽과 클라이언트쪽 메커니즘을 구현해야 한다, 왜냐하면 하나의 디바이스는 서버 소켓을 열어야 하고, 다른 하나는 (커넥션을 시도하기 위한 서버 디바이스의 MAC 어드레스를 사용해서) 커넥션을 만들어야 하기 때문이다.

해당 서버와 클라이언트는 동일한 RFCOMM 채널에서 그것들 각각이 연결된 BluetoothSocket을 가질 때 각기 다른 쪽에 연결되었다고 여긴다. 이 시점에, 각각의 디바이스는 입력과 출력 스트림을 획득할 수 있으며 데이타 전송이 시작될 수 있다. 그것은 이 섹션의 커넥션 관리에서 논의된다. 이 섹션은 두개의 디바이스간의 커넥션을 이루는 방법을 설명한다.

서버 디바이스와 클라이언트 디바이스 각각은 다른 방법으로 필요한 Bluetooth Socket을 획득한다. 서버는 들어오는 커넥션이 수락될 때 그것을 얻을 것이다. 클라이언트는 서버에 대한 RFCOMM 채널을 오픈할 때 그것을 얻을 것이다.

하나의 구현 기법은 각각이 서버 소켓을 오픈하고 커넥션을 정취listening하도록 서버로써 각각의 디바이스를 자동으로 준비하는 것이다. 그러면 두 디바이스는 각기 다른 것에 커넥션을 시도할 수 있으며, 클라이언트가 된다. 택일적으로, 하나의 다바이스는 명시적으로 그 커넥션을 호스트하고 요구되는 서버 소켓을 오픈할 것이며, 다른 디바이스는 간단하게 커넥션을 시도할 수 있다.

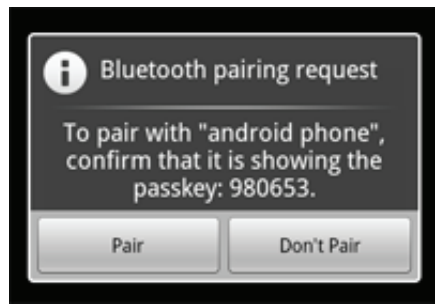


그림 3. 블루투스 페어링 다이얼로그

주의

만약 두개의 디바이스가 이전에 패어드paired된 적이 없다면, 안드로이드 프레임워크는 자동으로 패어링pairing 요청 통보 또는 그림 3에서 보여지듯이 커넥션 절차동안 사용자에게 다이얼로그를 보여준다. 그러므로 디바이스 커넥션을 시도할 때, 여러분의 애플리케이션은 디바이스가 패어드paired인 지 여부에 대해 고려할 필요가 없다. 여러분의 RFCOMM 커넥션 시도는 사용자가 성공적으로 패어드paired 될 때까지 멈춰있거나 또는 사용자가 패어링pairing을 거절, 또는 패어링pairing 실패 또는 타임 아웃되면 실패할 것이다.

서버로써의 연결

두개의 디바이스를 연결하길 원할 때, 하나는 오픈 BluetoothServerSocket를 보유함으로써 서버로써 역할해야 한다. 서버 소켓의 목적은 들어오는 커넥션 요청

을 청취listen하는 것이며, 하나가 수락될 때, 연결된 BluetoothSocket을 제공한다. BluetoothSocket이 BluetoothServerSocket으로부터 획득될 때, 만약 여러분이 더 많은 커넥션을 원하지 않는다면 그 BluetoothServerSocket은 폐기될 수 있다(되어야 한다).

> **UUID에 대해**
>
> UUID(Universally Unique Identifier)는 고유하게 정보를 식별하기 위해 사용되는 문자열 ID로 표준화된 128비트 포맷이다. UUID 핵심은 여러분이 임의의 난수random를 선택할 수 있고 그것이 충돌나지 않도록 충분히 크다는 것이다. 이 경우에, 그것은 여러분 애플리케이션의 블루투스 서비스를 식별하기 위해 고유하게 사용된다. 여러분의 애플리케이션이 사용할 UUID를 얻기 위해, 여러분은 웹에서 많은 난수 UUID 생성기 중 하나를 사용할 수 있다. 그런 다음 fromString(String)을 사용해서 UUID를 초기화하라.

여기에 서버 소켓을 셋업하고 커넥션을 수락하기 위한 기본적인 절차가 있다.

1. listenUsingRfcommWithServiceRecord(String, UUID)를 호출해서 BluetoothServerSocket을 얻어라.

   해당 문자열String은 여러분의 서비스에 대한 식별가능한 이름이다. 시스템은 이것을 자동으로 디바이스에 신규 SDP(Service Discovery Protocol) 데이터베이스 엔트리에 저장할 것이다(그 이름은 임의로 될 수도 있고 간단하게 여러분의 애플리케이션 이름이 될 수 있다). UUID는 또한 SDP 엔트리에 포함될 수 있으며 클라이언트 디바이스와의 커넥션 협약을 위한 기초가 될 것이다. 즉, 클라이언트가 이 디바이스에 연결을 시도할 때, 그것은 클라이언트가 연결하길 원하는 서비스를 고유하게 식별하는 UUID를 건낼 것이다. 이러한 UUID는 (다음단계에서) 수락되는 커넥션에 일치해야 한다.

2. accept()를 호출해서 커넥션 요청 청취listening를 시작하라.

   이것은 블락킹blocking 호출이다. 그것은 커넥션이 수락되거나 또는 예외가 발생할 때 리턴될 것이다. 커넥션은 원격 디바이스가 서버 소켓을 청취하면서 등록된 것에 일치되는 UUID를 커넥션 요청과 함께 보냈을 때에만 수락된다. 성공했을 때, accept()는 연결된 BluetoothSocket을 리턴할 것이다.

3. 만약 여러분이 추가적인 커넥션을 원하지 않는다면, close()를 호출하라.

이것은 서버 소켓과 그것의 모든 리소스를 릴리즈한다. 하지만 accept()에 의해 리턴되었던 연결된 BluetoothSocket을 닫지는 않는다. TCP/IP와 다르게, RFCOMM은 채널당 오직 한번에 하나의 연결된 클라이언트만을 허용한다. 그러므로 대부분의 경우에 그것은 연결된 소켓을 수락한 후 바로 BluetoothServerSocket에 close()를 호출하는 것이 타당하다.

accept() 호출은 메인 액티비티 UI 쓰레드에서 실행되어서는 안된다. 왜냐하면 그것은 블락킹blocking 호출이기 때문에 애플리케이션과의 임의의 다른 상호작용을 방해할 것이다. accept()와 같은 블록되는 호출을 막기 위해, 또 다른 쓰레드에서 BluetoothServiceSocket(또는 BluetoothSocket)에 close()을 호출하라. 그러면 그 블록되는 호출은 바로 리턴될 것이다. BluetoothServerSocket과 BluetoothSocket의 모든 메쏘드는 쓰레드에 안정적이다.

예제

여기에 들어오는 커넥션을 수락하는 서버 컴포넌트에 대한 간단한 쓰레드가 있다.

```
private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned
        // to mmServerSocket,
        // because mmServerSocket is final
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used
            // by the client code
            tmp = mAdapter.listenUsingRfcommWithServiceRecord(
                        NAME, MY_UUID);
        } catch (IOException e) { }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs
        // or a socket is returned
```

```
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }
            // If a connection was accepted
            if (socket != null) {
                // Do work to manage the connection
                // (in a separate thread)
                manageConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }

    /** Will cancel the listening socket,
         and cause the thread to finish */
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) { }
    }
}
```

이 예제에서, 오직 하나의 들어오는 커넥션만이 생각된다. 그러므로 커넥션이 수락되어 BluetoothSocket이 획득되자마자 애플리케이션은 별도의 쓰레드에 BluetoothSocket을 전송하고, BluetoothServerSocket을 닫고 루프를 멈춘다.

accept()가 BluetoothSocket을 리턴할 때를 주목하라. 그 소켓은 이미 연결되어 있다. 그러므로 여러분은 (클라이언트 쪽에서에서 여러분이 하는 것처럼) connect()를 호출해서는 안된다.

manageConnectedSocket()은 애플리케이션에 있는 가상의 메쏘드이다. 그것은 데이터 전송을 위한 쓰레드를 만들 것이다. 이것은 이 섹션의 커넥션 관리에서 논의된다.

통상적으로 여러분은 들어오는 커넥션에 대한 청취listening를 멈추자마자 여러분의 BluetoothServerSocket을 닫아야 한다. 이 예제에서, close()는 BluetoothSocket이 얻어지자마자 호출된다. 여러분은 또한 여러분이 쓰레드에서, 여러분이 서버 소켓에 청취를 멈출 필요가 있을 경우에 프라이빗 BluetoothSocket을 닫을 수 있는 퍼블릭 메쏘드를 제공하길 원할 수 있다.

클라이언트로써의 연결

원격 디바이스(오픈된 서버 소켓을 보유하고 있는 디바이스)에 커넥션을 만들기 위해서는, 여러분은 먼저 원격 디바이스를 나타내는 BluetoothDevice 오브젝트를 획득해야 한다. (위의 디바이스 찾기에서 나온 BluetoothDevice 얻기). 그런 다음에 여러분은 BluetoothSocket을 얻기 위해 BluetoothDevice를 사용할 수 있으며 커넥션을 만들수 있다.

여기에 기본적인 절차가 있다.

1. BluetoothDevice를 사용해서 createRfcommSocketToServiceRecord(UUID)를 호출하고 BluetoothSocket을 얻어라.

   이것은 BluetoothDevice에 연결할 BluetoothSocket을 초기화한다. 여기에 건네진 그 UUID는 서버가 (listenUsingRfcommWithServiceRecord(String, UUID)를 사용해서) 그것의 BluetoothServiceSocket을 오픈했을 때 그 서버 디바이스에 의해 사용되는 UUID와 일치해야 한다.

2. connect()를 호출해서 커넥션 만들기

   이 호출에서, 그 시스템은 UUID를 일치하기 위해 원격 디바이스에 SDP 룩업을 수행할 것이다. 만약 룩업이 성공적이고 원격 디바이스가 커넥션을 수락한다면, 그것은 커넥션 동안 사용할 RFCOMM 채널을 공유할 것이며 connect()를 리턴할 것이다. 이 메쏘드는 블락킹blocking 호출이다. 만약 어떤 이유에서든 커넥션이 실패하거나 connect() 메쏘드가 (대략 12초 이후) 타임아웃되면, 그것은 예외를 발생시킨다.

   connect()가 블락킹blocking 호출이기 때문에, 이 커넥션 프로시저는 항상 메

인 액티비티 쓰레드와 분리된 쓰레드에서 수행되어야 한다.

> **주의** 여러분은 항상 여러분이 connect()를 호출할 때 그 디바이스가 디바이스 검색discovery을 수행하지 않는가를 확인하라. 만약 검색이 진행되고 있다면, 그러면 그 커넥션 시도는 현저하게 느려지게 될 것이며, 아마도 실패할 것이다.

예제

여기에 블루투스 커넥션을 만드는 쓰레드의 기본적 예제가 있다.

```
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        // Use a temporary object that is later assigned to mmSocket,
        // because mmSocket is final
        BluetoothSocket tmp = null;
        mmDevice = device;

        // Get a BluetoothSocket to connect
        // with the given BluetoothDevice
        try {
            // MY_UUID is the app's UUID string,
            // also used by the server code
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) { }
        mmSocket = tmp;
    }

    public void run() {
        // Cancel discovery because it will slow down the connection
        mAdapter.cancelDiscovery();

        try {
            // Connect the device through the socket. This will block
            // until it succeeds or throws an exception
            mmSocket.connect();
        } catch (IOException connectException) {
```

```
            // Unable to connect; close the socket and get out
            try {
                mmSocket.close();
            } catch (IOException closeException) { }
            return;
        }

        // Do work to manage the connection (in a separate thread)
        manageConnectedSocket(mmSocket);
    }

    /** Will cancel an in-progress connection, and close the socket */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

cancelDiscovery()는 커넥션이 만들어지기 전에 호출된다는 것에 주목하라. 여러분은 항상 커넥션하기 전에 이것을 해야 하며, 그것은 그것이 실행 중인지 여부를 실제로 체크하는 것 없이 호출해도 안전하다 (하지만 만약 여러분이 체크하길 원한다면, isDiscovering() 호출하라).

manageConnectedSocket()은 애플리케이션에 있는 가상의 메쏘드이다. 그것은 데이터 전송을 위한 쓰레드를 만들 것이다. 이것은 이 섹션의 커넥션 관리에서 논의된다

여러분이 BluetoothSocket을 사용했다면, 항상 정리하기 위해 close()를 호출하라. 그렇게 하는 것은 바로 연결된 소켓을 닫을 것이며, 모든 내부 자원을 제거한다.

커넥션 관리

여러분은 성공적으로 연결된 두 개의 (또는 그 이상의) 디바이스를 가질 때, 각각은 연결된 BluetoothSocket을 가질 것이다. 이것이 즐거움이 시작하는 곳이다. 왜냐하면 여러분이 디바이스간의 데이터를 공유할 수 있기 때문이다.

BluetoothSocket을 사용해서, 임의의 데이터를 전송하는 일반적인 절차는 간단하다.

1. 소켓을 통한 전송transmission을 처리하는 InputStream과 OutputStream을 각각에 대응되는 getInputStream()과 getOutputStream()을 통해 얻어라.
2. read(byte[])와 write(byte[])를 사용해서 스트림에 데이터를 읽고 써라.

바로 그것이다.

물론, 고려할 구현 세부사항들이 있다. 다른 무엇보다도 여러분은 스트림을 읽고 쓰는 전용 쓰레드를 사용해야 한다. 이것은 중요하다. 왜냐하면 read(byte[])와 write(byte[]) 메쏘드 모두는 블락킹blocking 호출이다. read(byte[])는 스트림에서 읽혀지는 어떤 것이 있을 때까지 블락block될 것이다. write(byte[])는 통상 블락block되지 않을 것이다. 하지만 만약 원격 디바이스가 충분히 빠르게 read(byte[])를 호출하지 않아서 중간 버퍼가 풀full이라면 흐름제어를 위해 블락block될 수 있다. 그러므로 여러분의 쓰레드에서의 메인 루프는 해당 InputStream 읽기 전용이 되어서는 안된다. 쓰레드 내의 별도의 퍼블릭 메쏘드가 OutputStream에 쓰기 위해 사용될 수 있다.

예제

여기에 그것을 하는 방법에 대한 예제가 있다.

```
private class ConnectedThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        mmSocket = socket;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        // Get the input and output streams,
        // using temp objects because
        // member streams are final
```

```
        try {
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }

    public void run() {
        byte[] buffer = new byte[1024];  // buffer store for the stream
        int bytes; // bytes returned from read()

        // Keep listening to the InputStream until an exception occurs
        while (true) {
            try {
                // Read from the InputStream
                bytes = mmInStream.read(buffer);
                // Send the obtained bytes to the UI Activity
                mHandler.obtainMessage(MESSAGE_READ, bytes,
                                       -1, buffer)
                        .sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }

    /* Call this from the main Activity to send data
       to the remote device */
    public void write(byte[] bytes) {
        try {
            mmOutStream.write(bytes);
        } catch (IOException e) { }
    }

    /* Call this from the main Activity to shutdown the connection */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

생성자는 필요한 스트림을 얻고 일단 실행된다. 그 쓰레드는 InputStream을 통해 들어오는 데이터를 기다릴 것이다. read(byte[])가 스트림에서 바이트들이 리턴될 때, 그 데이터는 부모 클래스로부터 멤버 핸들러를 사용해서 메인 액티비티에 전송된다. 그런 다음 다시 돌아와서 스트림에서 더 많은 바이트를 기다린다.

데이터를 밖으로 전송하는 것은 쓰레드의 write() 메쏘드를 메인 액티비티에서 호출해서 보낼 바이트를 전달하는 것 만큼 간단하다. 그런 다음에 이 메쏘드는 간단하게 원격 디바이스에 데이터를 전송하기 위해 write(byte[])를 호출한다.

쓰레드의 cancel() 메쏘드는 BluetoothSocket을 닫음으로써 어느 때에나 커넥션이 종료될 수 있도록 하기 위해 중요하다. 이것은 항상 여러분이 Bluetooth 커넥션을 사용하고 있을 때 호출될 수 있어야 한다.

Section | 15

AIDL을 사용하여 원격 인터페이스 디자인하기

15.1. AIDL을 사용하여 IPC 구현하기

15.2. IPC 메쏘드 호출하기

15. AIDL을 사용하여 원격 인터페이스 디자인하기

각각의 애플리케이션이 자신만의 프로세스에서 실행되고, 여러분이 애플리케이션의 사용자 인터페이스와는 분리되어 있는 프로세스상에서 실행되는 서비스를 만들 수 있기 때문에, 가끔 여러분은 프로세스들간에 오브젝트를 전달해야 할 필요가 있다. 안드로이드 플랫폼상에서 하나의 프로세스는 보통 다른 프로세스의 메모리를 접근할 수 없다. 그렇기 때문에 프로세스들 간의 대화를 위해서는, 운영체제가 이해할 수 있는 프리미티브primitive 형태로 오브젝트를 분해하고 여러분을 위해 프로세스의 경계를 넘어서 오브젝트를 마샬링marshalling 할 필요가 있다.

마샬링을 하기 위한 코드를 작성하는 것은 장황하다. 그래서 우리는 여러분을 위해 그것을 해주는 AIDL 툴tool을 제공한다.

AIDL(Android Interface Definition Language , 안드로이드 인터페이스 정의 언어)은 안드로이드 디바이스에서 두 개의 프로세스가 IPC(Interprocess Communication, 프로세스 간 통신)를 사용해서 대화할 수 있는 코드를 작성하는데 사용되는 IDL 언어이다. 만약 여러분이 다른 프로세스(예를 들어 서비스)에 있는 오브젝트에 대한 메쏘드를 호출해야 하는 하나의 프로세스(예를 들어 액티비티) 내에 코드를 가지고 있다면, 파라미터를 마샬링하는 코드를 생성하는 AIDL을 사용하면 된다.

AIDL IPC 메커니즘은 COM 또는 Corba와 유사한 인터페이스 기반이지만, 보다 가볍다. 이 메커니즘은 클라이언트와 그것의 구현implementation간에 값을 전달하기 위해 프락시 클래스를 사용한다.

AIDL을 사용해서 IPC 구현하기

AIDL을 사용해서 IPC 서비스를 구현하기 위해서는 아래의 단계를 따라하라.

1. 여러분의 .aidl 파일을 생성하라 – 이 파일은 클라이언트에서 사용가능한 메쏘드들과 필드들을 정의하는 인터페이스(YourInterface.aidl)를 정의한다.

2. 여러분의 makefile에 .aidl 파일을 추가하라 – (이클립스의 ADT 플러그인은 여러분을 위해 이것을 잘 관리한다). 안드로이드는 tools/ directory에서 ADIL이라 불리는 컴파일러를 포함하고 있다.

3. 여러분의 인터페이스 메쏘드를 구현하라 – AIDL 컴파일러는 여러분의 AIDL 인터페이스로부터 Java 프로그래밍 언어로 된 인터페이스를 생성한다. 이 인터페이스는 해당 인터페이스를 상속하는 (그리고 IPC 호출을 위해 필요한 몇가지 추가적인 메쏘드를 구현하는) Stub라는 내부의 추상 클래스inner abstract class를 가진다. 여러분은 YourInterface.Stub을 확장해서 여러분의 .aidl 파일에 선언한 메쏘드를 구현하는 클래스를 만들어야만 한다.

4. 여러분의 인터페이스를 클라이언트에게 제시하라expose. 여러분이 서비스를 작성하고 있다면 여러분은 서비스를 확장해서 여러분의 인터페이스를 구현하는 여러분의 클래스에 대한 인스턴스를 리턴하는 Service.onBind(Intent)를 오버라이드 해야 한다.

.aidl 파일을 생성하라(Create an .aidl File)

AIDL은 여러분으로 하여금 파라미터와 리턴 값을 받을 수 있는 하나 또는 그 이상의 메쏘드로 인터페이스를 선언할 수 있게 하는 단순한 구문이다. 이 파라미터와 리턴 값은 어떠한 타입도 될 수 있고 심지어 다른 AIDL로부터 생성된 인터페이스일 수도 있다. 하지만 모든 내장되어built-in 있지 않은 타입들all non-built-in types이 여러분의 인터페이스로써 동일한 패키지에 정의되어 있다 할지라도 그 모든 타입들을 임포트import 해야한다는 것을 알고 있어야 한다. 여기에 AIDL이 지원하는 모든 데이터 타입들이 있다.

- 기본적인 Java 프로그래밍 언어의 타입들(정수(int), 불리언(boolean), 기타) – 임포트 import 절statement이 필요하지 않음.
- 아래의 클래스 중의 하나(임포트(import) 절(statement)이 필요하지 않음)
  - String(문자열)
  - List(리스트) – 리스트에 있는 모든 엘리먼트들은 다른 AIDL로부터 생성한 인터페이스와 파서러블parcelable을 포함한 그 리스트에 있는 타입 중 하나여야 한다. 리스트는 선택 사항으로 "일반적인generic" 클래스(예를 들어

List<String>)로써 사용되어질 수 있다. 다른 쪽에서 수신하게 될 실제의 구체적인 클래스concrete class는 해당 메쏘드가 리스트 인터페이스를 사용하기 위해 생성될지라도 항상 ArrayList가 될 것이다.

- Map – 맵에 있는 모든 엘리먼트들은 다른 AIDL로부터 생성한 인터페이스와 파서러블parcelable을 포함한 그 리스트에 있는 타입 중 하나여야 한다. (예를 들어 Map<String.Integer> 형태의) 일반적인 맵은 지원되지 않는다. 다른 쪽에서 수신하게 될 실제의 구체적인 클래스concrete class는 메쏘드가 맵 인터페이스를 사용하기 위해 생성될지라도 항상 HashMap이 될 것이다.
- CharSequence – 이것은 TextView와 다른 위젯 오브젝트에 의해 사용되는 CharSequence 타입들에 대해 유용하다.

- 다른 AIDL로부터 생성된 인터페이스들, 이것들은 항상 레퍼런스에 의해 전달된다. 이것들을 위해서 임포트import 절이 항상 필요하다.
- 파서러블Parcelable 프로토콜을 구현하고 값value에 의해 전달되는 커스텀 클래스. 이곳들을 위해서 항상 임포트import 절statement이 필요하다.

여기에 기본적인 AIDL 구문이 있다.

```
// My AIDL file, named SomeClass.aidl
// Note that standard comment syntax is respected.
// Comments before the import or package statements are not bubbled up
// to the generated interface, but comments above interface/method/field
// declarations are added to the generated interface.

// Include your fully-qualified package statement.
package com.android.sample;

// See the list above for which classes need
// import statements (hint--most of them)
import com.android.sample.IAtmService;

// Declare the interface.
interface IBankAccountService {

    // Methods can take 0 or more parameters, and
    // return a value or void.
    int getAccountBalance();
    void setOwnerNames(in List<String> names);
```

```
    // Methods can even take other AIDL-defined parameters.
    BankAccount createAccount(in String name, int startingDeposit,
        in IAtmService atmService);

    // All non-Java primitive parameters (e.g., int, bool, etc) require
    // a directional tag indicating which way the data will go. Available
    // values are in, out, inout. (Primitives are in by default, and cannot be otherwise).
    // Limit the direction to what is truly needed, because marshalling parameters
    // is expensive.
    int getCustomerList(in String branch, out String[] customerList);
}
```

인터페이스 구현하기

AIDL은 여러분의 .aidl 파일과 같은 이름을 가진 인터페이스 파일을 생성한다. 여러분이 이클립스 플러그인을 사용하고 있다면, AIDL은 자동으로 빌드 프로세스의 일부로 실행될 것이다(여러분은 먼저 AIDL을 실행하고 나서 여러분의 프로젝트를 빌드할 필요가 없다). 여러분이 이 플러그인을 사용하고 있지 않다면 ,AIDL을 먼저 실행시켜야 한다.

생성된 인터페이스는 여러분의 .aidl 파일에서 선언했던 모든 메쏘드를 선언하는 스터브Stub라고 불리는 추상 내부 클래스abstract inner class를 포함한다. 스터브Stub는 또한 몇가지 헬퍼 메쏘드들helper methods을 정의한다. 그 중에서도 가장 두드러지는 헬퍼 메쏘드인 asInterface()는 (애플리케이션의 contentext.bindService()가 성공할 때 클라이언트의 onServiceConnected()에 전달되는) IBinder를 가지며, IPC 메쏘드를 호출하기 위해 사용되는 인터페이스의 인스턴스를 리턴한다. 이것에 대한 더 자세한 사항에 대해서는 이후 나오는 "IPC 메쏘드 호출하기"를 보라.

여러분의 인터페이스를 구현하기 위해서는, YourInterface.Stub를 확장하고 메쏘드를 구현하라(여러분은 .aidl 파일을 생성하고 빌드할 필요없이 스터브(Stub) 메쏘드를 구현할 수 있다 – 안드로이드 빌드 프로세스는 .java 파일에 앞서서 .aidl 파일을 처리할 것이다).

여기에 익명의 인스턴스를 사용하는 getPid()라는 하나의 메쏘드를 드러내는 IRemoteService라는 인터페이스를 구현하는 예제가 있다.

```
// No need to import IRemoteService if it's in the same project.
private final IRemoteService.Stub mBinder = new IRemoteService.Stub(){
    public int getPid() {
        return Process.myPid();
    }
}
```

다음은 여러분의 인터페이스를 구현하는 것에 대한 몇 가지 규칙들이다.

- 여러분이 만들어내는 어떤 예외exception도 호출자에게 되돌려 보내지지 않는다.
- IPC 호출은 동기적이다. 여러분이 IPC 서비스가 완료되는데 1/1000초 이상이 걸린다는 것을 알고 있다면, 여러분은 액티비티/뷰 쓰레드에서 그것을 호출해서는 안 된다. 왜냐하면, 그것이 애플리케이션을 멈출게 할 지 모르기 때문이다(안드로이드는 "애플리케이션이 응답하지 않는다(Application is Not Responding)" 다이얼로그를 보여줄 지도 모른다). 그것을 별도의 쓰레드에서 호출하도록 시도하라.
- 오직 메쏘드만이 지원된다. 여러분은 AIDL 인터페이스에서 정적 필드static fields를 선언할 수 없다.

여러분의 인터페이스를 클라이언트에게 제시하기

이제 여러분의 인터페이스를 구현했다면, 여러분은 그 인터페이스를 클라이언트에게 드러낼 필요가 있다. 이것은 "서비스 배포하기publishing your service"로 알려져 있다. 서비스를 배포하려면 서비스Service를 상속받고, 그리고 여러분의 인터페이스를 구현하는 클래스의 인스턴스를 리턴하는 Service.onBind(Intent)를 구현하라. 여기에 IRemoteService 인터페이스를 클라이언트에 제시하는 하나의 서비스에 대한 코드 단편snippet이 있다.

```
public class RemoteService extends Service {
...
    @Override
    public IBinder onBind(Intent intent) {
       // Select the interface to return.
       // If your service only implements
```

```
        // a single interface,
        // you can just return it here without checking
        // the Intent.
        if (IRemoteService.class.getName().equals(intent.getAction())) {
            return mBinder;
        }
        if (ISecondary.class.getName().equals(intent.getAction())) {
            return mSecondaryBinder;
        }
        return null;
    }

    /**
     * The IRemoteInterface is defined through IDL
     */
    private final IRemoteService.Stub mBinder
            = new IRemoteService.Stub() {
        public void registerCallback(IRemoteServiceCallback cb) {
            if (cb != null) mCallbacks.register(cb);
        }
        public void unregisterCallback(IRemoteServiceCallback cb) {
            if (cb != null) mCallbacks.unregister(cb);
        }
    };

    /**
     * A secondary interface to the service.
     */
    private final ISecondary.Stub mSecondaryBinder
            = new ISecondary.Stub() {
        public int getPid() {
            return Process.myPid();
        }
        public void basicTypes(int anInt, long aLong, boolean aBoolean,
                float aFloat, double aDouble, String aString) {
        }
    };
}
```

파서러블을 사용해서 값 파라미터 전달

여러분이 AIDL 인터페이스를 통해 하나의 프로세스에서 다른 프로세스로 보내고자 하는 클래스가 있다면 여러분은 그것을 할 수 있다. 여러분은 해당 클래스에

대한 코드가 IPC의 다른 쪽에서도 사용가능하도록 보장해야 한다. 일반적으로 이것은 여러분이 시작한 서비스와 대화하고 있다는 것을 의미한다.

하나의 클래스가 파서러블Parcelable 프로토콜을 지원하도록 하는 다섯 가지 영역이 있다.

1. 여러분의 클래스에 파서러블Parcelable 인터페이스를 구현하라.
2. 오브젝트의 현재 상태를 얻어서 그것을 파셀parcel에 쓰는write, public void writeToParcel(Parcel out) 메쏘드를 구현하라.
3. 여러분의 오브젝트 안으로 파셀parcel 내의 값을 읽어오는 public void readFromParcel(Parcel in) 메쏘드를 구현하라.
4. Parcelable.Creator 인터페이스를 구현하는 오브젝트인 여러분의 클래스에 CREATOR 라고 불리는 정적 필드static field를 추가하라.
5. 마지막이지만 아주 중요한 것으로, (아래에서 보여지는 처럼) 여러분의 파서러블parcelable 클래스를 선언하는 aidl 파일을 생성하라. 만약 여러분이 커스텀 빌드 프로세스를 사용한다면, 여러분의 빌드에 aidl 파일을 추가하지 마라. C에서 헤더파일과 유사하게 aidl 파일은 컴파일되지 않는다.

AIDL은 여러분의 오브젝트를 마샬링하고marshall 언마샬링하기unmarshall 위해 그것이 생성하는 코드에서 위에서 언급한 메쏘드와 필드들을 사용할 것이다.

여기에 Rect 클래스가 파서러블Parcelable 프로토콜을 구현하는 방법의 예제가 있다.

```
import android.os.Parcel;
import android.os.Parcelable;

public final class Rect implements Parcelable {
    public int left;
    public int top;
    public int right;
    public int bottom;

    public static final Parcelable.Creator<Rect> CREATOR
            = new Parcelable.Creator<Rect>() {
```

```
        public Rect createFromParcel(Parcel in) {
            return new Rect(in);
        }
        public Rect[] newArray(int size) {
            return new Rect[size];
        }
    };

    public Rect() {
    }

    private Rect(Parcel in) {
        readFromParcel(in);
    }

    public void writeToParcel(Parcel out) {
        out.writeInt(left);
        out.writeInt(top);
        out.writeInt(right);
        out.writeInt(bottom);
    }

    public void readFromParcel(Parcel in) {
        left = in.readInt();
        top = in.readInt();
        right = in.readInt();
        bottom = in.readInt();
    }
}
```

여기에 위의 예제에 대한 Rect.aidl 이 있다.

```
package android.graphics;

// Declare Rect so AIDL can find it and knows that it implements
// the parcelable protocol.
parcelable Rect;
```

Rect 클래스에서 마샬링marshalling은 꽤 간단한다. 여러분이 파셀parcel에 쓸 수 있는 다른 종류의 값을 보려면 파셀Parcel에 있는 다른 메쏘드를 살펴보라.

경고: 다른 프로세스에서 데이터를 받는 것에 대한 보안 관련 사항을 잊지 마라. 이 경우 rect는 파셀parcel에서 네 가지 숫자들을 읽는다. 그러나 호출자가 시도하는 것이 무엇이든 간에 그 값들이 수용가능한 값의 범위 내에 있는지를 확실하게 하는 것은 여러분에게 달려 있다. 여러분의 애플리케이션을 악의적인 소프트웨어malware로부터 안전하게 지키는 방법에 대한 더 많은 정보는 8장. "보안 및 퍼미션"을 참고하라.

IPC 메쏘드 호출하기

여기에 호출하는 클래스가 여러분의 원격 인터페이스를 호출하기 위해 해야 하는 단계들이 있다.

1. 여러분의 .aidl 파일이 정의했던 인터페이스 타입 변수를 선언하라.
2. ServiceConnection을 구현하라.
3. 여러분의 ServiceConnection 구현을 전달하는 Context.bindService()를 호출하라.
4. 여러분의 ServiceConnection.onServiceConnected() 구현에서, 여러분은 (service라 불리우는) IBinder 인스턴스를 받을 것이다. 리턴된 파라미터를 YourInterface 타입에 캐스트하기 위한 YourInterfaceName.Stub.asInterface((IBinder)service)를 호출하라.
5. 여러분의 인터페이스에서 정의한 메쏘드를 호출하라. 여러분은 항상 DeadObjectException 예외 사항을 처리trap해야 한다. 이 예외 사항은 연결이 끊어질 때 발생한다. 이것은 원격 메쏘드에서 발생하는 유일한 예외 사항이다.
6. 연결을 끊으려면, 여러분의 인터페이스의 인스턴스를 가지고 Context.unbindService()를 호출하라.

IPC 서비스를 호출하는 것에 대한 몇 가지 커멘트comments

- 오브젝트는 프로세스들 전역에 걸쳐 레퍼런스 갯수가 계산된다.
- 여러분은 메쏘드 아규먼트로써 익명의 오브젝트를 보낼 수 있다.

여기에 ApiDemos 프로젝트의 Remote Activity 샘플에서 가져온 AIDL로 만들어진 서비스를 호출하는 것을 보여주는 예제 코드가 있다.

```
public class RemoteServiceBinding extends Activity {
    /** The primary interface we will be calling on the service. */
    IRemoteService mService = null;
    /** Another interface we use on the service. */
    ISecondary mSecondaryService = null;

    Button mKillButton;
    TextView mCallbackText;
    private boolean mIsBound;

    /* Standard initialization of this activity.  Set up the UI,
     * then wait for the user to poke it before doing anything. */

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.remote_service_binding);

        // Watch for button clicks.
        Button button = (Button)findViewById(R.id.bind);
        button.setOnClickListener(mBindListener);
        button = (Button)findViewById(R.id.unbind);
        button.setOnClickListener(mUnbindListener);
        mKillButton = (Button)findViewById(R.id.kill);
        mKillButton.setOnClickListener(mKillListener);
        mKillButton.setEnabled(false);

        mCallbackText = (TextView)findViewById(R.id.callback);
        mCallbackText.setText("Not attached.");
    }

    // Class for interacting with the main interface of the service.
    private ServiceConnection mConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className,
                IBinder service) {
            // This is called when the connection with the service
            // has been established, giving us the service object
            // we can use to interact with the service.
            // We are communicating with our service through an IDL
            // interface, so get a client-side representation of that
            // from the raw service object.
            mService = IRemoteService.Stub.asInterface(service);
            mKillButton.setEnabled(true);
```

```
        mCallbackText.setText("Attached.");

        // We want to monitor the service for as long as we are
        // connected to it.
        try {
            mService.registerCallback(mCallback);
        } catch (RemoteException e) {
            // In this case the service has crashed before we
            // could even do anything with it; we can count on
            // soon being disconnected (and then reconnected
            // if it can be restarted)
            // so there is no need to do anything here.
        }

        // As part of the sample, tell the user what happened.
        Toast.makeText(RemoteServiceBinding.this,
                   R.string.remote_service_connected,
                   Toast.LENGTH_SHORT).show();
    }
    public void onServiceDisconnected(ComponentName className) {
        // This is called when the connection with the service
        // has been unexpectedly disconnected -- that is,
        // its process crashed.
        mService = null;
        mKillButton.setEnabled(false);
        mCallbackText.setText("Disconnected.");

        // As part of the sample, tell the user what happened.
        Toast.makeText(RemoteServiceBinding.this,
                   R.string.remote_service_disconnected,
                   Toast.LENGTH_SHORT).show();
    }
};

// Class for interacting
// with the secondary interface of the service.
private ServiceConnection mSecondaryConnection
           = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
            IBinder service) {
        // Connecting to a secondary interface is the same as any
        // other interface.
        mSecondaryService = ISecondary.Stub.asInterface(service);
        mKillButton.setEnabled(true);
```

```
    }
    public void onServiceDisconnected(ComponentName className) {
        mSecondaryService = null;
        mKillButton.setEnabled(false);
    }
};

private OnClickListener mBindListener = new OnClickListener() {
    public void onClick(View v) {
        // Establish a couple connections with the service,
        // binding
        // by interface names.
        // This allows other applications to be
        // installed that replace the remote service
        // by implementing
        // the same interface.
        bindService(new Intent(IRemoteService.class.getName()),
                mConnection, Context.BIND_AUTO_CREATE);
        bindService(new Intent(ISecondary.class.getName()),
                mSecondaryConnection, Context.BIND_AUTO_CREATE);
        mIsBound = true;
        mCallbackText.setText("Binding.");
    }
};

private OnClickListener mUnbindListener = new OnClickListener() {
    public void onClick(View v) {
        if (mIsBound) {
            // If we have received the service,
            // and hence registered with
            // it, then now is the time to unregister.
            if (mService != null) {
                try {
                    mService.unregisterCallback(mCallback);
                } catch (RemoteException e) {
                    // There is nothing special we need to do
                    // if the service
                    // has crashed.
                }
            }
            // Detach our existing connection.
            unbindService(mConnection);
            unbindService(mSecondaryConnection);
            mKillButton.setEnabled(false);
```

```
            mIsBound = false;
            mCallbackText.setText("Unbinding.");
        }
    }
};

private OnClickListener mKillListener = new OnClickListener() {
    public void onClick(View v) {
        // To kill the process hosting our service,
        // we need to know its
        // PID.  Conveniently our service has a call
        // that will return
        // to us that information.
        if (mSecondaryService != null) {
            try {
                int pid = mSecondaryService.getPid();
                // Note that, though this API allows us
                // to request to
                // kill any process based on its PID,
                // the kernel will
                // still impose standard restrictions
                // on which PIDs you
                // are actually able to kill.
                // Typically this means only
                // the process running your application
                // and any additional
                // processes created by that app
                // as shown here; packages
                // sharing a common UID will also be able
                // to kill each
                // other's processes.
                Process.killProcess(pid);
                mCallbackText.setText("Killed service process.");
            } catch (RemoteException ex) {
                // Recover gracefully
                // from the process hosting the
                // server dying.
                // Just for purposes of the sample,
                // put up a notification.
                Toast.makeText(RemoteServiceBinding.this,
                        R.string.remote_call_failed,
                        Toast.LENGTH_SHORT).show();
            }
        }
```

```
        }
    };

    // Code showing how to deal with callbacks.

    // This implementation is used to receive callbacks
    // from the remote service.
    private IRemoteServiceCallback mCallback
            = new IRemoteServiceCallback.Stub() {
        /**
         * This is called by the remote service regularly
         * to tell us about
         * new values.  Note that IPC calls are dispatched
         * through a thread
         * pool running in each process,
         * so the code executing here will
         * NOT be running in our main thread like most other things
         * -- so,
         * to update the UI, we need to use a Handler to hop
         * over there.
         */
        public void valueChanged(int value) {
            mHandler.sendMessage(
               mHandler.obtainMessage(BUMP_MSG, value, 0));
        }
    };

    private static final int BUMP_MSG = 1;

    private Handler mHandler = new Handler() {
       @Override public void handleMessage(Message msg) {
           switch (msg.what) {
               case BUMP_MSG:
                   mCallbackText.setText("Received from service: " + msg.arg1);
                   break;
               default:
                   super.handleMessage(msg);
           }
       }
    };
}
```

Section | 16

애플리케이션 사인하기

16.1. 개요

16.2. 사인 전략

16.3. 사인을 위한 기본적 셋업

16.4. 디버그 모드에서 사인하기

16.5. 공식 릴리즈를 위해 사인하기

16.6. 개인 키 안전하게 하기

16. 애플리케이션 사인하기(Signing)

이 문서는 모바일 디바이스 사용자들에게 여러분의 안드로이드 애플리케이션을 배포하기 이전에 사인하는 것에 대한 정보를 제공한다.

개요

안드로이드 시스템은 설치된 모든 애플리케이션이 인증서를 사용해서 디지털적으로 사인될 것을 요구한다. 그 인증서의 개인 키private key는 애플리케이션의 개발자에 의해 보유된다. 시스템은 그 인증서를 사용자가 어떤 애플리케이션을 설치할 수 있는지를 제어하기 위해서라기 보다는, 애플리케이션의 제작자를 식별하고 애플리케이션들간의 신뢰 관계를 확립하기 위한 수단으로 사용한다. 인증서는 인증 당국authority에 의해 사인될 필요는 없다: 안드로이드 애플리케이션은 자체적으로 사인된 인증서 사용이 완전히 허용되며, 그렇게 하는 것이 일반적이다.

안드로이드 애플리케이션에 사인하는 것을 이해하는데 중요한 점은 다음과 같다.

- 모든 애플리케이션은 사인되어야 한다. 시스템은 사인되지 않은 애플리케이션은 설치하지 않을 것이다.
- 여러분은 애플리케이션에 사인하기 위해 자체적으로 사인된 인증서를 사용할 수 있다. 인증 당국은 필요하지 않다.
- 여러분은 여러분의 애플리케이션을 사용자에게 배포할 준비가 되었을 때 여러분의 애플리케이션을 알맞은 개인 키로 사인해야 한다. 여러분은 SDK 툴tool에 의해 생성한 디폴트 키로 사인된 애플리케이션을 배포할 수 없다.
- 시스템은 설치 시점에만 사인한 사람의 인증서 만료 날짜를 테스트한다. 사인된 애플리케이션의 인증서가 애플리케이션이 설치된 이후에 만료되더라도 애플리케이션은 정상적으로 동작할 것이다.
- 여러분은 키를 생성하고 여러분의 애플리케이션 .apk 파일에 사인하기 위해서

Keytool, Jarsigner와 같은 표준 툴tool을 사용할 수 있다.

- 일단 애플리케이션을 사인하면, 최종 APK 패키지를 최적화하기 위해 zipalign 툴tool을 사용하라.

안드로이드 시스템은 적절하게 사인되지 않은 애플리케이션을 설치하거나 실행하지 않을 것이다. 이것은 실제 디바이스던 또는 에뮬레이터던 상관없이 안드로이드 시스템이 구동되는 어느 곳에서든 적용된다. 이러한 이유로, 여러분은 에뮬레이터나 디바이스에서 애플리케이션을 실행하거나 디버그하기 이전에 여러분의 애플리케이션에 대한 사인을 해야 한다.

안드로이드 SDK 툴tool은 디버그할 때 여러분의 애플리케이션이 사인되게 하는 것을 지원한다. 이클립스의 ADT 플러그인과 Ant 빌드 툴tool 모두 디버그 모드debug mode와 릴리즈 모드release mode인 두 가지의 사인 모드signing mode를 제공한다.

- 개발과 테스트를 하는 동안, 여러분은 디버그 모드에서 컴파일 할 수 있다. 디버그 모드에서 빌드 툴tool은 알려진 앨리어스alias와 패스워드로 키스토어keystore와 키key를 생성하기 위해 JDK에 포함되어 있는 Keytool 유틸리티를 사용한다. 그런 다음에 그 툴tool은 각각의 컴파일 시점에 .apk 애플리케이션 파일을 사인하기 위해 디버그 키를 사용한다. 패스워드가 알려져 있기 때문에 그 툴tool은 여러분이 컴파일할 때마다 keystore/key 패스워드를 입력하도록 하는 프롬프트prompt를 필요로 하지 않는다.
- 여러분의 애플리케이션을 배포할 준비가 되었을 때, 여러분은 릴리즈 모드에서 컴파일해야 하며, 그런 다음에 여러분의 개인 키를 가지고 .apk 를 사인해야 한다. 이것을 하는 것에는 두 가지 방법이 있다.
  - 커맨드 라인command line에서 Keytool과 Jarsigner를 사용하는 것. 이 접근 방법에서, 여러분은 먼저 여러분의 애플리케이션을 사인되지 않은 .apk로 컴파일해야 한다. 그런 다음에 여러분은 .apk를 Jarsigner(또는 유사한 툴)를 사용해서 여러분의 개인 키로 수작업으로 사인해야 한다. 만약 여러분이 알맞은 개인 키를 이미 가지고 있지 않다면, 여러분 자신의 keystore/key를 생성하기 위해 수작업으로 Keytool을 실행하고 그런 다음에 Jarsigner를 사용해서 여러분의 애플리케이션을 사인할 수 있다.

- ADT Export 위저드를 사용하는 것. 여러분이 ADT 플러그인을 설치한 이클립스Eclipse에서 개발하고 있다면, 여러분은 애플리케이션을 컴파일하고 필요하다면 개인 키를 생성하고 .apk를 사인하는 모든 것을 하나의 프로세스로 하기 위해 Export 위저드를 사용할 수 있다.

일단 여러분의 애플리케이션이 사인되면, 추가적인 최적화를 위해 APK에 zipalign을 실행하는 것을 잊지마라.

사인 전략

애플리케이션 사인의 어떤 측면은 여러분의 애플리케이션 개발에 접근하는 방법에 영향을 미칠 수도 있다. 특히 여러분이 여러 개의 애플리케이션을 릴리즈하려고 한다면 그럴 수 있다.

일반적으로 모든 개발자에게 권장하는 전략은 여러분의 애플리케이션의 전체 기대 수명 동안 동일한 인증서로 여러분의 모든 애플리케이션을 사인하는 것이다. 여러분이 그렇게 해야 하는 다음과 같은 몇 가지 이유가 있다.

- **애플리케이션 업그레이드** – 여러분의 애플리케이션에 대한 업그레이드를 릴리즈할 때, 사용자가 새로운 버전을 일관되게 업그레이드 하기를 원한다면 여러분은 동일한 인증서로 업그레이드를 사인하고 싶어할 것이다. 시스템이 애플리케이션에 대한 업데이트를 설치할 때, 새로운 버전에 있는 임의의 인증서가 기존 버전 내의 있는 임의의 인증서와 일치한다면, 시스템은 업데이트를 허락한다. 만약 여러분이 일치하는 인증서를 사용하지 않고 새 버전을 사인한다면, 여러분은 또한 애플리케이션에 다른 패키지 이름을 지정하는 것이 필요할 것이다. 이 경우 사용자는 완전히 새로운 애플리케이션으로 새로운 버전을 인스톨한다.
- **애플리케이션 모듈성** – 동일한 인증서로 사인된 애플리케이션이 동일한 프로세스에서 실행되도록 그 애플리케이션들이 요구한다면, 안드로이드 시스템은 이를 허용하며, 애플리케이션들을 하나의 애플리케이션으로 취급한다. 이와 같은 방법에서 여러분은 애플리케이션을 모듈로 배치할 수 있고, 사용자는 필요할 때 각각의 모듈을 독립적으로 업데이트 할 수 있다.
- **퍼미션을 통한 코드/데이터를 공유하기** – 안드로이드 시스템은 시그니처

signature 기반의 퍼미션 강제enforcement를 제공한다. 따라서 하나의 애플리케이션은 특정 인증서로 사인된 또 다른 애플리케이션에 기능을 제공할 수 있다. 동일한 인증서로 여러 개의 애플리케이션을 사인하고 시그니처signature 기반의 퍼미션 체크를 사용함으로써, 여러분의 애플리케이션들은 안전한 방법으로 코드와 데이터를 공유할 수 있다.

여러분의 사인 전략을 결정하는 데 있어서 또 다른 중요한 고려 사항은 여러분의 애플리케이션을 사인하기 위해 사용할 키의 유효 기간을 설정하는 방법이다.

- 만약 여러분이 하나의 애플리케이션에 대한 업그레이드를 지원할 계획이라면, 여러분의 키key가 애플리케이션의 기대 수명보다 긴 유효 기간을 가지도록 해야 한다. 25년이나 그 이상의 유효 기간이 권장된다. 여러분의 키key 유효 기간이 만료되면 사용자는 더이상 여러분의 새로운 애플리케이션 버전을 일관되게 업그레이드 할 수 없을 것이다.
- 만약 여러분이 동일한 키key로 여러 개의 다른 애플리케이션을 사인할 것이라면, 여러분의 키key 유효기간을 미래에 그 애플리케이션 묶음에 추가될 수도 있는 종속된 애플리케이션을 포함하여 모든 애플리케이션의 모든 버전들의 수명보다 길게 해야 한다.
- 만약 여러분의 애플리케이션(들)을 안드로이드 마켓에 배포할 계획이라면, 애플리케이션(들)을 사인하기 위해 여러분이 사용할 키key는 2033년 10월 22일 이후의 유효 기간을 가져야 한다. 안드로이드 마켓 서버는 새로운 버전을 사용할 수 있을 때 사용자가 안드로이드 마켓 서버 상의 애플리케이션을 일관되게 업그레이드할 수 있도록 이러한 요구 사항을 강제하고 있다.

여러분의 애플리케이션을 디자인할 때, 이러한 점을 기억하고 애플리케이션을 사인하기 위한 알맞은 인증서를 사용하도록 하라.

사인을 위한 기본적 셋업

키스토어Keystore와 디버그 키debug key 생성을 지원하기 위해, 여러분은 먼저 Keytool을 SDK 빌드 툴tool에서 사용가능한 지를 확인해야 한다. 대부분의 경우에, 여러분은 적합한 JDK를 레퍼런스하기 위해 여러분의 JAVA_HOME 환경변수를 설정함으로써 SDK 빌드 툴tool이 Keytool을 찾는 방법을 알려줄 수 있

다. 다른 방법으로는, 여러분의 PATH 변수에 Keytool의 JDK 버전을 추가할 수 있다.
만약 여러분이 처음부터 Java GNU 컴파일러Compiler를 보유하고 있는 리눅스 버전에서 개발하고 있다면, 시스템이 gcj 버전이 아니라 Keytool의 JDK 버전을 사용하는 것을 보장하라. 만약 Keytool이 이미 여러분의 PATH에 있다면, 그것은 /usr/bin/keytool에서 심볼릭 링크로 포인팅pointing되어 지고 있을 지 모른다. 이 경우, 그 심볼릭 링크의 타겟이 JDK에 있는 Keytool을 가리키도록 그 심볼릭 링크 타겟을 확인하라.

만약 애플리케이션을 일반 대중에게 릴리즈할 것이라면, 여러분 머신machine에 사용가능한 Jarsigner 툴tool을 가질 필요가 있다. Jarsigner와 Keytool은 모두 JDK에 포함되어 있다.

디버그 모드에서 사인하기

안드로이드 빌드 툴tool은 디버그 사인 모드를 제공한다. 이것은 여러분의 .apk를 사인하는 것에 대한 안드로이드 시스템 요구사항을 만족시키면서, 여러분이 애플리케이션을 좀 더 쉽게 개발하고 디버그할 수 있도록 한다. 여러분이 디버그 모드를 사용할 때, SDK 툴tool은 디버그 키스토어keystore와 키key를 생성하기 위해 Keytool을 부른다invoke. 이 디버그 키는 .apk를 자동으로 사인하기 위해 사용되어서, 여러분은 여러분 자신의 키로 패키지를 사인할 필요가 없다.

SDK 도구는 미리 결정된 이름과 패스워드로 디버그 키스토어/키(keystore/key)를 생성한다.

- 키스토어Keystore 이름 – “debug.keystore”
- 키스토어Keystore 패스워드 – “android”
- 키Key 앨리어스 – “androiddebugkey”
- 키Key 패스워드 – “android”
- CN – “CN=Android Debug,O=Android,C=US”

필요하다면, 여러분은 디버그 키스토어/키keystore/key의 장소와 이름을 바꿀 수 있고, 또한 사용할 커스텀 디버그 키스토어/키keystore/key를 제공할 수 있다. 하지만 임의의 커스텀 키스토어/키keystore/key는 (위에서 설명한 대로) 디폴트 디버그 키와 동일한 키스토어/키keystore/key 이름과 패스워드를 사용해야 한다. (Eclipse/ADT에서 이를 수행하기 위해, 여러분은 Windows > Prefs > Android > Build를 사용할 수 있다.)

애플리케이션이 디버그 인증서로 사인되어 있다면, 애플리케이션을 일반 대중에게 릴리즈 할 수 없음을 유의하라.

이클립스 사용자(Eclipse Users)

만약 여러분이 Eclipse/ADT에서 개발하고 있고 위에서 언급한 대로의 셋업 Keytool을 가지고 있다면, 디폴트로 디버그 모드로 사인하는 것이 활성화된다. 여러분의 애플리케이션을 실행하거나 디버그할 때, ADT는 디버그 인증서로 .apk를 사인하고 패키지에 대한 zipalign을 수행하고 에뮬레이터 또는 연결된 디바이스에 그것을 설치한다. Keytool에 대한 접근이 ADT에 제공된다면, 여러분 입장에서 필요한 액션은 없다.

앤트 사용자(Ant Users)

만약 여러분이 .apk 파일을 빌드하기 위해 앤트Ant를 사용하고 있다면, 디버그 사인 모드는 ant 커맨드와 함께 debug 옵션을 사용함으로써 활성화된다. 여러분이 android 툴tool에 의해 생성된 build.xml 파일을 사용한다고 가정할 때 그렇다. 여러분의 애플리케이션을 컴파일하기 위해 ant debug를 실행할 때, build 스크립트는 여러분을 위해 키스토어/키keystore/key를 생성하고 .apk를 사인한다. 여러분 입장에서 필요한 액션은 없다. 더 많은 정보에 대해서는 Developing In Other IDEs: Building in debug mode(http://developer.android.com/guide/developing/other-ide.html#DebugMode)를 읽어라.

디버그 인증서의 만료

(디폴트로 이클립스/ADT와 Ant 빌드시) 디버그 모드에서 여러분의 애플리케이션을 사인하는 데 사용되는 자체 사인된self-signed 인증서는 그것이 생성된 날짜로부터 365일

의 만료 일자를 가질 것이다.

그 인증서가 만료되면, 여러분은 빌드 에러를 얻을 것이다. 앤트Ant 빌드에서, 그 에러는 아래와 같이 보인다.

```
debug:
[echo] Packaging bin/samples-debug.apk, and signing it with a debug
key...
[exec] Debug Certificate expired on 8/4/08 3:43 PM
```

Eclipse/ADT에서, 여러분은 안드로이드 콘솔console에서 비슷한 에러를 보게 될 것이다.

이 문제를 해결하기 위해서는, 간단하게 debug.keystore 파일을 삭제하라. AVD들을 위한 디폴트 저장공간 위치는 Linux/Mac OS X에서는 ~/.android/avd 이며, Windows XP에서는 C:\Documents and Settings\〈user〉\.android\ 이고, Windows Vista에서는 C:\Users\〈user〉\.android\ 이다.

여러분이 다음에 빌드할 때, 빌드 도구는 신규 키스토어keystore와 디버그 키debug key를 생성할 것이다.

여러분이 개발할 때 사용하는 기계가 태양력이 아닌non-Gregorian 로케일locale을 사용하고 있다면, 빌드 툴tool은 잘못되어 이미 만료된 디버그 인증서를 생성할 지도 모른다. 그러므로 여러분의 애플리케이션을 컴파일할 때 에러를 얻는다. 이 문제를 회피하는 법에 대한 정보에 대해서 'I can't compile my app because the build tools generated an expired debug certificate.'에 대한 아래의 URL 정보를 보라.

http://developer.android.com/guide/appendix/faq/troubleshooting.html#signingcalendar

공개 릴리즈를 위해 사인하기

여러분의 애플리케이션이 다른 사용자에게 릴리즈 할 준비가 되었을 때, 여러분은 다음과 같은 것을 해야 한다.

1. 적합한 개인 키 얻기

2. 릴리즈 모드에서 애플리케이션 컴파일하기
3. 개인 키private key로 애플리케이션 사인하기
4. 최종 APK 패키지를 얼라인align 하기

만약 여러분이 ADT 플러그인을 가진 이클립스를 사용한다면, 여러분은 개인 키private key를 가지고 apk를 컴파일하고 사인하고 얼라인align하기 위해 Export 위저드를 대신 사용할 수 있다. 그 Export 위저드는 여러분으로 하여금 그 과정에서 새로운 키스토어keystore와 개인 키private key를 생성하는 것까지 가능하게 한다. 따라서 여러분이 이클립스를 사용한다면 "이클립스 ADT를 사용해서 컴파일하고 사인하기" 섹션으로 건너 뛰어라.

1. 적합한 개인 키 사용하기

여러분의 애플리케이션에 사인하는 것을 준비하면서, 여러분은 사인할 적합한 개인 키private key를 가지고 있는 지를 먼저 확인해야 한다. 적합한 개인 키private key는 아래와 같은 것이다.

- 여러분의 소유 하에 있어야 한다.
- 그 애플리케이션에 관계된 개인, 기업, 또는 조직적 엔티티를 나타내야 한다.
- 애플리케이션이나 애플리케이션 묶음에 기대되는 수명을 넘어서는 유효 기간을 가지고 있어야 하며, 유효기간은 25년 이상이 권장된다.

  만약 여러분이 안드로이드 마켓에 여러분의 애플리케이션을 배포할 계획이라면, 유효기간이 2033년 10월 22일 이후에 끝나야 한다는 것에 유의하라. 여러분은 그 이전에 끝나는 유효 기간을 가진 키key로 사인된 애플리케이션은 업로드 할 수가 없다.

- 안드로이드 SDK 툴tool에 의해 생성된 디버그 키debug key가 아니어야 한다.

그 키key는 자체적으로 사인될 수 있다. 여러분이 적합한 키key를 가지고 있지 않다면, Keytool을 사용해서 그것을 생성해야 한다. "사인을 위한 기본적 셋업" 섹션에서 설명된 대로, 여러분이 사용할 수 있는 Keytool을 가지고 있는지 확인하라.

Keytool을 가지고 자체적으로 사인된 키key를 생성하기 위해서는, keytool 명령어를 사용해서 아래에 나열된 옵션들 중 어떤 것을 전달하라(필요하다면 다른것들도).

Warning 여러분의 개인 키를 안전하게 유지하라. 여러분이 Keytool을 실행하기 전에, 여러분의 키key를 안전하게 보유하는 방법과 그렇게 하는 것이 여러분과 사용자들에게 중요한 이유를 알기 위해 "개인 키를 안전하게 하기" 섹션을 읽도록 하라. 특히, 여러분의 키key를 생성할 때 키스토어keystore와 키key에 강력한 패스워드를 선택해야만 한다.

Keytool 옵션	설명
-genkey	(공개 키와 개인 키) 키 쌍을 생성한다.
-v	Verbose output을 가능하게 한다.
-keystore <keystore-name>. keystore	개인 키를 보관하는 keystore의 이름
-storepass <password>	Keystore를 대한 패스워드. 안전에 대한 예방조치로 여러분이 안전한 컴퓨터에서 작업을 하고 있지 않다면, 이 옵션을 커맨드 라인에 포함시키지 말아라. 만약 이것이 제공되지 않으면, Jarsigner는 여러분에게 패스워드를 넣으라고 할 것이다. 이러한 방법에서는, 여러분의 패스워드가 shell history에 저장되지 않는다.
-alias <alias_name>	키에 대한 앨리어스
-keyalg <alg>	키를 생성할 때 사용하는 암호 알고리즘. DSA와 RSA 모두 지원된다.
-dname <name>	키를 생성한 사람을 설명하는 구분되는 이름. 이 값은 자체적으로 사인된 인증서에서 이슈어(issuer)와 서브젝트 필드(subject fields)로 사용된다. 이 옵션을 커맨드 라인에서 명시할 필요는 없음을 알아두라. 적어넣지 않으면 Jarsigner가 여러분에게 각각의 구분되는 이름 필드(CN, OU 등)를 입력하라고 한다.
-validity <valdays>	키의 유효기간, 날짜 단위. 주의: 10,000 이나 그 이상의 값을 권장한다.
-keypass <password>	키 패스워드 안전에 대한 예방조치로 여러분이 안전한 컴퓨터에서 작업을 하고 있지 않다면, 이 옵션을 커맨드 라인에 포함시키지 말아라. 만약 이것이 제공되지 않으면, Jarsigner는 여러분에게 패스워드를 넣으라고 할 것이다. 이러한 방법에서는, 여러분의 패스워드가 shell history에 저장되지 않는다.

여기에 개인 키private key를 생성하는 Keytool 명령어에 대한 예제가 있다.

```
$ keytool -genkey -v -keystore my-release-key.keystore
  -alias alias_name -keyalg RSA -validity 10000
```

위의 예제 명령어를 실행하면 Keytool은 여러분에게 키스토어keystore와 키key

에 대한 패스워드를 입력하고 여러분의 키key를 구별하는 이름 필드를 입력하라고 프롬프트prompt를 보인다. 그리고 나서 키스토어keystore를 my-release-key.keystore로 불리는 파일로 생성한다. 키스토어Keystore와 키key는 여러분이 입력한 패스워드에 의해 보호된다. 키스토어Keystore는 10,000일 동안 유효한 하나의 키를 포함한다. 앨리어스alias는 여러분의 애플리케이션을 사인할 때, 키스토어keystore를 참조하기 위해 여러분이 나중에 사용할 이름이다.

Keytool에 대한 더 많은 정보는 아래의 URL을 보라.

http://java.sun.com/j2se/1.5.0/docs/tooldocs/#security

2. 릴리즈 모드에서 컴파일하기

여러분의 애플리케이션 릴리즈를 준비하기 위해, 여러분은 그것을 릴리즈 모드에서 컴파일 해야 한다. 릴리즈 모드에서, 컴파일된 애플리케이션은 디폴트로 사인되지 않으므로 여러분은 그것을 여러분의 개인 키로 사인할 필요가 있다.

주의 여러분은 사인되지 않은 애플리케이션 또는 디버그 키debug key로 사인된 애플리케이션을 릴리즈 할 수 없다.

이클립스 사용(with Eclipse)

이클립스에서 사인되지 않은 .apk를 만들기 위해서는, Package Explorer의 프로젝트에서 마우스 우측 버튼을 클릭해서 Android Tools 〉 Export Unsigned Application Package를 선택하라. 그런 다음에 사인되지 않은 .apk를 위한 파일 위치를 지정하라(다른 방법으로는, 이클립스에서 여러분의 AndroidManifest.xml 파일을 열고, Overview 탭을 열어서, Export an unsigned .apk 를 클릭하라).

여러분은 Export 위저드를 사용해서 컴파일과 사인 과정을 결합시킬 수 있다는 것에 주의하라. "이클립스 ADT를 사용해서 컴파일하고 사인하기" 섹션을 보라.

앤트 사용(with Ant)

만약 여러분이 앤트Ant를 사용하고 있다면, 여러분은 ant 명령어에 release 옵션을 사용함으로써 릴리즈 모드를 활성화할 수 있다. 예를 들어 만약 여러분의

build.xml 파일을 포함하고 있는 디렉토리에서 Ant를 실행하고 있다면 명령어는 아래와 같이 보일 것이다.

```
$ ant release
```

디폴트로, 빌드 스크립트는 애플리케이션 .apk를 사인하지 않고 빌드한다. 여러분의 프로젝트 bin/에 있는 출력파일은 〈your_project_name〉-unsigned.apk 일 것이다. 애플리케이션 .apk는 여전히 사인되어 있지 않기 때문에, 여러분은 그것을 여러분의 개인 키를 가지고 수작업으로 사인하고 zipalign을 이용하여 얼라인align해야 한다.

하지만, 여러분이 프로젝트의 build.properties 파일에 여러분의 키스토어/키 앨리어스의 경로를 제공한다면, Ant 빌드 스크립트는 또한 여러분을 위해 사인과 얼라인을 수행할 수 있다. 제공된 이 정보를 가지고, 여러분이 ant release 를 수행할 때 빌드 스크립트는 여러분의 키스토어와 앨리어스 패스워드를 입력하라고 프롬프트를 보일 것이다. 그리고 나서 빌드 스크립트는 패키지를 사인하고 얼라인할 것이다. bin/ 에 있는 최종 출력파일은 대신 〈your_project_name〉_release.apk 일 것이다. 여러분을 위한 자동화된 이 과정들을 사용하여, 여러분은 아래의 수작업 절차(과정 3, 4)를 건너 뛸 수 있다. build.properties 파일에 여러분의 키스토어와 앨리어스를 지정하는 방법을 알기 위해서는 아래의 URL에 있는 "Developing In Other IDEs: Building in release mode"를 보라.

http://developer.android.com/guide/developing/other-ide.html#ReleaseMode

3. 개인 키로 애플리케이션 사인하기

여러분이 실제 릴리즈를 위해 .apk 파일에 사인할 준비가 되었다면, 여러분은 Jarsigner 툴tool을 이용해서 그렇게 할 수 있다. "사인을 위한 기본적 셋업" 섹션에서 설명된 대로 여러분의 머신machine에 Jarsigner가 있는지 확인하라. 그리고 여러분의 개인 키private key를 가지고 있는 키스토어keystore가 유효한지도 확인하라.

여러분의 애플리케이션을 사인 하기 위해서, 여러분은 애플리케이션의 .apk와

.apk를 사인할 개인 키private key를 가지고 있는 키스토어keystore 둘 모두를 레퍼런스하는 Jarsigner를 실행한다. 아래의 테이블은 여러분이 사용할 수 있는 옵션을 보여준다.

Jarsigner 옵션	설명
-keystore <keystore-name>.keystore	여러분의 개인키를 포함하는 keystore의 이름
-verbose	Verbose output을 가능하게 한다.
-storepass <password>	Keystore에 대한 패스워드. 안전에 대한 예방조치로 여러분이 안전한 컴퓨터에서 작업을 하고 있지 않다면, 이 옵션을 명령 라인에 포함시키지 말아라. 만약 이것이 제공되지 않으면, Jarsigner가 여러분에서 패스워드를 넣으라고 할 것이다. 이러한 방법에서는, 여러분의 패스워드는 shell history에 저장되지 않는다.
-keypass <password>	개인 키 패스워드. 안전에 대한 예방조치로 여러분이 안전한 컴퓨터에서 작업을 하고 있지 않다면, 이 옵션을 명령 라인에 포함시키지 말아라. 만약 이것이 제공되지 않으면, Jarsigner가 여러분에서 패스워드를 넣으라고 할 것이다. 이러한 방법에서는, 여러분의 패스워드는 shell history에 저장되지 않는다.

여기에 위에서 생성한 예제 키스토어keystore를 사용하여 my_application.apk라고 불리는 애플리케이션 패키지에 사인하기 위한 Jarsigner 사용 방법이 있다.

```
$ jarsigner -verbose -keystore my-release-key.keystore
  my_application.apk alias_name
```

위의 예제 명령을 실행하면, Jarsigner가 여러분에게 키스토어keystore와 키key에 대한 패스워드를 입력하라고 한다. 그러면 그것은 .apk를 수정하며, 이는 .apk가 이제 사인되었다는 것을 의미한다. 여러분이 다른 키key로 .apk를 여러 번 사인할 수 있음에 유의하라.

여러분의 .apk가 사인되었는 지를 확인하기 위해, 여러분은 아래와 같은 명령어을 사용할 수 있다.

```
$ jarsigner -verify my_signed.apk
```

만약 .apk가 제대로 사인되었다면, Jarsigner는 "jar verfied"라고 출력한다. 더 자세한 정보를 원한다면 아래 명령어 중의 하나를 시도해 볼 수 있다.

```
$ jarsigner -verify -verbose my_application.apk
```

또는

```
$ jarsigner -verify -verbose -certs my_application.apk
```

위의 명령어에 -certs 옵션이 추가되면, 여러분에게 누가 키key를 생성했는지를 설명하는 "CN=" 라인을 보여줄 것이다.

만약 여러분이 "CN=Android Debug"를 보게 된다면, 이것은 .apk가 안드로이드 SDK가 생성한 디버그 키key로 사인되었음을 의미한다. 여러분의 애플리케이션을 릴리즈하고자 한다면, 디버그 키debyg key 대신 여러분의 개인 키private key로 사인해야만 한다.

Jarsigner에 대한 더 많은 정보에 대해서 아래의 URL을 보라.

http://java.sun.com/j2se/1.5.0/docs/tooldocs/#security

4. 최종 APK 패키지를 얼라인하기

일단 여러분이 여러분의 개인 키로 .apk를 사인했다면, 그 파일에 대해서 zipalign을 수행하라. 이 도구는 모든 압축되지 않은 데이터가 그 파일의 시작으로부터 상대적으로 특정 바이트 얼라인먼트를 가지고 시작한다는 것을 보장한다. 4-바이트 경계에서 얼라인먼트를 보장하는 것은 패키지가 디바이스 상에 설치될 때 성능 최적화를 제공한다. 파일들이 얼라인먼트 제약을 가진 바이너리 데이터를 포함한다 할지라도 파일들이 얼라인될 때, 안드로이드 시스템은 패키지로부터 모든 데이터를 복사하기 보다는 mmap()을 사용해서 파일들을 읽을 수 있다.

zipalign 도구는 Android SDK와 함께 제공되며, tools/ 디렉토리안에 위치해 있다. 여러분의 사인된 .apk를 얼라인하기 위해 아래를 수행하라.

```
zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

-v 플래그는 verbose 출력(선택사항임)을 활성화한다. 4는 바이트-얼라인먼트이다. (4가 아닌 값을 사용하지 마라) 첫번째 파일인자는 여러분의 사인된 .apk(입력)이고 두번째 파일인자는 최종 .apk 파일(출력)이다. 여러분이 이미 존재하는 .apk를 오버라이드할 것이라면, -f 플래그를 추가하라.

주의 : 여러분은 zipalign을 사용해서 패키지를 최적화하기 전에 여러분의 입력

.apk는 여러분의 개인 키를 가지고 사인되어야 한다. 만약 여러분이 zipalign을 사용한 후 패키지를 사인한다면, 패키지는 얼라인먼트 이전 상태로 돌아갈 것이다. 더 많은 정보에 대해서는 zipalign 도구에 대해서 읽어라.(http://developer.android.com/guide/developing/tools/zipalign.html)

이클립스 ADT를 사용해서 컴파일하고 사인하기

여러분이 ADT를 가진 이클립스를 사용한다면, 여러분은 사인된 .apk를 만들기 위해 Export 위저드를 사용할 수 있다(심지어 필요하다면 새로운 키스토어(keystore)를 생성할 수 있다). Export 위저드는 여러분을 위해 Keytool 및 Jarsigner와의 모든 상호작용을 수행한다. 이는 여러분으로 하여금 패키지를 컴파일하고 사인하고 얼라인하기 위해 수작업 절차를 수행하는 대신에 GUI를 이용하여 사인하도록 허용한다. 위저드가 일단 여러분의 패키지를 컴파일하고 사인하면, zipalign을 이용하여 패키지 얼라인먼트를 수행할 것이다. Export 위저드가 Keytool과 Jarsigner 모두를 사용하기 때문에, 여러분은 "사인을 위한 기본적 셋업"에서 위에 설명한 대로 여러분의 컴퓨터에서 그것들이 접근 가능하다는 것을 보장해야 한다.

이클립스에서 사인되고 얼라인된 .apk를 생성하기 위해,

1. Package Explorer에서 프로젝트를 선택하고 File 〉 Export를 선택하라.
2. 안드로이드 폴더를 열고, Export Android Application을 선택하고, Next를 클릭하라.

   Export Android Application 위저드는 이제 시작하며, 이것은 (새로운 키스토어와 개인 키 생성 또는) .apk를 사인하기 위해 개인 키를 선택하는 과정들을 포함하여 여러분의 애플리케이션을 사인하는 과정을 통해 여러분을 안내해 줄 것이다.
3. Export 위저드를 종료하라. 그러면 여러분의 애플리케이션은 컴파일되고, 사인되고, 얼라인되고 배포할 준비가 되어 있을 것이다.

개인 키 안전하게 하기

여러분의 개인 키private key에 대한 안전을 유지하는 것은, 여러분과 사용자 모두에게 매우 중요하다. 만약 여러분의 키key를 다른 사람이 사용할 수 있도록 여러분의 키스토어keystore와 패스워드를 안전하지 않은 장소에 놓아 두어 제3자가 찾아서 사용할 수 있게 한다면 여러분의 제작 신원authoring identity과 사용자의 신뢰 모두 훼손된다.

만약 여러분의 인지 또는 승인없이 제3자가 여러분의 키key를 가지게 된다면, 그 사람은 여러분의 원래 애플리케이션을 악의적으로 바꾸고 오류를 발생시키는 애플리케이션을 사인해서 배포할 수 있다. 그 사람은 또한 다른 애플리케이션이나 시스템을 공격하거나 사용자 데이터에 오류를 일으키거나 그것을 훔치는 애플리케이션을 여러분의 신원identity으로 사인해서 배포할 수도 있다.

개발자로서 여러분의 명성은 여러분의 개인 키private key가 만료될 때까지 그 키를 언제나 올바르게 안전하게 하는 지에 달려있다. 여기에 여러분의 키key를 안전하게 하는 몇 가지 팁이 있다.

- 키스토어keystore와 키key에 대한 강력한 패스워드를 선택하라.
- Keytool로 여러분의 키를 생성할 때, 커맨드 라인에 -storepass, -keypass 옵션을 사용하지 마라. 만약 사용한다면 여러분의 패스워드는 shell history에 남아있게 되어 여러분 컴퓨터의 다른 사용자 누구라도 접근이 가능하게 된다.
- 그와 비슷하게 Jarsigner로 애플리케이션을 사인할 때, 커맨드 라인에 -storepass, -keypass 옵션을 사용하지 마라.
- 여러분의 개인 키private key는 누구에게라도 주거나 빌려주지 말아라. 권한이 없는 사람들이 여러분의 키스토어keystore와 키 패스워드를 알게 하지 말아라.

일반적으로 여러분의 키key를 생성, 사용, 저장할 때, 상식 선의 주의사항들을 따른다면 키는 안전하게 유지될 것이다.

Section | 17

애플리케이션 버전 부여

17.1. 애플리케이션 버전 설정

17.2. 애플리케이션의 시스템 API 요건 지정하기

17. 애플리케이션 버전 부여

버전을 부여하는 것은 여러분의 애플리케이션을 업그레이드/유지보수하기 위한 전략의 중요한 요소이다.

- 사용자는 자신들의 디바이스에 설치되는 애플리케이션 버전와 설치가 가능한 업그레이드 버전에 대한 분명한 정보를 가지는 것이 필요하다.
- 다른 애플리케이션들은 – 여러분이 하나의 묶음으로써 배포하는 다른 애플리케이션들을 포함함 – 호환성을 결정하고 의존도를 확인하기 위해, 시스템에 애플리케이션 버전을 질의하는 것이 필요하다.
- 여러분의 애플리케이션을 배포할 때 사용하는 서비스는 여러분의 애플리케이션에게 그것의 버전을 질의할 필요가 있을 수 있다. 이로써 서비스는 사용자들에게 그 버전을 보여줄 수 있다. 또한 배포 서비스는 호환성을 결정하고 업그레이드/다운그레이드 관계를 확립하기 위해서 애플리케이션 버전을 확인할 필요가 있을 수도 있다.

안드로이드 시스템은 업그레이드, 호환성 등에 대한 제약을 강제하기enforce 위해 애플리케이션에 대한 애플리케이션 버전 정보를 확인하지는 않는다. 대신에 사용자나 애플리케이션 스스로가 애플리케이션에 대한 임의의 버전 제약을 강제하는 것에 대해 책임이 있다.

안드로이드 시스템은 애플리케이션의 매니페스트의 minSdkVersion 애트리뷰트에 나타나는 시스템 버전 호환성을 체크한다. 이것은 애플리케이션으로 하여금 호환가능한 최소한의 시스템 API를 지정하는 것을 허용한다. 더 많은 정보는 17장. 애플리케이션 버전 부여하기의 최소 시스템 API 버전 지정하기를 보라.

애플리케이션 버전 설정

애플리케이션 버전 정보를 정의하기 위해서, 여러분은 애플리케이션의 매니페스트 파일의 애트리뷰트를 설정한다. 두 개의 애트리뷰트가 사용 가능하며, 여러분은 두 개의 애트리뷰트 값 모두를 정의해야만 한다.

- android:versionCode – 다른 버전에 상대적인 애플리케이션 코드 버전을 나타내는 정수integer 값.

  이 값은 예를 들어 업그레이드나 다운그레이드 관계를 확인하기 위해서 다른 애플리케이션이 프로그램적으로 그것을 평가할 수 있는 정수integer 값이다. 여러분이 원하는 임의의 정수integer로 그 값을 설정할 수 있다. 하지만 여러분은 애플리케이션 각각의 다음 릴리즈가 더 큰 값을 사용하도록 보장해야 한다. 시스템이 이것을 강제하는 것은 아니지만, 다음 릴리즈 값을 증가시키는 것이 표준이다.

  일반적으로, 여러분은 첫 번째 애플리케이션 버전에 versionCode 1을 설정해서 릴리즈한다. 그런 다음에는 그 릴리즈가 메이저 또는 마이너 릴리즈든 상관없이 단순하게 각각의 릴리즈 값을 증가시킨다. 이것은 android:versionCode 값이 사용자에게 보여지는 애플리케이션 릴리즈 버전을 꼭 닮지 않아도 되는 것을 의미한다(아래의 android:versionName를 보라). 애플리케이션과 배포 서비스는 사용자에게 버전 값을 보여주지 않는다.

- android:versionName – 사용자에게 보여져야 하는 애플리케이션 코드의 릴리즈 버전을 나타내는 문자열 값.

  이 값은 여러분이 애플리케이션 버전을 설명할 수 있도록 하는 하나의 문자열이다. 그것은 〈major〉.〈minor〉.〈point〉 문자열로써 표현되거나, 또는 절대적 또는 상대적인 어떤 다른 버전 식별자 타입으로 표현된다.

  Android:versionCode와는 다르게, 시스템은 이 값을 어떤 내부 목적으로도 사용하지 않는다. 그 보다는 애플리케이션은 이 값을 사용자에게 보여주기 위해 사용한다. 배포 서비스는 또한 사용자에게 보여주기 위해, android:versionName 값을 추출해 낼 수도 있다.

여러분은 매니페스트 파일의 〈manifest〉 엘리먼트에 있는 이러한 버전 애트리뷰

트들 모두를 정의한다.

여기에 〈manifest〉 엘리먼트에 있는 android:versionCode와 android:versionName 애트리뷰트들을 보여주는 예제 매니페스트가 있다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.package.name"
      android:versionCode="2"
      android:versionName="1.1">
    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
        ...
    </application>
</manifest>
```

이 예제에서, android:versionCode 값은 현재의 .apk가 애플리케이션 코드의 두 번째 릴리즈를 포함하지만, android:codeName 문자열에 의해 보여지는 것처럼 마이너한 후속의 릴리즈에 해당한다는 것에 유의하라.

안드로이드 프레임워크는 애플리케이션으로 하여금 여러분의 애플리케이션에 관한 버전 정보를 시스템에 재질의requery하는 API를 제공한다. 버전 정보를 얻기 위해, 애플리케이션은 패키지 매니저PackageManger에 있는 getPackageInfo(java.lang.String, int) 메쏘드를 사용한다.

애플리케이션의 시스템 API 요건 지정하기

여러분의 애플리케이션이 안드로이드 플랫폼의 특정 최소 버전을 요구하거나 특정 버전의 안드로이드 플랫폼 버전을 지원하도록만 설계되어 있다면, 여러분은 애플리케이션의 매니페스트 파일의 API Level 식별자를 사용하여 그 버전을 명시할 수 있다. 그렇게 하는 것은 여러분의 애플리케이션이 호환가능한 버전의 안드로이드 시스템을 실행하고 있는 디바이스에만 설치될 수 있다는 것을 보장한다.

API 레벨 요구사항을 지정하기 위해, 아래와 같이 하나 또는 그 이상의 애트리뷰트를 이용하여 애플리케이션의 매니페스트에 〈uses-sdk〉 엘리먼트를 추가하라.

- android:minSdkVersion – 애플리케이션이 실행될 안드로이드 플랫폼의 최소 버전으로 API 레벨 식별자에 의해 지정된다.
- android:targetSdkVersion – 애플리케이션이 실행되도록 계획된 API를 지정한다. 몇몇 경우에, 이것은 최초 API 레벨을 위해 정의된 것들만을 사용하는 것으로 제약되기보다는, 애플리케이션이 타켓 API 레벨에 정의된 매니페스트의 엘리먼트 또는 작동형태를 사용하는 것을 가능하게 한다.
- android:maxSdkVersion – 애플리케이션이 실행될 안드로이드 플랫폼의 최대 버전으로 API 레벨 식별자에 의해 지정된다. Important: 이 애트리뷰트를 사용하기 전에 〈uses-sdk〉 문서를 읽어라.

애플리케이션을 설치하려고 준비할 때, 시스템은 이 애트리뷰트 값을 확인해서 그것을 시스템 버전과 비교한다. android:minSdkVersion 값이 시스템 버전보다 높다면 시스템은 애플리케이션 설치를 중단한다. 비슷하게, android:maxSdkVersion이 플랫폼 버전과 호환가능하면 시스템은 여러분의 애플리케이션을 설치할 것이다.

만약 여러분의 매니페스트에 이 애트리뷰트를 지정하지 않는다면, 시스템은 여러분의 애플리케이션이 최대 API 레벨없이 모든 플랫폼 버전과 호환된다고 가정한다.

애플리케이션의 최소 플랫폼 버전을 지정하기 위해서는, 〈manifest〉의 자식 child에 〈uses-sdk〉 엘리먼트를 추가하라. 그리고 나서 애트리뷰트로 android:minSdkVersion을 정의하라.

더 많은 정보를 위해서는, 〈uses-sdk〉 매니페스트 엘리먼트 문서와 API 레벨 문서를 보라.

Section | 18

배포를 위한 준비 : 체크리스트

18.1. 애플리케이션 릴리즈 준비 이전 단계

18.2. 애플리케이션 최종 컴파일 이전 단계

18.3. 애플리케이션 컴파일 이후 단계

18. 배포하기 위한 준비 : 체크리스트

애플리케이션을 배포한다는 것은 그것을 테스트하고 적절하게 패키징하고 안드로이드 모바일 디바이스 사용자들에게 사용가능하도록 만든다는 것을 의미한다.

만약 여러분이 안드로이드 디바이스에 설치가 가능하도록 여러분의 애플리케이션을 배포하려고 한다면, 여러분의 애플리케이션이 준비되도록 하기 위해 해야 할 몇가지 일들이 있다. 이 문서는 성공적인 릴리즈를 위해 여러분의 애플리케이션을 준비하는 것에 대한 중요한 체크포인트를 강조한다.

만약 여러분이 안드로이드 마켓에 애플리케이션을 배포하고자 한다면, 또한 여러분의 애플리케이션을 위한 구체적 준비 요구사항에 대해서 19장. "애플리케이션 배포하기"에서 "안드로이드 마켓에 배포하기"를 보라.

애플리케이션을 배포하는 방법에 대한 일반적인 정보는 19장. "애플리케이션 배포하기"를 보라.

여러분이 애플리케이션 릴리즈 준비를 고려하기 전에

1. 실제 디바이스에서 여러분의 애플리케이션을 광범위하게 테스트하라.
2. 여러분의 애플리케이션에 최종 사용자 라이센스 동의를 추가하는 것을 고려하라.
3. 애플리케이션의 매니페스트에 아이콘과 라벨을 지정하라.
4. 로깅과 디버깅을 끄고, 데이터/파일들을 정리하라.

여러분이 애플리케이션의 최종 컴파일을 하기 전에

5. 여러분의 애플리케이션에 버전을 부여하라.
6. 알맞은 암호 키key를 얻어라.

7. 여러분의 애플리케이션이 맵뷰MapView 엘리먼트를 사용하고 있다면, 맵 API 키key를 등록하라.

여러분의 애플리케이션 컴파일하기...

여러분의 애플리케이션을 컴파일한 후에

8. 여러분의 애플리케이션을 사인하라.
9. 여러분의 컴파일된 애플리케이션을 테스트하라.

애플리케이션 릴리즈 준비 이전 단계

1. 실제 디바이스에서 여러분의 애플리케이션을 광범위하게 테스트하라.

여러분의 애플리케이션을 가능한 많은 영역에서 가능한 광범위하게 테스트하는 것은 중요하다. 그렇게 하는 것을 돕기 위해 안드로이드는 다양한 테스트 클래스들과 도구들을 제공한다. 여러분은 JUnit와 다른 테스트 클래스를 실행하기 위해 인스트루멘테이션Instrumentation을 사용할 수 있다. 그리고 여러분은 UI/Application Exerciser 몽키Monkey와 같은 테스트 도구를 사용할 수 있다.

- 여러분의 애플리케이션이 사용자에게 적절하게 실행될 것인 지를 보장하기 위해서, 그 애플리케이션이 실행될 것으로 기대되는 하나 또는 그 이상의 실제 모바일 디바이스를 얻기 위한 모든 노력을 해야 한다. 그리고 나서 실제 네트워크 환경하에서 실제 디바이스 상에서 여러분의 애플리케이션을 테스트 해야 한다. 실제 디바이스에서 애플리케이션을 테스트하는 것은 무척 중요하다. 왜냐하면 그런 테스트는 여러분의 사용자 인터페이스 엘리먼트가 정확하게 크기를 가지고 있는 지(특히 터치 스크린 UI에서)와 애플리케이션 퍼포먼스와 배터리 효율이 적당한 지를 확인할 수 있도록 해주기 때문이다.
- 만약 여러분이 애플리케이션을 위한 대상이 되는 모바일 디바이스 타입을 얻을 수 없다면, -dpi, -device, -scale, -netspeed, -netdelay, -cpu-delay와 같은 에뮬레이터 옵션을 사용할 수 있으며, 그리고 가능한 최대로 타겟 디바이스에 맞도록 에뮬레이터 스크린, 네트워크 성능, 그리고 다른 애트리뷰트 값을 모델링 하기 위해 다른 것들을 사용할 수 있다. 그런 다음에 여러분

의 애플리케이션 UI와 성능을 테스트할 수 있다. 그러나 배포하기 전에, 여러분의 애플리케이션을 실제 타겟 디바이스에서 테스트할 것을 강력하게 권장한다.

- 만약 여러분의 애플리케이션 타겟이 T-Mobile G1 디바이스라면, 여러분의 UI가 스크린 오리엔테이션 변화를 처리하는 지를 확인해야 한다.

2. 여러분의 애플리케이션에 최종 사용자 라이센스 동의를 추가하는 것을 고려하라.

여러분의 개인, 조직, 그리고 지적 소유권을 보호하기 위해, 여러분은 애플리케이션에 최종 사용자 라이센스 동의(EULA, End User License Agreement)를 제공하길 원할 수도 있다.

3. 애플리케이션 매니페스트에 아이콘과 라벨을 지정하라.

애플리케이션의 매니페스트에 여러분이 지정한 아이콘과 라벨은, 그것들이 사용자에게 여러분의 애플리케이션 아이콘과 이름으로 표시된다는 점에서 중요하다. 아이콘과 라벨은 Manage Applicstions, My Downloads와 그 밖에 다른 곳뿐만아니라, 디바이스의 홈 스크린에 보여진다. 추가적으로, 배포 서비스도 사용자에게 그 아이콘과 라벨을 보여줄 수도 있다.

아이콘과 라벨을 지정하기 위해, 여러분은 매니페스트의 〈application〉 엘리먼트에 있는 android:icon과 android:label 애트리뷰트 값을 정의한다.

여러분의 아이콘 디자인을 고려할 때, 여러분은 그것이 내장되어 있는built-in 안드로이드 애플리케이션에 의해 사용된 스타일과 가능한 많이 일치되도록 만들도록 노력하라.

4. 로깅과 디버깅을 끄고 데이터/파일들을 정리하라.

릴리즈를 위해, 여러분은 디버그 설비들이 꺼졌는지와 그 디버그와 다른 불필요한 데이터/파일들이 여러분의 애플리케이션 프로젝트에서 제거되었는지를 확인해야 한다.

- 매니페스트의 〈application〉 엘리먼트에서 android:debuggable="true"를 제거하라.
- 로그 파일들, 백업 파일들, 그리고 다른 불필요한 파일들을 애플리케이션 프로젝트에서 제거하라.
- 사적 또는 소유권이 있는 데이터private or proprietary data를 확인해서 필요에 따라 제거하라.
- 소스 코드에 있는 모든 Log 메쏘드 호출을 비활성화시켜라.

애플리케이션의 최종 컴파일 이전 단계

5. 여러분의 애플리케이션에 버전 부여하라.

여러분은 애플리케이션을 컴파일하기 전에, 애플리케이션의 매니페스트 파일에 있는 〈manifest〉 엘리먼트의 android:versionCode와 android:versionName 애트리뷰트에 적절한 값을 지정하면서 여러분의 애플리케이션에 버전 숫자를 정의했는지 확인해야 한다. 여러분의 전체 애플리케이션 업그레이드 전략에서 버전 넘버링 계획을 주의깊게 고려하라.

만약 여러분의 애플리케이션을 이전에 배포했었다면, 여러분은 현재 애플리케이션의 버전 숫자를 증가시키는 것을 확인해야 한다. 여러분은 적절한 값을 사용해서 애플리케이션의 매니페스트 파일에 있는 〈manifest〉 엘리먼트의 android:versionCode와 android:versionName 애트리뷰트 모두를 증가시켜야 한다.

여러분의 애플리케이션의 버전 정보를 정의하는 방법에 대한 더 자세한 정보에 대해서 17장. "애플리케이션 버전 부여"를 보라.

6. 적합한 암호화 키(key) 얻어라.

만약 여러분이 지금까지 모든 준비 단계를 모두 읽고 따라 왔다면, 여러분의 애플리케이션은 컴파일 되었으며 사인을 하기 위한 준비가 되었다. 그 .apk 파일 내부에, 애플리케이션은 적절하게 버전화되어 있으며, 그리고 여러분은 위에 설명된 대로 추가 파일들과 사적인 데이터를 모두 말끔히 정리하였다.

여러분의 애플리케이션에 사인하기 이전에, 여러분은 알맞은 개인 키private key를 가지고 있는지를 확인해야 한다. 개인 키private key를 얻는(또는 생성하는) 방법에 대한 전체 정보를 위해서는 16장. 애플리케이션 사인하기의 공식 릴리즈를 위해 사인하기 중 적합한 개인 키 얻기를 보라.

일단 적합한 개인 키private key를 얻었다면(또는 생성했다면), 여러분은 그것을 다음과 같은 것을 하기 위해 사용할 것이다.

- 여러분의 애플리케이션이 맵뷰MapView 엘리먼트를 사용한다면, 맵 API 키Key를 등록하라(아래를 보라).
- 이후 준비과정에서 배포를 위해 애플리케이션을 사인하라.

7. 여러분의 애플리케이션이 MapView 엘리먼트를 사용하고 있다면, 맵 API Key를 등록하라.

만약 여러분의 애플리케이션이 하나 또는 그 이상의 맵뷰Mapview 엘리먼트를 사용하고 있다면, 여러분의 맵뷰MapView(s)가 구글 맵에서 데이터를 가져오기 전에 여러분의 애플리케이션을 구글 맵 서비스에 등록하고 맵 API 키Key를 얻을 필요가 있을 것이다. 그러기 위해서는 맵 서비스에 여러분의 사인 인증서의 MD5 fingerprint를 제공해야 한다.

> 맵 API Key를 얻는 것에 대한 전체 정보에 대해서는 Obtaining a Maps API Key(http://code.google.com/android/add-ons/google-apis/mapkey.html)를 보라.

개발하는 도중에, 여러분은 SDK 툴에 의해 생성된 디버그 키debug key를 등록함으로써, 임시 맵 API 키Key를 얻을 수 있다. 그러나 여러분의 애플리케이션을 배포하기 전에 여러분의 개인 키private key에 기반한 새로운 맵 API 키Key를 등록해야 한다.

만약 여러분의 애플리케이션이 맵뷰MapView 엘리먼트를 사용한다면, 이해해야 할 중요한 요점들은 다음과 같다.

1. 여러분은 릴리즈를 위한 여러분의 애플리케이션을 컴파일하기 전에 맵 API 키Key를 먼저 얻어야 한다. 왜냐하면 여러분의 애플리케이션 레이아웃 파일의 각각의 맵뷰MapView 엘리먼트의 특정 애트리뷰트인 android:apiKey에 키Key를 추가해야 하기 때문이다. 만약 여러분이 맵뷰MapView 오브젝트를 코드에서 직접 인스턴스화 한다면, 맵 API 키Key를 생성자의 파라미터로 전달해야 한다.

2. 여러분은 애플리케이션의 맵뷰MapView 엘리먼트에 의해 레퍼런스되는 맵 API 키Key는 (구글 맵에서) 애플리케이션을 사인하기 위해 사용되는 인증서에 등록되어야 한다. 이것은 특히 여러분의 애플리케이션을 배포할 때 중요하다. 여러분의 맵뷰MapView 엘리먼트들은 여러분이 애플리케이션을 사인하기 위해 사용할 릴리즈 인증서에 등록된 키Key를 레퍼런스해야 한다.

3. 만약 여러분이 SDK 툴tool에 의해 생성된 디버그 인증서를 등록함으로써 임시 맵 API 키Key를 이전에 얻었다면, 여러분의 릴리즈 인증서를 등록해서 새로운 맵 API 키Key를 얻어야 한다는 것을 기억하라. 여러분은 맵뷰MapView 엘리먼트를 디버그 인증서와 결합된 키Key가 아니라 새로운 키Key를 레퍼런스하도록 변경시켜야하는 것을 기억해야 한다. 만약 그렇게 하지 않으면, 여러분의 맵뷰MapView 엘리먼트는 맵 데이터를 다운로드하지 못할 것이다.

4. 여러분의 애플리케이션을 사인하기 위해 사용할 개인 키private key를 바꾸고자 한다면, 구글 맵 서비스에서 새로운 맵 API 키Key를 얻어야 한다는 것을 기억해야 한다. 새로운 맵 API 키Key를 얻어서 모든 맵뷰MapView 엘리먼트에 적용하지 않는다면, 예전의 키Key를 레퍼런스하는 맵뷰MapView 엘리먼트는 맵 데이터를 다운받지 못할 것이다.

여러분의 애플리케이션을 컴파일하라.

이전 섹션에 설명된 대로 애플리케이션을 준비했다면, 여러분은 배포를 위해 애플리케이션을 컴파일 할 수 있다.

애플리케이션 컴파일 이후 단계

8. 여러분의 애플리케이션을 사인하라.

여러분의 개인 키private key를 사용해서 여러분의 애플리케이션을 사인하고 그것을 zipalign 도구를 이용하여 얼라인하라. 여러분의 애플리케이션에 정확하게 사인하는 것은 매우 중요하다. 전체 정보를 보려면 16장. "애플리케이션 사인하기"를 보라.

9. 컴파일되고 사인된 여러분의 애플리케이션을 테스트하라.

여러분의 컴파일된 애플리케이션을 배포하기 전에, 여러분을 그것을 타겟 모바일 디바이스 상에서 (그리고 가능하다면 타겟 네트워크상에서) 광범위하게 테스트해야 한다. 특히, 여러분의 UI에 있는 임의의 맵뷰MapView 엘리먼트들이 적절하게 맵 데이터를 받는 지를 확인해야 한다. 만약 그렇지 않다면, Register for a Maps API Key로 돌아가서 문제점을 수정하라. 여러분은 또한 그 애플리케이션이 여러분이 제공하고 신뢰하는 임의의 서버쪽 서비스와 데이터를 가지고 정확하게 동작하는 지와 그 애플리케이션이 임의의 인증 요구사항을 정확하게 처리하는 지를 확인해야 한다.

테스트를 한 후, 여러분은 이제 여러분의 애플리케이션을 모바일 디바이스 사용자들에게 배포할 준비가 되어 있다.

Section | 19

애플리케이션 배포하기

19.1. 안드로이드 마켓에 배포하기

19. 애플리케이션 배포하기

애플리케이션을 배포한다는 것은 애플리케이션을 적절하게 테스트하고 패키지하며 그것을 안드로이드 모바일 디바이스 사용자들에게 다운로드 또는 사이드로드sideload 할 수 있도록 만드는 것을 의미한다.

만약 여러분이 18장. "배포를 위한 준비"에서 기술된 단계들을 따랐다면, 그 결과는 여러분의 릴리즈 개인 키private key를 가지고 사인된 컴파일된 .apk 파일이다. 그 .apk 내부에서, 애플리케이션은 정확하게 버전화되어 있으며 임의의 맵뷰MapView 엘리먼트들은 .apk를 사인하기 위해 사용된 것과 동일한 인증서의 MD5 fingerprint를 등록하여 얻은 맵 API 키Key를 레퍼런스한다. 여러분의 애플리케이션은 이제 배포할 준비가 되었다.

아래의 섹션들은 모바일 디바이스 사용자들에게 여러분의 안드로이드 애플리케이션을 배포하는 것에 관한 정보를 제공한다.

안드로이드 마켓에 배포하기

안드로이드 마켓은 사용자가 자신들의 안드로이드 디바이스에 안드로이드 애플리케이션을 쉽게 찾아서 다운로드 할 수 있고 개발자들에게는 안드로이드 사용자들에게 애플리케이션을 쉽게 배포할 수 있는 호스트된 서비스이다.

안드로이드 마켓에 여러분의 애플리케이션을 배포하고자 한다면, 여러분은 먼저 구글 계정을 사용해서 서비스에 등록하고 서비스 규정에 동의해야 한다. 일단 여러분이 마켓에 등록되면, 원할 때면 언제라도 원하는 횟수만큼 서비스에 애플리케이션을 업로드할 수 있으며, 그런 다음에 여러분이 준비될 때 그것을 배포할 수 있다. 일단 배포가 되면, 사용자들은 그들의 안드로이드 디바이스에 설치된 마켓 애플리케이션을 사용해서 애플리케이션을 보고, 다운로드 하고, 평가할 수 있다.

안드로이드 마켓 개발자로 등록하고 배포를 하려고 한다면, 아래의 안드로이드 마켓(Android Marekt)을 방문하라.

```
http://market.android.com/publish
```

만약 여러분이 안드로이드 마켓에 여러분의 애플리케이션을 배포할 계획이라면, 여러분의 애플리케이션이 아래에 나열된 요구 사항들을 만족시키는 지를 확인해야 하며 이 요구사항들은 여러분이 애플리케이션을 업로드할 때 마켓 서버에 의해 강제된다enforced.

안드로이드 마켓 서버에 의해 강제되는 요구사항.

1. 여러분의 애플리케이션은 유효 기간이 2033년 10월 22일 이후에 끝나는 암호화된 개인 키private key로 사인되어야 한다.
2. 여러분의 애플리케이션은 매니페스트의 〈manifest〉 엘리먼트에 있는 android:versionCode와 android:versionName 애트리뷰트 모두를 정의해야 한다. 서버는 애플리케이션을 내부적으로 식별하고 업그레이드를 처리하기 위한 근거로 android:versionCode를 사용하고 애플리케이션 버전으로써 사용자에게 android:versionName을 보여준다.
3. 여러분의 애플리케이션은 매니페스트의 〈application〉 엘리먼트에 있는 android:icon과 android:label 애트리뷰트 모두를 정의해야 한다.

안드로이드 마켓에 업데이트 배포하기

안드로이드 마켓에 애플리케이션을 배포한 이후 언제든지, 여러분은 동일 애플리케이션에 대한 업데이트를 업로드하고 배포할 수 있다. 여러분이 애플리케이션에 대한 업데이트를 배포할 때, 그 애플리케이션을 이미 설치했던 사용자는 해당 애플리케이션에 대한 업데이트가 유용하다는 공지를 자동으로 수신할 것이다. 그 때 사용자는 해당 애플리케이션을 최종 버전으로 업데이트하는 것을 선택할 수 있다.

업데이트된 애플리케이션을 업로드하기 전에, 매니페스트 파일의 〈manifest〉 에리멘트에 있는 android:versionCode와 android:versionName 애트리뷰트

들이 증가되었는지를 확실하게 하라. 또한 패키지 이름이 동일해야 하면, .apk는 동일한 개인 키로 사인되어야 한다. 만약 패키지 이름과 사인하는 인증서가 이미 존재하는 버전의 것들과 일치하지 않는다면, 마켓은 그것을 신규 애플리케이션으로 간주할 것이고, 사용자에게 업데이트 형태로 그것을 제안하지는 않을 것이다.

디바이스 상의 마켓 애플리케이션 런치를 위한 인텐트 사용

안드로이드 디바이스들은 사용자가 안드로이드 마켓 사이트에 접속할 수 있도록 이미 설치된 마켓Market 애플리케이션을 포함하고 있다. 마켓Market으로 부터, 사용자는 가능한 애플리케이션을 브라우징하거나 찾고 사용자 평가와 리뷰를 읽고 애플리케이션을 다운로드/설치할 수 있다.

여러분은 시스템에 인텐트를 전송함으로써 다른 안드로이드 애플리케이션에서 마켓 애플리케이션을 런치할 수 있다. 예를 들어 여러분은 사용자가 이미 설치된 애플리케이션에 대한 업그레이드 위치와 다운로드하는 것을 돕기 위해, 또는 사용자가 다운로드 가능한 관련된 애플리케이션에 대해서 알게 하기 위하여 이러한 것을 하고 싶을 수도 있다.

마켓을 런치하기 위해서, 여러분은 인텐트 데이터로 마켓이 취급하는 URI 문자열을 전달하는ACTION_VIEW 인텐트를 전송한다. 대부분의 경우에, 여러분의 애플리케이션은 마켓이 취급하는 URI를 가지고 ACTION_VIEW 인텐트를 전송하기 위해 startActivity()를 호출한다.

인텐트를 사용해서 여러분이 제공하는 URI는 시스템이 적절하게 인텐트를 인도하게route 하며, 또한 여러분이 마켓이 런치 후에 수행하길 원하는 액션의 유형을 설명한다. 현재, 여러분은 마켓이 다음과 같은 액션을 하게 할 수 있다.

- 여러분이 제공하는 쿼리 파라미터에 기반해서 마켓 애플리케이션이 안드로이드 마켓 상의 애플리케이션을 검색한다. 또는
- 애플리케이션 패키지 이름에 기반해서 안드로이드 마켓상의 특정 애플리케이션에 대한 상세한 페이지를 가져온다.

검색하기(Initiating a search)

여러분은 안드로이드 마켓에서 여러분이 제공하는 쿼리 파라미터와 일치하는 애플리케이션을 검색할 수 있다. 그렇게 하기 위해서, 여러분의 애플리케이션은 다음과 같은 포맷으로 된 URI와 쿼리 파라미터를 포함하는 ACTION_VIEW 인텐트를 발송해야 한다.

```
market://serarch?q=<paramtype>:<value>
```

이 URI 포맷을 사용해서 여러분은 다음과 같은 것으로 애플리케이션을 검색할 수 있다.

- 패키지 이름
- 개발자 이름
- 애플리케이션 이름, 개발자 이름, 설명 등과 일치되는 문자열
- 위의 것들에 대한 임의의 조합

이 페이지의 하단에 있는 테이블은 이러한 각각의 검색 유형에 대응되는 paramtype들과 value들을 명세한다.

안드로이드 마켓상의 검색들

여러분이 검색을 시작할 때, 안드로이드 마켓은 안드로이드 마켓 프로파일 또는 애플리케이션 배포 정보에 있는 개발자에 의해 제공된 공개 메타데이터에서 일치하는 결과들을 반환한다. 하지만 개발자의 사적 계정 또는 애플리케이션을 사인하는데 사용된 인증서에서는 검색하지 않는다.

여러분이 애플리케이션을 검색하기 위한 인텐트를 발송할 때, 마켓은 서버에 검색 쿼리를 전송하고 그 결과를 보여준다. 사용자에게 있어서, 이러한 경험은 다음과 같은 것들이다.

1. 사용자는 여러분의 애플리케이션에서 링크 또는 버튼을 누른다.
2. 마켓 애플리케이션이 시작되며, 그것이 검색 결과들을 수신할 때까지 "Searching" 라벨이 붙은 진행상태 지시자를 보여주면서 스크린에 대한 컨트롤을 점유한다.
3. 마켓은 검색 결과를 수신하며 그것들을 보여준다. 쿼리 파라미터에 기반해서,

검색 결과는 하나 또는 그 이상의 애플리케이션의 목록을 포함할 수 있다.

4. 결과 페이지에서, 사용자는 애플리케이션의 상세 페이지로 가기 위해 그것을 선택할 수 있다. 그것은 그 애플리케이션에 대한 정보를 제공하며 사용자가 그 애플리케이션을 다운로드/구매하는 것을 가능하게 한다.

애플리케이션의 상세 페이지 가져오기(Loading)

안드로이드 마켓에서, 모든 애플리케이션은 사용자에게 애플리케이션의 개요를 제공하는 상세 페이지를 갖는다. 예를들어, 그 페이지는 애플리케이션에 대한 간단한 설명과 그것이 사용될 때의 스크린 샷들을 포함한다. 만약 개발자에 의해 제공된다면, 사용자에 대한 피드백과 개발자에 대한 정보 또한 포함한다. 또한 그 상세 페이지는 사용자가 애플리케이션을 다운로드/구매를 시작할 수 있게하는 "install" 버튼을 포함한다.

만약 여러분이 사용자가 특정 애플리케이션을 참조하길 원한다면, 여러분의 애플리케이션은 사용자를 직접적으로 그 애플리케이션의 상세 페이지로 가도록 할 수 있다. 그렇게 하기 위해, 여러분의 애플리케이션은 다음과 같은 포맷으로 된 URI와 쿼리 파라미터를 포함하는 ACTION_VIEW 인텐트를 발송한다.

```
market://detail?id=<packagename>
```

이 경우에, packagename 파라미터는 그 애플리케이션의 매니페스트 파일에 있는 manifest 엘리먼트의 package 애트리뷰트에서 선언된 것과 같은 타겟 애플리케이션의 전체 패키지 이름이다. 예를 들어 다음과 같다.

```
market://detail?id=com.example.android.jetboy
```

안드로이드 마켓 URI들

아래의 테이블은 현재 마켓Market 애플리케이션에 의해 지원되는 URI와 액션 목록이다.

이 URI들은 오직 인텐트 데이터data로 전달될 때만 동작한다는 것에 주의하라 – 여러분은 데스크탑 머신 또는 디바이스 상의 웹 브라우저에서 현재 그 URI를 로드할 수 없다.

이 결과를 위해서는.	ACTION_VIEW 인텐트와 함께 이 URI를 전달하라.	비고
애플리케이션의 매니페스트에 정의된, 그 애플리케이션의 전체 패키지 이름으로 식별되는 특정 애플리케이션에 대한 상세 스크린을 보여준다.	http://market.android.com/detail?id=〈패키지이름〉 또는 market://detail?id=〈패키지이름〉	여러분이 지정한 패키지 이름은 임의의 애플리케이션 버전에 국한되지 않는다. 그러므로, 마켓은 항상 그 애플리케이션의 최종 버전에 대한 상세 페이지를 보여준다.
전체 Java 패키지 이름으로 애플리케이션을 검색해서, 그 결과를 보여준다.	http://market.android.com/search?q=pname:〈패키지〉 또는 market://search?q=pname:〈패키지〉	애플리케이션의 자바 패키지 이름만 검색한다. 정확하게 일치하는 것만 반환한다.
개발자 이름으로 애플리케이션을 검색해서, 그 결과를 보여준다	http://market.android.com/search?q=pub:"〈개발자 이름〉" 또는 market://search?q=pub:"〈개발자 이름〉"	마켓 공개 프로파일의 "개발자 이름"만 검색한다. 정확하게 일치하는 것만 반환한다.
부분 문자열로 애플리케이션을 검색해서, 그 결과를 보여준다	http://market.android.com/search?q=〈하위문자열〉 또는 market://search?q=〈하우문자열〉	모든 애플리케이션에 대한 공개필드(애플리케이션 제목, 개발자 이름, 그리고 애플리케이션 설명)를 검색한다. 정확하게 일치하는 것만 반환한다.
여러 개의 쿼리 파라미터를 사용해서 검색하고, 그 결과를 보여준다	예제 : http://market.android.com/search?q=world pname:com.android.hello pub:Android	제공되는 모든 파라미터에 맞는 애플리케이션 목록을 리턴한다.

Section 20

성능을 위한 설계

20.1. 소개

20.2. 오브젝트 생성을 피하라

20.3. 네이티브(Native) 메쏘드를 사용하라

20.4. 인터페이스(Interface) 보다 버추얼(Virtual)을 선호하라

20.5. 버추얼(Virtual)보다 정적(Static)을 선호하라

20.6. 내부적인 Getters/Setters를 피하라

20.7. 캐쉬 필드 룩업(Lookup)

20.8. 상수를 Final로 선언하라

20.9. Enhanced For Loop 문법을 주의하여 사용하라

20.10. Enums을 피하라

20.11. 내부 클래스를 가지는 패지지 영역(Scope)을 사용하라

20.12. 부동소수점을 피하라

20.13. 약간의 샘플 성능 숫자들

20.14. 마무리 노트

20. 성능을 위한 설계

안드로이드 애플리케이션은 빨라야 한다. 아마도 그것은 안드로이드 애플리케이션이 효율적이어야 한다고 말하는 것이 더 정확할 것이다. 즉, 그것은 제한된 컴퓨팅 파워와 데이터 저장 공간, 더 작은 스크린, 그리고 한정적인 배터리 수명을 가진 모바일 디바이스 환경에서 가능한 한 효율적으로 실행되여야 한다.

여러분이 애플리케이션을 개발할 때, 여러분의 듀얼 코어 개발 컴퓨터에서 실행되는 여러분의 에뮬레이터에서는 그 애플리케이션이 충분히 잘 동작할지라도, 그것이 모바일 디바이스에서 실행될 때는 그렇게 잘 동작하지 않을 지도 모른다 – 심지어 가장 강력한 모바일 디바이스라도 보통의 데스크탑 시스템의 성능과 일치할 수는 없다. 이런 이유로, 여러분은 다양한 모바일 디바이스에서 가능한 최고의 성능을 낼 수 있도록 효율적인 코드를 작성하기 위해 노력해야 한다.

일반적으로 말해서, 빠르거나 효율적인 코드를 작성한다는 것은 최소한의 메모리 할당을 유지하고, 타이트한tight 코드를 작성하고, 그리고 미묘하게 성능이 떨어질 수 있는 어떤 언어와 프로그래밍 문구를 피하는 것이다. 객체지향적인object-oriented 조건에서, 이러한 대부분의 작업은 메쏘드 레벨에서 코드, 루프 등의 실제 라인들의 순서대로 일어난다.

이 문서는 아래의 토픽을 다룬다.

- 소개
- 오브젝트를 생성하는 것을 피하라.
- 네이티브 메쏘드들을 사용하라.
- 인터페이스(Interface)보다 버추얼(Virtual)을 선호하라.
- 버추얼(Virtual) 보다 정적(Static)을 선호하라.
- 내부적인 Getters/Setters를 피하라.
- 캐쉬 필드 룩업(Lookup)

- 상수를 Final로 선언하라.
- Enhanced For Loop 문법을 주의하여 사용하라.
- Enums를 피하라.
- 내부 클래스를 가지는 패키지 범주(Scope)를 사용하라.
- 부동소수점을 피하라.
- 약간의 샘플 성능 숫자들.
- 마무리 노트.

소개

리소스가 제한된 시스템에는 두 가지의 기본적 규칙이 있다.

- 여러분이 필요로 하지 않는 작업을 하지 마라.
- 만약 여러분이 피할 수 있다면, 메모리를 할당하지 마라.

아래에 적힌 모든 팁들은 이러한 두 가지의 기본적인 원칙으로부터 나온다.

어떤 사람들은 이 페이지에 적힌 대부분의 조언들이 "성급한 최적화premature optimization"에 해당한다고 주장할 수도 있다. 마이크로한 최적화micro-optimization가 때때로 효율적인 데이터 구조와 알고리즘을 개발하기 더 어렵게 만든다는 것이 사실임에도 불구하고, 핸드폰과 같이 임베디드 디바이스에서는 종종 선택의 여지가 없게 된다. 예를 들어 데스크탑 머신에서의 가상머신 성능에 대한 가정을 안드로이드에 가지고 온다면 시스템 메모리를 소진시키는 코드를 작성하게 될 가능성이 크다. 이것은 시스템에서 실행되는 다른 프로그램은 말할 것도 없이 여러분의 애플리케이션이 천천히 실행되도록 할 것이다.

위에서 언급한 것은 이 가이드라인이 중요한 이유이다. 안드로이드의 성공은 여러분의 애플리케이션이 제공하는 사용자 경험user experience에 달려있고 사용자 경험user experience은 여러분의 코드가 재빨리 반응하는 지 또는 느린 지에 부분적으로 의존한다. 우리의 모든 애플리케이션이 하나의 동일한 디바이스에서 실행될 것이기 때문에, 한편으로 우리는 이 문제에 모두 함께 속해 있다. 이 문서를 운전면허를 취득할 때 배워야 했던 도로에 대한 규칙처럼 생각해보라: 모든 사람이 그

것을 따른다면 일들이 매끄럽게 처리될 것이지만, 그렇지 않는다면 여러분의 차는 부서질smashed up 것이다.

우리가 실제 문제를 다루기 전에 간략한 주의 사항이 있다: 아래에 설명된 거의 모든 이슈들은 가상머신이 JIT(Just-In Time) 컴파일러 특징을 가지는지 여부와 상관없이 유효하다. 만약 내가 동일한 것을 달성하기 위한 두 가지 메쏘드를 가지고 있고, foo()에 대한 인터프리티드interpreted 실행이 bar()보다 더 빠르다면, 그러면 foo()의 컴파일된 버전은 bar()의 컴파일된 버전 만큼 빠르거나 또는 더 빠를 것이다. 컴파일러에 의존해서 여러분의 코드를 충분히 빠르게 할 수 있다고 생각하는 것은 현명하지 않다.

오브젝트 생성을 피하라

오브젝트를 생성하는 것은 결코 자유롭지 않다. 임시적인 오브젝트에 대한 쓰레드별 할당 풀per-thread allocation pools을 사용하는 generational GC(Garbage Collection)은 할당을 더 값싸게 할 수 있다. 그러나 메모리를 할당하는 것은 메모리를 할당하지 않는 것보다 항상 더 값 비싸다.

여러분이 사용자 인터페이스 루프에 오브젝트를 할당한다면, 여러분은 사용자 인터페이스 내에 작은 "끊김hiccups"을 만드는 주기적인 가비지 콜렉션(GC, Garbage Collection)을 야기할 것이다.

따라서, 여러분은 필요하지 않은 오브젝트 인스턴스를 생성하는 것을 피해야 한다. 다음과 같은 약간의 예제들이 도움을 줄 것이다.

- 입력 데이터 집합set으로로부터 문자열을 추출할 때, 복사본을 만드는 대신에 원래 데이터의 하위 문자열substring을 리턴하라. 여러분은 새로운 String 오브젝트를 만들 것이다. 그러나 그것은 그 char[]를 데이터와 공유할 것이다.
- 만약 여러분이 문자열을 리턴하는 메쏘드를 가지고 있고, 그리고 여러분이 그 결과가 항상 StringBuffer에 어떻게든 추가될 것을 알고 있다면, 짧게 생존하는 임시 오브젝트를 생성하는 대신에 그 함수가 직접적으로 추가하도록 여러분의 시그니쳐signature와 구현implementation을 변경하라.

다소 극단적인 아이디어는 다차원의 배열array를 병렬적인 일차원 배열들로 조각내는 것이다.

- int 배열은 integer 배열보다 훨씬 더 낫다. 그러나, 이것은 또한 두 개의 병렬 int 배열이 하나의 (int,int) 오브젝트의 배열보다 훨씬 더 효과적이라는 사실로 일반화된다. 이와 같은 것은 프리미티브 타입primitive type들의 임의의 조합에도 적용된다.
- 여러분이 (Foo,Bar) 오브젝트 집합들을 저장하는 컨테이너를 구현할 필요가 있다면, 두 개의 병렬 Foo[]와 Bar[] 배열이 일반적으로 하나의 (Foo,Bar) 오브젝트 배열보다 더 낫다는 점을 기억하라(물론, 여기에 예외는 여러분이 다른 코드가 접근할 수 있도록 하는 API를 디자인할 때이다. 이런 경우에, 보통 정확한 API 디자인보다 속도 면에서 약간의 향상이 더 낫다. 그러나 여러분 자신의 내부 코드에서, 여러분은 가능한 한 효율적으로 되도록 노력해야만 한다).

일반적으로 이야기해서, 가능하다면 단기적인 임시 오브젝트를 생성하는 것을 피하라. 더 적게 생성된 오브젝트는 더 작은 빈도의 가비지 콜렉션(GC, Garbage Collection)을 의미하며, 이것은 사용자 경험user experience에 직접적인 영향을 준다.

네이티브(Native) 메쏘드를 사용하라

문자열을 처리할 때, String.indexOf(), String.lastindexOf(), 그리고 그와 유사한 것들과 같은 전문speciality 메쏘드를 사용하는데 주저하지 마라. 이러한 것들은 보통 Java 루프에서 동일한 것을 하는 것 보다 10배에서 100배 더 빨리 쉽게 실행되는 C/C++ 코드로 구현되어 있다.

이 조언의 이면에는 네이티브 메쏘드를 통한 입력punching이 인터프리티드 메쏘드를 호출하는 것보다 더 비싸다는 것이 있다. 만약 여러분이 그것을 피할 수 있다면, 사소한 연산에는 네이티브 메쏘드를 사용하지 마라.

인터페이스(Interface)보다 버추얼(Virtual)을 선호하라

여러분이 HashMap 오브젝트를 가지고 있다고 가정하자. 여러분은 그것을 HashMap 또는 일반적인 Map으로 선언할 수 있다.

```
Map myMap1 = new HashMap();
HashMap myMap2 = new HashMap();
```

어떤 쪽이 더 나은가?

전통적인 지식은 여러분이 Map을 더 선호해야 한다고 말한다. 왜냐하면 그것은 여러분으로 하여금 기본 구현을 Map 인터페이스 구현으로 변경하는 것을 허용하기 때문이다. 전통적인 프로그래밍에 대해서서는 전통적 지식이 맞다. 그러나, 임베디드 시스템에 대해서는 그렇지 않다. 인터페이스 레퍼런스를 통해 호출하는 것은 구체적인 레퍼런스를 통해 가상 메쏘드를 호출하는 것 보다 두 배 더 오래 걸릴 수 있다.

만약 여러분이 하고 있는 것에 잘 맞기 때문에 HashMap을 선택했다면, 그것을 Map이라고 부르는 것에는 거의 의미가 없다. 여러분을 위해 여러분의 코드를 재분해하는refactor IDEs의 유효성 하에서 여러분이 그 코드가 향하는 곳이 어디인지를 확실히 모름에도 불구하고 그것을 Map이라고 부르는 것에는 별로 의미가 없다(다시말해, public APIs는 예외이다: 좋은 API는 대개 사소한 성능 우려사항을 무시한다).

버추얼(Virtual)보다 정적(Static)을 선호하라

만약 여러분이 오브젝트의 필드에 접근할 필요가 없다면, 여러분의 메쏘드를 정적static으로 만들어라. 그것은 더 빠르게 호출될 수 있다. 왜냐하면 그것이 버추얼 메쏘드 테이블 인다이렉션indirection을 요구하지 않기 때문이다. 그것은 또한 좋은 사례이다. 왜냐하면 여러분은 그 정적인 메쏘드를 호출하는 것이 오브젝트 상태를 변경할 수 없다는 것을 그 정적 메쏘드 시그니처signature로부터 알 수 있기 때문이다.

내부적인 Getters/Setters를 피하라

C++와 같은 네이티브 언어들에서 필드를 직접 접근(i = mCount)하는 것 대신에 getters(i = getCount())를 사용하는 것은 일반적인 사례이다. 이것은 컴파일러가 보통 그 접근을 인라인inline할 수 있기 때문에, C++을 위해서는 아주 훌륭한 방법이다. 그리고 만약 여러분이 필드 접근을 제한하거나 디버그하는 것이 필요하다면 여러분은 언제라도 그 코드를 추가할 수 있다.

안드로이드에서 이것은 나쁜 생각이다. 버추얼 메쏘드 호출은 인스턴스 필드 룩업lookup 보다 훨씬 더 값비싸다. 일반적인 객체지향object-oriented 프로그래밍 사례를 따르고 퍼블릭public 인터페이스에서 getter와 setter를 가지는 것은 합리적이다. 그러나 하나의 클래스 내에서 여러분은 항상 필드들을 직접 접근해야 한다.

캐쉬 필드 룩업(Lookups)

오브젝트 필드에 접근하는 것이 지역 변수에 접근하는 것보다 훨씬 느리다. 다음과 같이 작성하는 것 대신에

```
for (int i = 0; i < this.mCount; i++)
      dumpItem(this.mItems[i]);
```

여러분은 이렇게 작성해야 한다.

```
int count = this.mCount;
Item[] items = this.mItems;

for (int i = 0; i < count; i++)
      dumpItems(items[i]);
```

(우리는 이것들이 멤버 변수들이라는 것을 명료하게 하기 위해 명시적인 "this"을 사용하고 있다)

비슷한 가이드라인은 "for" 구문의 두 번째 절에서 메쏘드를 결코 호출하지 않는 것이다. 예를 들어 아래의 코드는 반복iteration될 때마다 한 번씩 getCount() 메쏘드를 실행할 것이다. 그것은 엄청난 낭비이다. 여러분은 그 값을 int로 간단히 캐쉬할 수 있었다.

```
for (int i = 0; i < this.getCount(); i++)
    dumpItems(this.getItem(i));
```

만약 여러분이 한번 이상 인스턴스에 접근하려고 한다면 지역 변수를 생성하는 것은 또한 보통 좋은 생각이다. 예를 들어 다음과 같다.

```
protected void drawHorizontalScrollBar(Canvas canvas,
                                int width, int height) {
   if (isHorizontalScrollBarEnabled()) {
        int size = mScrollBar.getSize(false);
        if (size <= 0) {
```

```
                size = mScrollBarSize;
            }
            mScrollBar.setBounds(0, height - size, width, height);
            mScrollBar.setParams(
                        computeHorizontalScrollRange(),
                        computeHorizontalScrollOffset(),
                        computeHorizontalScrollExtent(), false);
            mScrollBar.draw(canvas);
        }
    }
```

위의 것은 멤버 필드 mScrollBar의 네 가지의 별도의 룩업lookup이다. 로컬 스택 변수에 mScrollBar를 캐쉬함으로써 네 가지 멤버 필드 룩업lookup은 스택 변수 레퍼런스가 되고, 그것이 훨씬 더 효율적이다.

덧붙여 말하자면, 메쏘드 아규먼트들은 로컬 변수와 같은 동일한 성능 특징을 가진다.

상수(Constants)를 Final로 선언하라

클래스의 맨 위에 있는 다음과 같은 선언을 생각해 보라.

```
static int intVal = 42;
static String strVal = "Hello, world!";
```

컴파일러는 〈clinit〉라고 불리는 클래스 이니셜라이저initializer 메쏘드를 생성한다. 클래스 이니셜라이저는 클래스가 처음으로 사용될 때 실행된다. 이 메쏘드는 intVal에 값 42를 저장한다. 그리고 strVal을 위한 클래스 파일 문자열 상수 테이블로부터 하나의 레퍼런스를 추출한다. 이러한 값들이 나중에 레퍼런스될 때면, 그것들은 필드 룩업을 사용해서 접근된다.

우리는 "final" 키워드를 사용해서, 이 문제들을 개선할 수 있다.

```
static final int intVal = 42;
static final String strVal = "Hello, world!";
```

그 클래스는 더 이상 〈clinit〉 메쏘드를 요구하지 않는다. 왜냐하면 그 상수들은 클래스 파일 정적static 필드 이니셜라이저initializers 안으로 가고, 가상머신에 의

해 직접적으로 처리되기 때문이다. intVal을 접근하는 코드는 정수 값 42를 직접 사용할 것이다. 그리고 strVal에 대한 접근들은 필드 룩업lookup 대신에 상대적으로 값싼 "string constant" 명령어를 사용할 것이다.

메쏘드 또는 클래스 "final"을 선언하는 것이 즉각적인 성능 잇점을 주지는 않는다. 그러나 그것은 어느 정도의 최적화를 허용한다. 예를 들어 만약 컴파일러가 "getter" 메쏘드는 서브클래스에 의해 오버라이드 될 수 없다는 것을 안다면, 그 컴파일러는 메쏘드 호출을 인라인inline할 수 있다.

여러분은 또한 지역 변수를 final로 선언할 수 있다. 하지만 이것은 명확한 성능 잇점을 가지지 않는다. 지역 변수에 대해서, "final"이 그 코드를 더 명료하게 만든다면 그것을 사용하라(또는 예를 들어 익명의 내부(inner) 클래스에서 final을 사용하라).

Enhanced For Loop 문법을 주의하여 사용하라

(때때로 "for-each" loop로 알려져 있는) enhanced for loop는 Iterable interface를 구현하는 콜렉션을 위해 사용될 수 있다. 이러한 오브젝트 콜렉션을 가지고 iterator는 hasNext()와 next()에 대한 인터페이스 호출을 위해서 할당된다. ArrayList로 여러분은 그 콜렉션을 직접 더 잘 다룰 수 있다. 그러나 다른 콜렉션에 대해서 enhanced for loop 문법은 명시적인 iterator 사용법과 같다.

그럼에도 불구하고 다음의 코드는 enhanced for loop의 수용가능한 사용 예제를 보여준다.

```
public class Foo {
    int mSplat;
    static Foo mArray[] = new Foo[27];

    public static void zero() {
        int sum = 0;
        for (int i = 0; i < mArray.length; i++) {
            sum += mArray[i].mSplat;
        }
    }

    public static void one() {
```

```
        int sum = 0;
        Foo[] localArray = mArray;
        int len = localArray.length;

        for (int i = 0; i < len; i++) {
            sum += localArray[i].mSplat;
        }
    }

    public static void two() {
        int sum = 0;
        for (Foo a: mArray) {
            sum += a.mSplat;
        }
    }
}
```

zero()는 정적 필드static field를 두 번 얻어오고 루프를 통해 반복할 때마다 배열 길이array length를 얻는다.

one()은 모든 것을 지역 변수local variables로 바꾸고 검색lookup을 피하게 한다.

two()는 Java 프로그래밍 언어 버전 1.5에서 소개된 enhanced for loop 문법을 사용한다. 컴파일러에 의해 발생한 이 코드는 배열 레퍼런스array reference와 배열 길이array length를 지역 변수local variables로 복사하며, 이것은 배열의 모든 구성요소를 다루는 데 좋은 선택이 되게 한다. 이것은 (분명히 "a"를 보존하는) main loop에서 추가적인 지역적인 로드/저장local load/store를 발생시키고 one()보다는 약간 느리고 4 바이트 길어진다.

이 모든 것을 좀 더 분명하게 요약하자면: enhanced for loop 문법은 배열arrays와 함께 잘 동작하지만 그것을 Iterable(반복) 오브젝트와 함께 사용할 때면 추가적인 오브젝트 생성이 있으므로 주의하여야 한다.

Enums을 피하라

Enums는 매우 편리하다. 그러나 불행하게도 크기와 속도가 문제가 될 때 힘들어질 수 있다. 예를 들어,

```
public class Foo {
   public enum Shrubbery { GROUND, CRAWLING, HANGING }
}
```

위의 문장은 900 바이트의 .class 파일 (Foo$Shrubbery.class)로 바뀐다. 처음 사용할 때, 클래스 이니셜라이저(initialize)는 열거된 각각의 값들을 나타내는 오브젝트들에 대한 〈init〉 메쏘드를 호출한다. 각각의 오브젝트는 그 자신의 정적static 필드를 얻는다. 그리고 그 전체 집합은 하나의 배열("$VALUES"로 불리는 정적 필드)에 저장된다. 이것은 단지 세 개의 정수integer에 대해서는 많은 양의 코드와 데이터이다.

```
Shrubbery shrub = Shrubbery.GROUND;
```

위의 문장은 정적(static) 필드 룩업을 초래한다. 만약 "GROUND"가 static final int라면 컴파일러는 그것을 하나의 알려진 상수로 취급할 것이고, 그것을 인라인inline할 것이다.

물론 이면에는, 여러분이 enums를 사용해서 더 나은 API들과 약간의 컴파일 시점의 값 체크를 얻는다. 그러므로 통상적인 트레이드오프trade-off가 적용된다: 여러분은 반드시 public API에서는 enums를 사용해야 한다. 그러나 성능 문제가 있을 때는 enum 사용을 피하도록 하라.

어떤 환경에서는 그것은 ordinal() 메쏘드를 통해서 enum 정수integer 값을 얻는 것이 도움이 될 수 있다. 예를 들어 아래의 코드를

```
for (int n = 0; n < list.size(); n++) {
    if (list.items[n].e == MyEnum.VAL_X)
       // do stuff 1
    else if (list.items[n].e == MyEnum.VAL_Y)
       // do stuff 2
}
```

다음과 같이 바꾸어라.

```
int valX = MyEnum.VAL_X.ordinal();
int valY = MyEnum.VAL_Y.ordinal();
int count = list.size();
```

```
MyItem items = list.items();

for (int  n = 0; n < count; n++)
{
     int  valItem = items[n].e.ordinal();

     if (valItem == valX)
        // do stuff 1
     else if (valItem == valY)
        // do stuff 2
}
```

어떤 경우에는, ordinal() 메쏘드를 통해 enum 정수 값을 얻는 것이 완전히 보장되지는 않아도 더 빠를 것이다.

내부 클래스를 가지는 패키지 영역(Scope)을 사용하라

다음의 클래스 정의를 생각해 보라.

```
public class Foo {
    private int mValue;

    public void run() {
        Inner in = new Inner();
        mValue = 27;
        in.stuff();
    }

    private void doStuff(int value) {
        System.out.println("Value is " + value);
    }

    private class Inner {
        void stuff() {
            Foo.this.doStuff(Foo.this.mValue);
        }
    }
}
```

여기서 주목해야 할 중요한 것은 외부outer 클래스에 있는 프라이빗private 메쏘드

와 프라이빗private 인스턴스 필드에 직접 접근하는 내부inner 클래스(Foo$Inner)를 정의했다는 점이다. 이것은 적법하다. 그리고 그 코드는 기대한 대로 "Value is 27"을 출력한다.

문제는 Foo$Inner가 기술적으로 (배후의) 하나의 완전히 별도의 클래스이고, 이는 Foo의 프라이빗private 멤버들에 대한 직접 접근을 불법으로 만든다는 데 있다. 그 간격을 메우기 위해, 컴파일러는 두개의 합성 메쏘드들synthetic methods을 생성한다.

```
/*package*/ static int Foo.access$100(Foo foo) {
    return foo.mValue;
}
/*package*/ static void Foo.access$200(Foo foo, int value) {
    foo.doStuff(value);
}
```

내부 클래스Inner-class 코드는 외부 클래스outer class에 있는 "mValue" 필드를 접근하거나 또는 "doStuff" 메쏘드를 호출해야 할 필요가 있을 때마다 이러한 정적static 메쏘드를 호출한다. 이것이 의미하는 바는 위의 코드가 여러분이 직접적으로 멤버 필드들을 접근하는 대신에 접근자accessor 메쏘드를 통해 실제로 그 멤버 필드들을 접근하는 경우가 되는 것이다. 앞에서 우리는 접근자accessors가 어떻게 직접적인 필드 접근보다 느린지에 대해 이야기 했다. 그러므로 이것은 보이지 않는 성능 문제를 일으키는 특정 언어 구문의 예제이다.

우리는 이 문제를 내부 클래스에 의해 접근되는 필드와 메쏘드를 프라이빗private 범주private scope보다는 패키지 범주package scope을 가지도록 선언함으로써 피할 수 있다. 이것은 더 빨리 실행되고 생성된 메쏘드의 과부하overhead를 제거한다(불행하게도 그것은 또한 그 필드가 동일한 패키지에 있는 다른 클래스들에 의해서 직접적으로 접근될 수 있다는 것을 의미하며, 이것은 모든 필드를 프라이빗(private)하게 만드는 표준 OO 사례에 어긋난다.다시 한번, 여러분이 public API를 디자인하고 있다면, 여러분은 주의깊게 이러한 최적화를 사용하는 것을 고려하기를 원할 수도 있다).

부동소수점(Float)을 피하라

펜티엄 CPU가 나오기 이전에, 게임 개발자가 가능한 한 많은 것을 정수integer 수

치연산을 사용하는 것이 일반적이었다. 펜티엄과 함께, 부동소수점 수치 연산 보조 프로세서가 내장되었다. 그리고 정수integer와 부동소수점floating-point 오퍼레이션을 끼워 넣음으로써 여러분의 게임은 순수하게 정수 수치 연산만을 사용했을 때보다 실제로 더 빨라지게 되었다. 데스크탑 시스템상에서 일반적인 사례는 부동소수점을 자유롭게 사용하는 것이다.

불행하게도, 임베디드 프로세서들은 종종 하드웨어 부동 소수점 지원을 지원하지 않는다. 그러므로 "float"와 "double"에 대한 모든 오퍼레이션들은 소프트웨어에서 수행된다. 약간의 기본적인 부동 소수점 오퍼레이션들을 완료하기 위해 대략 1 밀리세컨드를 가질 수 있다.

또한, 심지어 정수integer에 대해서도, 일부 칩은 하드웨어적 곱셈은 가지고 있지만 하드웨어적인 나눗셈은 가지고 있지 않다. 이러한 경우에, 정수 나눗셈과 몫 오퍼레이션은 소프트웨어로 수행된다 – 여러분이 해쉬 테이블hash table을 디자인하고 있거나 또는 많은 수치 연산을 하고 있는지 생각해 봐야 한다.

약간의 샘플 성능 숫자들

우리의 생각들 일부를 설명하기 위해서, 여기에 몇 가지 기본 액션에 대한 대략적인 실행 시간을 나열하는 테이블이 있다. 이 값들은 절대 숫자로 받아들여져서는 안된다는 사실에 주의하라: 이 값들은 CPU와 벽 시계 시간의 조합이며, 시스템이 향상되면 변하게 된다. 하지만 이 값들이 각기 다른 것에 대해 얼마나 상대적으로 적용되는 지에 주목할 가치가 있다. 예를 들어 일반적으로 하나의 멤버 변수를 추가하는 것은 하나의 로컬 변수를 추가하는 시간의 네 배가 걸린다.

액 션	시 간
하나의 로컬 변수 추가	1
하나의 멤버 변수 추가	4
String.length() 호출	5
빈 정적(static) 네이티브 메쏘드 호출	5
빈 정적(static) 메쏘드 호출	12
빈 버추얼 메쏘드 호출	12.5
빈 인터페이스 메쏘드 호출	15
HashMap에서 Iterator:next() 호출	165

HashMap에서 put() 호출	600
XML로부터 1개의 뷰 인플레이트(Inflate)	22,000
1개의 TextView를 가지는 1개의 LinearLayout 인플레이트	25,000
6개의 뷰 오브젝트를 가지는 1개의 LinearLayout 인플레이트	100,000
6개의 TextView 오브젝트를 가지는 1개의 LinearLayout 인플레이트	135,000
하나의 빈 액티비티 런치	3,000,000

마무리 노트

임베디드 시스템에 대한 좋고, 효율적인 코드를 작성하는 최고의 방법은 여러분이 작성한 코드가 실제로 무엇을 하는지를 이해하는 것이다. 만약 여러분이 반복실행자iterator 를 할당하고자 한다면, 반드시 List에 대한 enhanced for loop 문법을 사용하라: 그것이 의도하지 않은 부작용으로 작용하지 않게 하고, 신중한 선택이 되도록 하라.

사전경고는 사전 무장과 같다!(유비무환). 여러분이 어디로 가고 있는지를 알라! 여러분이 가장 좋아하는 좌우명을 여기에 넣어라. 그러나 항상 여러분의 코드가 무엇을 하고 있는 지에 대해 주의깊게 생각하라. 그리고 그것의 속도를 올리기 위한 방법에 대해 고려하라.

Section | 21

응답성을 위한 설계

21.1. 무엇이 ANR을 유발하는가?

21.2. 어떻게 ANR을 피할 것인가?

21.3. 응답성 강화하기

21. 응답성을 위한 설계

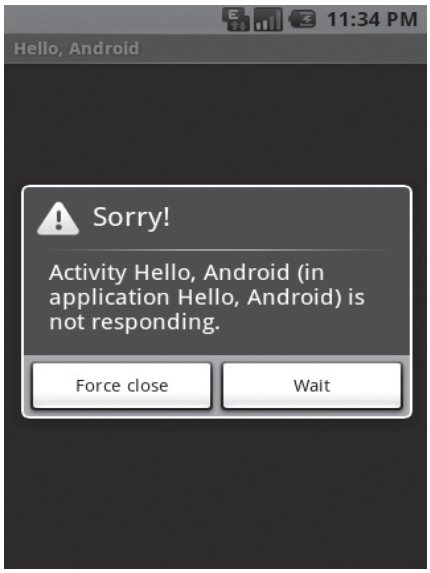
세상의 모든 성능 테스트에서 승리하는 코드를 작성하는 것은 가능하다. 그러나 사용자가 애플리케이션을 사용하려 할 때 그것은 여전히 사용자를 거세게 화나게 하는 것도 가능하다. 이것들은 충분하게 응답하지 않는 애플리케이션들이다 – 느리다고 느끼게 하거나, 상당한 기간 동안 동작하지 않거나 너무 시간이 오래걸려 입력을 처리하지 못하는 애플리케이션들이 그것이다.

안드로이드에서, 시스템은 사용자에게 ANR(애플리케이션이 응답하지 않는다. Application Not Responding) 다이얼로그라고 불리는 하나의 다이얼로그를 사용자에게 보여줌으로써 일정한 시간동안 불충분하게 응답하는 애플리케이션을 감시한다. 사용자는 애플리케이션이 계속되도록 선택할 수는 있지만, 여러분의 애플리케이션을 사용할 때마다 이러한 다이얼로그에 대응해야 하는 것을 좋게 생각하지 않을 것이다. 그러므로 시스템이 사용자에게 결코 ANR을 표시하지 않도록 여러분의 애플리케이션에 응답성responsiveness을 디자인하는 것은 중요하다.

일반적으로 시스템은 애플리케이션이 사용자 입력에 응답할 수 없을 때 ANR을 보여준다. 예를 들어 만약 하나의 애플리케이션이 어떤 I/O 오퍼레이션(자주 네트워크 접속) 상에서 멈춘다면, 메인 애플리케이션 쓰레드는 들어오는 사용자 입력 이벤트를 처리할 수 없게될 것이다. 일정 시간이 흐른 뒤에, 시스템은 애플리케이션이 움직이지 않는 상태가 되었다고 결론내린다. 그리고 사용자에게 애플리케이션을 강제 종료시킬 수 있는 옵션을 주기 위해 ANR을 보여준다.

유사하게, 여러분의 애플리케이션이 복잡한 메모리 내의 구조를 만들거나, 또는 게임에서 다음번 이동을 계산하는데 너무 많은 시간을 사용한다면, 시스템은 여러분의 애플리케이션이 움직이지 않는 상태라고 결론내릴 것이다. 위의 테크닉을 사용해서 이러한 계산을 효율적으로 만드는 것은 항상 중요하다. 그러나 가장 효율적인 코드조차도 실행하는 데 여전히 시간이 걸린다.

이런 모든 경우에, 그 해결책은 보통 자식 쓰레드child thread를 생성하고, 그곳에서 여러분의 작업의 대부분을 수행하는 것이다. 이것은 (사용자 인터페이스 이벤트 루프를 관리하는) 메인 쓰레드를 계속 실행 상태로 유지하고, 시스템이 여러분의 코드가 멈추었다고 판단하는 것을 예방한다. 그러한 쓰레딩threading은 보통 클래스 레벨에서 이루어지기 때문에, 여러분은 응답성을 클래스 문제로 생각할 수 있다(method-level 문제로 위에서 설명했던 기본적인 성능과 이것을 비교해 보라).



이 문서는 안드로이드 시스템이 애플리케이션이 응답하고 있지 않는 지 여부를 결정하는 방법을 설명하고 여러분의 애플리케이션이 응답하는 것을 보증하기 위한 가이드라인을 제공한다.

이 문서는 아래의 주제들을 다룬다.

- 무엇이 ANR을 유발하는가?
- 어떻게 ANR을 피할 것인가?
- 응답성 강화하기.

무엇이 ANR을 유발하는가?

안드로이드에서, 애플리케이션 응답성은 액티비티 매니저와 윈도우 매니저 시스템 서비스에 의해 감시된다. 안드로이드는 다음의 조건 중 하나를 감지할 때 특정한 애플리케이션에 대해 ANR 다이얼로그를 보여줄 것이다.

- 입력 이벤트(예를 들어 키 입력, 스크린 터치)에 5초 이내에 반응이 없음.
- 브로드캐스트 리시버가 10초 이내에 실행을 끝내지 않음.

어떻게 ANR을 피할 것인가?

ANR에 대한 위의 주어진 정의 하에서, 안드로이드 애플리케이션에서 왜 이것이 발생할 수 있는 지와 ANR을 피하기 위해서 여러분의 애플리케이션을 어떻게 최적으로 구조화할 지에 대해 알아보자.

안드로이드 애플리케이션은 하나의 쓰레드(예를 들어 메인(main))상에서 보통 전적으로 실행된다. 이것이 의미하는 바는 메인 쓰레드에서 여러분의 애플리케이션이 수행하는 어떤 것이 완료되는데 긴 시간을 가진다면 ANR을 유발할 수 있다는 것이다. 왜냐하면 여러분의 애플리케이션이 그 자신에게 입력 이벤트 또는 인텐트 브로드캐스트를 처리할 기회를 주지 않기 때문이다.

그러므로 메인 쓰레드에서 실행되는 임의의 메쏘드는 가능한 한 최소한의 일을 해야 한다. 특히 액티비티들은 onCreate(), onResume()와 같은 핵심 생명주기 메쏘드에서 셋업하기 위해 가능한 한 적은 일을 수행해야 한다. 네트워크나 데이터베이스 오퍼레이션 같은 잠재적으로 길게 실행되는 오퍼레이션과 또는 비트맵 크기를 조정하는 것 같이 계산상 값비싼 연산은 자식 쓰레드에서 수행되어야 한다(또는 데이터 베이스 오퍼레이션 경우에는, 비동기적인 요청(asynchronous request)을 통해 수행되어야 한다). 하지만 이것은 여러분의 메인 쓰레드가 자식 쓰레드가 완료되도록 기다리는 동안 멈춰 있어야 하는 것을 의미하지 않는다 – 그리고 또한 Thread.wait() 또는 Thread.sleep()을 호출하지 말아야 한다. 자식 쓰레드가 완료되도록 기다리는 동안 멈추는 대신에, 여러분의 메인 쓰레드는 자식 쓰레드가 완료될 때 알려주기 위한 핸들러를 제공해야 한다. 이러한 방식으로 여러분의 애플리케이션을 디자인하는 것은 여러분의 메인 쓰레드가 입력에 대해 응답성을 유지하는 것을 가능하게 하고, 따라서 5초 입력 이벤트 타임아웃에 의해 발생되는 ANR 다이얼로그를 피할 수 있게 한다. 이와 같은 사례는 UI를 보여주는 임의의 다른 쓰레드들에서도 수반되어야 한다. 왜냐하면 그것들 또한 동일한 타임아웃에 영향을 받기 때문이다.

IntentReciever 실행 시간에 대한 명확한 제약은 그것들이 하기로 되어 있었던 것이 무엇인지를 강조한다: 세팅을 저장하거나 노티피케이션을 등록하는 것과 같은 백그라운드에서의 작고 분리된 양의 작업. 그러므로 메인 쓰레드에서 호출된

다른 메쏘드들처럼, 애플리케이션들은 브로드캐스트 리시버에서 잠재적으로 긴 실행 오퍼레이션 또는 연산을 피해야 한다. 그러나 자식 쓰레드를 통해 집중적인 업무를 수행하는 것 대신에(브로드캐스트의 수명이 짧기 때문에), 만약 인텐트 브로드캐스트에 응답하여 잠재적으로 긴 시간동안 액션이 취해져야 한다면, 여러분의 애플리케이션은 하나의 서비스를 시작시켜야 한다. 주석으로, 여러분은 인텐트 리시버로부터 하나의 액티비티를 시작시키는 것을 또한 피해야 한다. 왜냐하면 그것은 사용자가 현재 실행하고 있는 애플리케이션이 무엇이든지 간에 그것으로부터 포커스를 빼앗을 새로운 스크린을 만들어내기 때문이다. 만약 여러분의 애플리케이션이 인텐트 브로드캐스트에 응답하여 사용자에게 보여주기 위한 어떤 것을 가지고 있다면, 그 애플리케이션은 노티피케이션 매니저를 사용해서 사용자에게 보여줘야 한다.

응답성 강화하기

일반적으로 100ms에서 200ms가 사용자가 애플리케이션에서 지연을 느낄 수 있는 경계 지점threshold이다. 이와 같이, 여러분의 애플리케이션이 사용자에게 더 잘 응답하는 것처럼 보이도록 도와주는 ANR을 피하기 위해 여러분이 해야할 것 이외의 추가적인 팁들이 있다.

- 만약 여러분의 애플리케이션이 사용자 입력에 응답하여 백그라운드에서 일을 하고 있다면, 진행이 되고 있음을 보여주라(ProgressBar와 ProgressDialog가 이 경우 유용하다).
- 특히 게임에서는, 자식 쓰레드에서 움직임에 대한 계산을 하라.
- 여러분의 애플리케이션이 시간이 걸리는 초기 셋업 단계라면 착수 화면splash screen을 보여주거나 가능한 한 빨리 메인 뷰를 렌더링하고 비동기적으로 정보를 채워넣는 것을 고려하라. 어떤 경우든, 여러분은 사용자가 애플리케이션이 멈춘 것으로 인식하지 않도록 진행이 되고 있음을 어떻게든 보여주어야 한다.

Section | 22

무결절성을 위한 설계

22.1. 데이터를 누락하지 마라

22.2. 원시 데이터를 드러내지 마라

22.3. 사용자를 방해하지 마라

22.4. 해야 할 것이 많다면, 쓰레드에서 하라

22.5. 단일 액티비티 스크린에 오버로드를 주지 마라

22.6. 시스템 테마를 확장하라

22.7. 다양한 스크린 해상도에서 작동하도록 UI를 디자인하라

22.8. 네트워크가 느리다는 것을 가정하라

22.9. 터치 스크린 또는 키보드를 가정하지 마라

22.10. 디바이스 배터리를 절약하라

22. 무결절성을 위한 설계

여러분의 애플리케이션이 빠르고 응답이 좋다 할지라도, 어떤 디자인 결정들은 여전히 사용자에게 문제를 야기시킬 수 있다 – 다른 애플리케이션 또는 다이얼로그와 계획되지 않은 상호작용, 데이터의 부주의한 손실, 의도되지 않은 멈춤 등 때문에 그럴 수 있다. 이러한 문제점을 피하기 위해, 여러분의 애플리케이션이 실행되는 컨텍스트와 여러분의 애플리케이션에 영향을 줄 수 있는 시스템 상호작용들을 이해하는 것은 도움이 된다. 간략하게 말하자면, 여러분은 시스템 그리고 다른 애플리케이션과 끊김없이 상호작용하는 애플리케이션을 개발하기 위해 노력해야 한다.

일반적인 무결절성 문제는 하나의 애플리케이션의 백그라운드 프로세스 – 예를 들어 서비스 또는 브로드캐스트 리시버 – 가 어떤 이벤트에 응답하여 다이얼로그를 팝업할 때이다. 이러한 현상은 특히 여러분이 애플리케이션을 에뮬레이터 상에서 독립적으로 빌드하고 테스트할 때에는 해가 없는 것처럼 보일 수도 있다. 하지만 여러분의 애플리케이션이 실제 디바이스 상에서 실행될 때, 여러분의 애플리케이션이 백그라운드 프로세스로 다이얼로그를 보여줄 때 사용자 포커스를 가지지고 있지 않을 수도 있다. 그러므로 여러분의 애플리케이션은 결국에는 액티브(역주 : 현재 포커스를 가지고 있는) 애플리케이션 뒤에서 다이얼로그를 보여줄 것이다. 또한 그것은 현재의(역주 : 포커를 가지고 있는) 애플리케이션으로부터 포커스를 가져올 것이며 (예를 들어 전화를 거는 것과 같은) 사용자가 무엇을 하고 있던지 상관없이 그것 앞에 다이얼로그를 보여줄 수 있다. 이러한 동작방식은 여러분의 애플리케이션 또는 사용자를 위한 일이 아니다.

이러한 문제점을 피하기 위해, 여러분의 애플리케이션은 사용자에게 노티피케이션을 주기 위한 적절한 시스템 설비 – 노티피케이션Notification 클래스를 사용해야 한다. 노티피케이션을 사용할 때, 여러분의 애플리케이션은 포커스를 가져가서 사용자를 방해하기 보다는 상태바status bar에 아이콘을 보여줌으로써 사용자에게 이벤트가 발생했음을 알려줄 수 있다.

무결절성 문제의 또 다른 예제는 하나의 액티비티가 onPause()와 다른 생명주기 메쏘드들을 정확하게 구현하지 않아서 상태나 사용자 데이터를 부주의하게 상실하게 될 때이다. 또는 만약 여러분의 애플리케이션이 다른 애플리케이션에 의해 사용되어지도록 데이터를 보여준다면, 여러분은 그 데이터를 world-readable 원시 파일이나 데이터 베이스를 통해서 하기 보다는 컨텐트 프로바이더를 통해 보여주어야 한다.

이러한 예제들이 공통적으로 가지고 있는 것은, 그 예제들이 시스템과 다른 애플리케이션들과 아주 잘 협력하고 있다는 것이다. 안드로이드 시스템은 애플리케이션을 블랙박스 코드 덩어리가 아닌, 일종의 느슨하게 결합된 컴포넌트의 연합체로 취급하도록 디자인되어 있다. 이것은 여러분이 개발자로서 전체 시스템을 이러한 컴포넌트의 훨씬 더 큰 연합체로 볼 수 있게 한다. 이것은 여러분으로 하여금 다른 애플리케이션을 깨끗하고cleanly 끊김없이seamlessly 통합할 수 있게 함으로써 여러분에게 이점을 준다. 그러므로 여러분은 자신의 코드가 그 혜택을 돌려주도록 디자인해야 한다.

이 문서는 일반적인 무결정성 문제들과 그것들을 회피하는 방법을 논의한다. 이것은 다음의 토픽들을 다룬다.

- 데이터를 누락하지 마라.
- 원시 데이터를 드러내지 마라.
- 사용자를 방해하지 마라.
- 해야 할 것이 많다면, 쓰레드에서 하라.
- 단일 액티비티 스크린에 오버로드를 주지 마라.
- 시스템 테마를 확장하라.
- 다양한 스크린 해상도에서 작동하도록 UI를 디자인하라.
- 네트워크가 느리다는 것을 가정하라.
- 터치 스크린 또는 키보드를 가정하지 마라.
- 디바이스 배터리를 절약하라.

데이터를 누락하지 마라

안드로이드가 모바일 플랫폼이라는 사실을 항상 유의하라. 이것은 당연해 보일 수도 있지만, ("전화가 오는 것"과 같은) 다른 액티비티가 어떤 순간에 여러분 자신의 액티비티 위에 팝업될 수 있다는 점을 기억하는 것은 중요하다. 이것은 onSaveInstanceState()와 onPause() 메쏘드를 호출할 것이고, 여러분의 애플리케이션이 종료되도록 할 것이다.

만약 다른 액티비티가 나타날 때 사용자가 여러분의 애플리케이션 데이터를 편집하는 중이었다면, 여러분의 애플리케이션은 종료될 때 진행 중인 작업을 먼저 저장하지 않는다면 데이터를 상실하게 될 것이다. 안드로이드 방식Android Way은 진행중인 작업을 저장하는 것이다: 입력을 받거나 편집하는 안드로이드 애플리케이션은 onSaveInstanceState() 메쏘드를 오버라이드해야 하고 어떤 적절한 방식으로 그 애플리케이션의 상태를 저장해야 한다. 그 사용자가 그 애플리케이션을 다시 방문할 때, 그 데이터를 다시 가져오는 것이 가능해야 한다.

이런 행동을 잘 사용한 전통적인 예제가 메일 애플리케이션이다. 만약 사용자가 다른 액티비티가 시작되었을 때 이메일을 작성하고 있는 중이었다면, 애플리케이션은 작성 중인 이메일을 초고draft로 저장해야 한다.

원시 데이터를 드러내지 마라

여러분이 속옷 차림으로 거리를 걷지 않는 것과 마찬가지로 여러분의 데이터도 그러지 않아야 한다. 특정 종류의 애플리케이션을 외부에 드러내어 그 애플리케이션의 데이터를 읽도록 하는 것은 가능하지만, 이것은 보통 최선의 안은 아니다. 원시 데이터를 드러내는 것은 다른 애플리케이션들이 여러분의 데이터 포맷을 이해하도록 요구한다. 만약 여러분이 그 포맷을 변경한다면, 비슷하게 업데이트되지 않은 다른 애플리케이션들을 망가뜨릴 것이다.

"안드로이드 방식Android Way"은 여러분의 데이터를 다른 애플리케이션들에 깨끗하고, 잘 정리되어 있고, 그리고 유지보수 가능한 API를 드러내기 위해 컨텐트 프로바이더를 생성하는 것이다. 컨텐트 프로바이더를 사용하는 것은 두 개의 긴밀히 연결된 코드를 나누어 컴포넌트화기 위한 Java 언어 인터페이스를 삽입하는 것과

매우 비슷하다. 이것은 컨텐트 프로바이더에 의해 드러난 인터페이스를 변경하지 않고, 그리고 다른 애플리케이션에 영향을 주지 않으면서도 데이터의 내부 포맷을 수정할 수 있다는 것을 의미한다.

사용자를 방해하지 마라

만약 사용자가 (통화하는 동안 폰 애플리케이션과 같은) 애플리케이션을 실행하고 있다면, 그가 목적을 가지고 그것을 했다는 것은 확실한 것이다. 이것은 여러분이 현재의 액티비티로부터 사용자의 입력에 직접 응답한 것을 제외하고 액티비티들을 만드는 것을 피해야 하는 이유이다.

즉, 백그라운드에서 실행되는 브로드캐스트 리시버나 서비스에서 startActivity()를 호출하지 말아라. 그렇게 하는 것은 애플리케이션이 현재 실행하고 있는 것이 무엇이든지 간에 방해를 할 것이며, 그래서 사용자에게 불편을 초래할 것이다. 더 나쁜 상황은, 여러분의 액티비티가 "키입력 도적keystoke bandit" 이 될 수도 있고, 사용자가 이전의 액티비티에 제공하고 있었던 입력을 수신할 수도 있다. 여러분의 애플리케이션이 무엇을 하는 지에 따라서, 이것은 좋지 않은 소식이 될 수도 있다.

백그라운드로부터 직접적으로 액티비티 UI를 생성하는 것 대신, 여러분은 노티피케이션을 설정하기 위한 노티피케이션 매니저를 대신 사용해야 한다. 이것들은 상태 바에서 나타날 것이고, 그래서 사용자는 여러분의 애플리케이션이 그에게 보여주어야 하는 것을 보기 위해 틈이 날 때 그것을 클릭할 수 있다.

(위에서 언급한 모든 것은 여러분의 액티비티가 이미 포어그라운드에 있는 모든 경우에는 적용되지는 않는다는 것에 주의하라: 그 경우에, 사용자는 입력에 반응하여 여러분의 다음 액티비티를 보길 기대한다.)

해야 할 것이 많다면, 쓰레드에서 하라

만약 여러분의 애플리케이션이 다소 비싸거나 오랜시간 실행되는 연산을 수행하는 것이 필요하다면, 여러분은 아마도 그것을 하나의 쓰레드로 옮겨야 한다. 이것은 여러분의 애플리케이션이 종료되는 최종 결과를 가지는, "애플리케이션이 응답하지 않습니다(Application Not Responding)"라는 공포스러운 다이얼로그가 사용자에게 보여지는 것을 막을 것이다.

디폴트로, 액티비티에 있는 모든 뷰들과 모든 코드들은 동일한 쓰레드에서 실행된다. 이것은 또한 UI 이벤트를 처리하는 것과 동일한 쓰레드이다. 예를 들어 사용자가 키를 눌렀을 때, 키 다운 이벤트가 액티비티의 메인 쓰레드 큐에 추가된다. 그 이벤트 핸들러 시스템은 그 이벤트를 재빨리 큐에서 꺼내어 처리하는 것이 필요하다. 만약 그것이 안된다면, 시스템은 몇 초 후에 애플리케이션이 멈춰있다고 결론내리고, 사용자에게 그것을 강제 종료하라고 제안한다.

만약 여러분이 오랜시간 실행되는 코드를 가지고 있다면, 그것을 여러분의 액티비티에서 인라인으로 실행하는 것은 그 이벤트 핸들러 쓰레드 상에서 그것을 실행할 것이다. 그것은 그 이벤트 핸들러를 사실상 방해한다. 이것은 입력 처리를 지연시킬 것이고, 결국 ANR 다이얼로그를 초래할 것이다. 이것을 피하기 위해, 여러분의 연산을 쓰레드로 옮겨라. 21장. "응답성을 위한 설계" 문서가 그것을 하는 방법에 대해 설명한다.

단일 액티비티 스크린에 오버로드를 주지 마라

사용할 가치가 있는 임의의 애플리케이션은 아마 여러 개의 다른 스크린을 가지고 있을 것이다. 여러분의 UI 스크린을 디자인할 때, 여러 개의 액티비티 오브젝트 인스턴스들을 사용하도록 하라.

여러분의 개발 경험에 의존해서, 여러분은 하나의 액티비티가 여러분 애플리케이션에 대한 시작점이라는 점에서 하나의 Java 애플릿과 유사하다고 해석할 수 있다. 하지만 그것은 그렇게 정확하지는 않다: Applet 서브클래스는 Java 애플릿에 대한 단일한 시작점이지만, 액티비티는 여러분의 애플리케이션에 대한 잠재적인 여러 시작점 중에 하나로 간주되어야 한다. 여러분의 "메인main" 액티비티와 여러분이 가질 수도 있는 다른 액티비티들간의 유일한 차이는, 그 메인main 액티비티는 단지 여러분의 AndroidManifest.xml 파일에 있는 "android.intent.action.MAIN" 액션에 관심을 나타내고 있다는 것이다.

그러므로 여러분의 애플리케이션을 디자인할 때, 여러분의 애플리케이션을 액티비티 오브젝트의 연합체로 생각하라. 이것은 결국에는 여러분의 코드를 훨씬 더 유지보수 가능하게 할 것이며, 그리고 또한 좋은 부수 효과로써 안드로이드 애플리케이션 히스토리와 "뒤로 가기backstack" 모델과 잘 협력하게 한다.

시스템 테마를 확장하라

사용자 인터페이스의 룩앤필look-and-feel에 대해, 잘 조화를 이루는 것은 중요하다. 사용자들은 그들이 기대했던 사용자 인터페이스와 현저하게 다른 애플리케이션에 의해 거슬리게 된다. 여러분의 UI를 디자인할 때, 여러분은 가능한 많이 여러분 자신을 튀게 하는 것을 해서는 안된다. 대신, 테마를 사용하라. 여러분은 필요로 하는 그 테마의 일부를 오버라이드하거나 또는 확장할 수 있다. 그러나 최소한 여러분은 다른 모든 애플리케이션과 같은 UI 기반에서 시작해야 한다. 자세한 사항에 대해서는 "스타일과 테마 적용하기"를 읽어라.

다양한 스크린 해상도에서 작동하도록 UI를 디자인하라

다른 안드로이드 디바이스들은 다른 스크린 해상도를 지원할 것이다. 어떤 것은 가로 모드landscape mode로 변경하는 것과 같이, 심지어는 동작 중에 해상도를 변경할 수도 있을 것이다. 여러분의 레이아웃과 드로어블drawables을 다양한 디바이스 스크린 상에서 적절하게 표시될 수 있을 만큼 유연하게 만드는 것은 중요하다.

다행히 이것은 매운 쉬운 일이다. 간략하게 말해서, 여러분이 해야만 하는 것은 주요 해상도에 대해 다른 버전의 아트워크artwork를 제공하는 것이다. 그래서 여러분의 레이아웃이 다양한 크기를 수용하도록 디자인하라(예를 들어 힘들게 직접 계산한 위치를 사용하는 것을 피하고 대신 RelativeLayout을 사용하라). 만약 여러분이 이렇게 많이 한다면, 시스템은 그 나머지를 처리할 것이고 여러분의 애플리케이션은 어떤 디바이스에서도 멋지게 보일 것이다.

네트워크가 느리다는 것을 가정하라

안드로이드 디바이스들은 다양한 네트워크 커넥션 옵션과 함께 동작한다. 어떤 것이 다른 어떤 것들보다 더 빠르다 할 지라도, 모든 옵션들은 약간의 데이터 접근 규정data-access provision을 가질 것이다. 하지만 그 최소 공통 분모는 GSM 네트워크에 대해서는 3G 데이터 서비스가 아닌 GPRS이다. 심지어 3G가 가능한 디바이스 조차도 많은 시간을 3G가 지원되지 않는 네트워크 상에서 보내게 될 것이다. 그러므로 느린 네트워크는 꽤나 오랜 시간 동안 접하게 되는 현실로 남을 것이다.

이것이 여러분이 애플리케이션을 네트워크 접속과 대여폭을 최소화하도록 작성해야 하는 이유이다. 여러분은 네트워크가 빠르다고 가정할 수 없다. 그러므로 여러분은 느린 네트워크에 대한 계획을 항상 가지고 있어야 한다. 만약 여러분의 사용자가 더 빠른 네트워크에 있을 수 있다면, 그것은 좋은 일이다 – 단지 사용자 경험이 개선될 뿐이다. 여러분은 그 반대의 경우를 피하고 싶을 뿐이다: 어떤 시간에는 사용될 수 있지만, 어느 특정 시점에 주어진 사용자가 있는 곳에 기반하여 나머지 시간에는 실망스럽게도 느린 애플리케이션은 인기가 없을 것이다.

만약 여러분이 에뮬레이터를 사용하고 있다면 이런 함정에 빠지는 것이 매우 쉽다. 왜냐하면 그 에뮬레이터가 여러분의 데스크탑 컴퓨터의 네트워크 커넥션을 사용하고 있기 때문이다. 그것은 대개 무선 네트워크보다 훨씬 더 빠르다는 것이 보장된다. 그러므로 여러분은 더 느린 네트워크 속도를 시뮬레이션하는 에뮬레이터의 설정을 변경하고 싶을 것이다. 여러분은 이클립스eclipse에서 런치 설정의 "Emulator Settings" 탭이나 또는 에뮬레이터를 시작할 때 커맨드 라인의 옵션을 통해 이것을 할 수 있다.

터치 스크린 또는 키보드를 가정하지 마라

안드로이드는 다양한 핸드폰 형태를 지원할 것이다. 어떤 안드로이드 디바이스는 전체 QWERTY 키보드를 가지고 있을 것이고, 그에 반해 다른 것들은 40-key, 12-key, 또는 다른 키 설정을 가질 것이라고 말하는 것은 엉뚱한 생각이다. 비슷하게, 어떤 디바이스들은 터치 스크린을 가질 것이고, 많은 것들은 그렇지 않을 것이다.

여러분의 애플리케이션을 만들 때, 이를 염두에 두어라: 특정 키보드 레이아웃에 대해 가정하지마라 – 당연하겠지만, 만약 여러분의 애플리케이션이 특정 디바이스에서만 사용될 수 있도록 그것을 제한하는 것에 관심을 가지지 않는다면 이를 염두에 두어야 한다.

디바이스 배터리를 절약하라

모바일 디바이스가 항상 벽 플러그에 꽂혀져 있다면 전혀 모바일이 아니다. 모바

일 디바이스는 배터리로 동작한다. 그리고 배터리가 더 오래가도록 만들 수 있다면, 사용자는 더 행복하게 될 것이다 – 배터리 전원에 대한 가장 큰 두 소비자는 프로세서와 라디오radio이다; 이것이 여러분의 애플리케이션을 가능한 한 적게 작업하고, 그리고 가능한 한 드물게 네트워크를 사용하게 작성하는 것이 중요한 이유이다.

여러분의 애플리케이션이 사용하고 있는 프로세서 시간의 양을 최소화하는 것은 실제로는 효율적인 코드를 작성하는 것이 된다. 라디오 사용으로부터 배터리 소모를 최소화하도록 하기 위해 우아하게 에러 조건을 처리하고 여러분이 필요한 것만을 가져오도록 하라. 예를 들어 한 번 실패하면 네트워크 오퍼레이션을 계속 재시도하지 마라. 만약 그것이 한 번 실패하면, 그것은 아마도 사용자가 어떤 수신도 하지 못하기 때문이다. 그러므로 만약 여러분이 바로 다시 시도한다면 그것은 아마도 다시 실패하게 될 것이다; 여러분이 하는 모든 것은 배터리 전원을 낭비한다.

사용자들은 꽤 영리하다: 만약 여러분의 프로그램이 배터리를 많이 소모한다면, 사용자들은 그것을 알아챌 것이다. 그 시점에서 여러분이 확실하게 할 수 있는 유일한 것은 여러분의 프로그램이 오랫동안 설치되어 있지 않을 것이라는 것이다.

Section | 23

용어집

23. 용어집

용어집

아래의 리스트는 안드로이드 플랫폼의 기본적인 용어들을 정의하고 있다.

.apk 파일(.apk file)

안드로이드 애플리케이션 패키지 파일. 각각의 안드로이드 애플리케이션은 컴파일 되어 하나의파일로 패키지된다. 그것은 애플리케이션의 코드(.dex 파일), 리소스, 에셋, 그리고 매니페스트 파일 모두를 포함하고 있다. 애플리케이션 패키지 파일은 어떠한 이름도 가질 수 있지만 .apk 확장자를 사용해야만 한다. 예를 들어: myExampleAppname.apk. 편의상, 애플리케이션 패키지 파일은 종종 ".apk"로 언급된다.

관련: Application

.dex 파일(.dex file)

컴파일된 안드로이드 애플리케이션 코드 파일.

안드로이드 프로그램은 .dex (Dalvik Executable) 파일로 컴파일된다. 그것은 디바이스 상의 하나의 .apk 파일내에 zip으로 압축된다. .dex 파일들은 자바 프로그래밍 언어로 작성된 컴파일된 애플리케이션을 자동적으로 변환함으로써 생성된다.

액션(Action)

인텐트Intent를 발송하는 것이 수행하길 원하는 어떤 것에 대한 설명. 액션Action은 인텐트에 할당된 문자열 값이다. 액션 문자열들은 안드로이드 또는 제3의 개발자에 의해 정의될 수 있다. 예를 들어 웹 URL에 대한 android.intent.action.VIEW, 또는 폰을 진동시키는 커스텀 애플리케이션에 대한 com.example.rumbler.SHAKE_PHONE이 있다.

관련 : Intent

액티비티(Activity)

애플리케이션 내의 하나의 화면이며, Java code를 지원함으로써 Activity 클래스로부터 파생된다. 가장 일반적으로, 액티비티는 그것이 윈도우를 렌더render하기 위해 사용하는 윈도우 때문에, UI 이벤트를 받아서 처리할 수 있고 복잡한 태스크를 수행하는 전체 스크린 윈도우에 의해 시각적으로 표현된다. 액티비티가 일반적으로는 전체 스크린임에도 불구하고, 그것은 또한 화면 위에 떠 있거나floating 투명하게 될 수도 있다.

adb

Android Debug Bridge로, SDK와 함께 포함된 커맨드 라인 디버깅 애플리케이션. 그것은 디바이스를 브라우징하기 위한 도구, 디바이스에 복사하는 도구, 그리고 디버깅을 위해 포워드 포트를 제공한다. 만약 여러분이 ADT 플러그인을 사용하는 이클립스에서 개발하고 있다면, adb는 여러분의 개발 환경안에 통합된다. 더 많은 정보를 위해서는 다음에 나오는 URL에서 adb를 보라. http://developer.android.com/guide/developing/tools/adb.html

애플리케이션(Application)

컴포넌트 측면에서 보면, 안드로이드 애플리케이션은 하나 또는 그 이상의 액티비티, 서비스, 컨텐트 프로바이더, 브로드캐스트 리시버로 구성된다. 소스 파일 측면에서 보면, 안드로이드 애플리케이션은 코드, 리소스, 에셋, 그리고 하나의 매니페스트로 구성된다. 컴파일 도중에, 이러한 파일들은 애플리케이션 패키지 파일(.apk)로 불리는 하나의 파일로 패키지된다.

관련: .apk, Activity

캔버스(Canvas)

비티맵Bitmap이나 서피스Surface 오브젝트에 대한 실질적인 비트bit 컴포지션compositing을 처리하는 그리기를 하는 표면surface. 그것은 비트맵, 선, 원, 사각형, 텍스트, 그리고 기타 등등에 대한 표준 컴퓨터 그리기에 대한 메쏘드를 가지고 있다. 캔버스는 스크린에 2D 오브젝트들을 그리기 위한 가장 간단하고 가장 손쉬운 방법이다. 하지만 이것은 OpenGL ES 하는 것 처럼 하드웨어 가속기를 지원하지는 않는다. 베이스 클래스는 캔버스Canvas이다.

관련: Drawable, OpenGL ES

컨텐트 프로바이더(Content Provider)

여러분의 애플리케이션의 데이터를 다른 애플리케이션에게 안전하게 보여주기 위해 여러분이 사용할 수 있는 데이터 추상화 계층data-abstraction layer. 컨텐트 프로바이더는 ContentProvider 클래스 위에서 만들어졌다. 이것은 특정 포맷의 데이터를 리턴하기 위한 특정 포맷의 컨텐트 쿼리query 포맷을 처리한다. 더 많은 정보를 위해서는 7장. "컨텐트 프로바이더"를 보라.

관련: URI Usage in Android

달빅(Dalvik)

안드로이드 플랫폼의 가상 머신. 달빅 가상머신은 효율적인 저장공간과 메모리 매핑가능한 실행을 위해 최적화 된 .dex (Davik Executable) 포맷의 파일들을 실행하는 인터프리터 전용(interpreter-only) 가상 머신이다. 이 가상 머신은 레지스터 기반이고, Java 언어 컴파일러에 의해 컴파일된 클래스를 실행할 수 있다. 그 클래스는 포함되어 있는 "dx" 도구를 사용해서 달빅 네이티브 포맷으로 변환되어 지게 된다. 그 가상 머신은 Posix 호환 운영 체제의 위에서 실행된다. 그것은 (쓰레딩과 저수준 메모리 관리와 같은) 그 아래에 있는 기능을 위해서 해당 운영체제에 의존한다. 달빅 코어 클래스 라이브러리Dalvik core class library는 Java Standard Edition으로 프로그래밍하기 위해 사용된 것들과 비슷한 개발 기반을 제공하고자 의도된 것이지만, 특히 작은 모바일 디바이스의 필요에 맞게 조정되었다.

DDMS

달빅 디버그 모니터 서비스Dalvik Debug Monitor Service로, SDK와 함께 포함된 GUI 디버깅 애플리케이션. 그것은 스크린 캡쳐, 로그 덤프, 프로세스 검사 기능들을 제공한다. 여러분이 ADT 플러그인을 사용하는 이클립스에서 개발하고 있다면, DDMS는 여러분의 개발 환경에 통합되어 진다. 이 프로그램에 대하여 배우고 싶다면, Dalvik Debug Monitor Server를 보라.

다이얼로그(Dialog)

가벼운 폼form처럼 동작하는 떠 있는floating 윈도우. 다이얼로그는 오직 버튼 컨트롤들만을 가질 수 있으며, (버튼 선택과 같은) 간단한 액션을 수행하고 아마 하나의 값을 리턴하도록 되어 있다. 다이얼로그는 히스토리 스택에 존속하거나,

복잡한 레이아웃을 가지거나, 또는 복잡한 액션을 수행하도록 되어 있지 않다. 안드로이드는 여러분을 위해 옵션 버튼을 가지는 간단한 디폴트 다이얼로그를 제공한다. 그럼에도 불구하여 여러분은 자신의 다이얼로그 레이아웃을 정의할 수 있다. 다이얼로그에 대한 베이스 클래스는 다이얼로그Dialog 이다.

관련: Activity

드로어블(Drawable)

배경, 타이틀, 또는 스크린의 다른 영역으로 사용될 수 있는 컴파일 된 시각적 리소스. 드로어블Drawable은 예를 들어 백그라운드 이미지 같은 다른 UI 엘리먼트 안에 일반적으로 로드load된다. 드로어블은 이벤트를 받을 수 없지만, 애니메이션 오브젝트나 이미지 라이브러리와 같은 서브 클래스를 활성화하기 위해 "state"와 스케쥴링과 같은 다양한 다른 속성들이 부여한다. 많은 드로어블 오브젝트들은 드로어블 리소스 파일들, xml 또는 이미지를 그리는 비트맵 파일들로부터 적재된다. 드로어블 리소스들은 android.graphics.drawable의 서브클래스로 컴파일된다. 드로어블과 다른 리소스에 대해 더 많은 정보를 위해서는 4장. "리소스와 에셋"을 보라.

관련: Resources, Canvas

인텐트(Intent)

다른 애플리케이션/액티비티를 런치하거나 또는 비동기적으로 커뮤니케이션 하는데 사용할 수 있는 메시지 오브젝트. 인텐트 오브젝트는 인텐트에 대한 하나의 인스턴스이다. 그것은 어떤 애플리케이션/액티비티가 인텐트를 수신할지와 그 인텐트를 처리할 때 그 리시버가 무엇을 하는지를 결정하기 위해, 여러분이 제공할 수 있는 다양한 기준 필드들을 포함한다. 유용한 기준들은 요구되는 액션, 카테고리, 데이터 문자열, 데이터의 MIME 타입, 처리할 클래스, 그리고 그 밖의 것들을 포함한다. 하나의 애플리케이션은 다른 애플리케이션/액티비티에게 직접 인텐트를 전송한다기보다는 안드로이드 시스템에 인텐트를 보낸다. 애플리케이션은 하나의 타겟 애플리케이션에 인텐트를 보낼 수 있다. 또한 그것은 여러 개의 애플리케이션에 의해 순서대로 차례로 처리될 수 있는 브로드캐스트로 그것을 전송할 수도 있다. 안드로이드에 시스템은 인텐트에 제공된 기준들과 다른 애플리케이션에 의해 정의된 인텐트 필터에 기반해서, 각각의 인텐트에 대한 가장 유용한 리시버를 확정하는 것에 대해 책임을 진다.

더 많은 정보를 보고자 한다면 5장. “인텐트와 인텐트 필터”를 보라.

관련: Intent Filter, Broadcast Receiver

인텐트 필터(Intent Filter)

애플리케이션이 그것의 매니페스트 파일에 선언하는 필터 오브젝트로, 시스템에게 그것의 컴포넌트 각각이 받아들이고자 하는 인텐트의 타입이 무엇인지, 그리고 어떤 기준을 가지고 있는지를 알려준다. 인텐트 필터를 통해, 애플리케이션은 특정한 데이터 타입, 인텐트 액션, URI 포맷 등에 관심을 표현할 수 있다. 인텐트를 확정할 때, 시스템은 모든 애플리케이션 내의 유용한 인텐트 필터 모두를 평가하고, 그 인텐트와 기준들이 가장 일치하는 애플리케이션/액티비티에 인텐트를 전달한다. 더 많은 정보는 5장. “인텐트와 인텐트 필터”를 보라

관련: Intent, Broadcast Receiver

브로드캐스트 리시버(Broadcast Receiver)

하나의 타겟 애플리케이션/액티비티에게 전송되는 것보다는 브로드캐스트 된 인텐트를 수신하는 애플리케이션 클래스. 시스템은 모든 관심갖는 브로드캐스트 리시버에게 브로드캐스트 인텐트를 전달한다. 그것들은 순차적으로 인텐트를 처리한다.

관련: Intent, Intent Filter

레이아웃 리소스(Layout Resource)

액티비티 스크린의 레이아웃을 설명하는 XML 파일..

관련: Resources

매니페스트 파일(Manifest File)

각각의 애플리케이션이 애플리케이션의 패키지 이름, 버전, 컴포넌트(액티비티, 프로바이더, 리시버, 서비스), 임포트import 라이브러리, 그리고 다양한 액티비티를 설명하고, 그리고 기타 등등을 설명하기 위해 정의해야 하는 XML 파일. 완벽한 정보를 위해서는 9장. “AndroidManifest.xml 파일”을 보라.

나인-패치/9-패치/나인패치 이미지

(Nine-patch/9-patch/ Ninepatch image)

디바이스 상의 배경이나 다른 이미지를 위해 사용될 수 있는 크기 조정이 가능한 비트맵 리소스. 더 많은 정보를 위해서는 4장. "리소스와 에셋"의 "사용가능한 리소스 타입" 중 "나인패치 확대가능한stretchable 이미지"를 보라.

관련: Resources

OpenGL ES

안드로이드는 빠르고 복잡한 3D 이미지를 위해 여러분이 사용할 수 있는 OpenGL ES 라이브러리를 제공한다. 이것은 캔버스Canvas 오브젝트보다 사용하기는 어렵지만 3D 오브젝트를 위해서는 더 좋다. android.opengl과 javax.microedition.khronos.opengles 패키지는 OpenGL ES 기능을 보여준다.

관련: Canvas, Surface

리소스(Resources)

컴파일된 애플리케이션 코드 외부에 있는 프로그램적이지 않는 애플리케이션 컴포넌트, 그러나 그것은 잘 알려진 참조 포맷을 사용하여 애플리케이션 코드로 부터 적재될 수 있다. 안드로이드는 다양한 리소스 타입을 지원한다. 그러나 일반적인 애플리케이션 리소스는 UI 문자열, UI 레이아웃 컴포넌트, 그래픽, 또는 다른 미디어 파일, 기타 등등으로 구성될 것이다. 애플리케이션은 효율적으로 로컬라이제이션localization과 다양한 디바이스 프로파일과 상태를 지원하기 위해 리소스를 사용한다. 예를 들어 애플리케이션은 지원되는 각각의 로컬 또는 디바이스 타입에 대해 별도의 리소스 집합을 포함할 것이고, 그리고 그것은 현재의 스크린 방위(가로 혹은 세로)에 대한 특정 레이아웃 리소스를 포함할 수 있다. 리소스에 대한 더 많은 정보를 위해서는 Resoureces and Assets를 참고하라. 애플리케이션의 리소스는 항상 프로젝트의 res/* 하위 폴더에 저장되어 있다.

서비스(Service)

음악 재생 또는 네트워크 액티비티 모니터링과 같은 다양한 지속적인 액션들을 수행하기 위해 (어떤 UI를 표시하지 않고) 백그라운드에서 실행되는 클래스 서비스의 오브젝트.

관련: Activity

서피스(Surface)

스크린으로 컴포지트되는composited 메모리 블록을 나타내는 서피스Surface 타입의 오브젝트. 서피스Surface는 그리기를 위한 하나의 캔버스Canvas 오브젝트를 보유하며, 레이어를 그리고 그 서피스의 크기를 바꾸기 위한 다양한 헬퍼helper 메쏘드를 제공한다. 여러분은 이 클래스를 직접 사용하지 말아야 하며, 대신 서피스뷰SurfaceView를 사용하라.

관련: Canvas

서피스뷰(SurfaceView)

그리기를 위한 서피스Surface를 감싸고 있는 뷰 오브젝트로, 동적으로 그것의 크기와 포맷을 지정하기 위한 메쏘드들을 제시한다. 서피스뷰SurfaceView는 (게임이나 카메라 미리보기와 같은) 리소스 집약적resource-intensive 오퍼레이션을 위해 UI thread와는 독립적으로 그리는 방법을 제공한다. 그러나 그것은 결과적으로 추가적인 메모리를 사용한다. 서피스뷰SurfaceView는 캔버스Canvas와 OpenGL ES 그래픽 모두를 지원한다. 베이스 클래스는 서피스뷰SurfaceView이다.

관련: Surface

테마(Theme)

다양한 디폴트 디스플레이 설정을 정의하기 위해 함께 묶어진 속성들(텍스트 크기, 백그라운드컬러, 그리고 기타 등등)의 집합. 안드로이드는 (“Theme_”으로 시작하는) R.style에 리스트된 몇 가지 표준 테마를 제공한다.

안드로이드 URI들(URIs in Android)

안드로이드는 (컨택 리스트를 얻어오는 것과 같은) 컨텐트 프로바이더에 데이터를 요청하는 것과 (브라우저에서 웹 페이지를 오픈하는 것과 같은) 인텐트에 액션을 요청하는 것에 대한 기초로 URI 문자열을 사용한다. 그 URI 스키마scheme과 포맷은 사용 유형에 따라 달라지며, 애플리케이션은 원하는 어떠한 방식으로도 특정 URI 스키마와 문자열을 처리할 수 있다. 어떤 URI 스키마는 시스템 컴포넌트에 의해 예약되어 있다. 예를 들어 컨텐트 프로바이더로부터의 데이터에 대한 요청은 content://를 사용해야 한다. 인텐트에서, http:// 스키마를 사용하는 URI는 브라우저에 의해 처리될 것이다.

뷰(View)

스크린에서 직사각형의 영역을 그리는 오브젝트이며, 클릭click, 키스트로크keystroke, 그리고다른 상호작용 이벤트를 처리한다. 뷰는 액티비티나 다이얼로그 스크린(텍스트 상자, 윈도우, 그리고 기타 등등)의 레이아웃 컴포넌트 대부분에 대한 베이스 클래스이다. 뷰는 스스로를 그리기 위해 그것의 부모parent 오브젝트(아래의 뷰그룹(Viewgroup)을 보라)로부터 호출을 받는다. 그리고 (부모(parent) 오브젝트에 의해 반영될 수도 있고 그렇지 않을 수도 있지만) 그것이 어디에 있어야 하고 얼마나 커야 하는지에 대해 그것의 부모 오브젝트에게 알려준다. 더 많은 정보는 View를 보라.

관련: Viewgroup, Widget

뷰그룹(Viewgroup)

자식child 뷰들의 집합을 그룹화하는 컨테이너 오브젝트. 뷰그룹은 자식 뷰들이 적절할 때 스스로를 그리도록 하기 위해 각각을 호출하는 것뿐만 아니라, 자식뷰들이 어디에 위치하고 얼마나 커야하는가를 결정하는 것에 대해 책임을 진다. 어떤 뷰그룹은 다른 것들이 고유의 UI(예를 들어 스크롤되는 리스트 상자)를 가지는 것에 반해, 보여지지 않고 단지 레이아웃을 위해서만 존재하게 된다. 뷰그룹들은 widget 패키지 안에 모두 있지만, 뷰그룹Viewgroup을 확장extend한다.

관련: View

위젯(Widget)

텍스트 상자 또는 팝업 메뉴와 같은 폼 엘리먼트와 다른 UI 컴포넌트를 렌더 render하는 완전하게 구현된 뷰 서브클래스들의 집합 중 하나. 위젯은 완전하게 구현되었기 때문에, 그것은 스스로를 측정하는 것과 그리는 것, 그리고 스크린 이벤트에 반응하는 것을 처리한다. 위젯은 모두 android.widget 패키지에 있다.

윈도우(Window)

안드로이드 애플리케이션에서 룩앤필(look and feel, 타이틀 바 텍스트, 메뉴의 위치와 컨텐츠, 그리고 기타 등등)과 같은, 일반적인 윈도우의 엘리먼트를 명시하는 추상 클래스 Window로부터 파생된 오브젝트. 다이얼로그와 액티비티는 윈도우를 렌더 render하기 위해서 이 클래스의 구현물을 사용한다. 여러분은 이 클래스를 구현할 필요없이 여러분의 애플리케이션에서 윈도우를 사용할 수 있다.

Index

Numbers

2차원 그래픽 331
나인패치 드로어블(NinePatchDrawable) 338
드로어블(Drawables) 332
쉐이프 드로어블(ShapeDrawable) 336
트윈 애니메이션(Tween Animation) 340
프레임 애니메이션(Frame Animation) 343

Roman

A

Adapter 63
AdapterView 142
Affinity 32
AIDL 378
AIDL을 사용해서 IPC 구현하기 378
AlertDialog 87
AlertDialog 생성 91
리스트 추가하기(Adding a list) 93
버튼 추가하기(Adding buttons) 92
체크 박스와 라디오 버튼 추가하기 93
AndroidManifest.xml 252
action 261
activity 262
activity-alias 274
application 277
category 281
data 282
grant-uri-permission 285
instrumentation 287
intent-filter 288
meta-data 293
path-permission 294
permission 296
permission-group 298
permission-tree 300
provider 301
receiver 306
service 310
supports-screens 313
uses-configuration 315
uses-feature 317
uses-library 320
uses-sdk 321
매니페스트 파일 구조 253
파일 규약 254
파일 특징 257
ANR 446
무엇이 ANR을 유발하는가? 447
어떻게 ANR을 피할 것인가? 448
App 위젯 364
AppWidgetProviderInfo 메타 데이터 추가하기 366
AppWidgetProvider 클래스 사용하기 369
App 위젯 레이아웃 생성하기 368
App 위젯 선언하기 365
App 위젯 설정 액티비티 생성하기 372

C

Constants 436

D

DatePickerDialog 87

E

Enums 438

F

Float 441

G

Getters 434

I

ID 69

Inner Classes 440

Interface 433

IPC 메쏘드 호출 386

L

Launch Mode 33

Loop 437

M

manifest 291

N

Native Method 433

O

OpenGL 345

P

ProgressDialog 87

ProgressDialog 생성 94, 95

두 번째 쓰레드를 사용하는 ProgressDialog 예제 96

진행상태바 표시하기
(Showing a progress bar) 95

R

RPC(Remote procedure call) 40

S

Setters 434

Signing 394

Static 434

T

Task 30

TimePickerDialog 87

U

UI 이벤트 62

UI 이벤트 처리 102

이벤트 리스너(Listener) 102

이벤트 핸들러(Handler) 106

터치(Touch) 모드 107

포커스(Focus) 처리 108

V

Virtual 433, 434

Korean

ㄱ

구글 맵 외부 라이브러리 360

그래픽 326

ㄴ

내부 클래스 440

네이티브(Native) 메쏘드 433

네트워크 220

네트워크가 느리다는 것을 가정하라 457

ㄷ

다양한 스크린 해상도에서 작동하도록 UI를 디자인하라 457

다이얼로그(Dialog) 노티피케이션 109, 111

다이얼로그 생성 87

다이얼로그 닫기(Dismissing) 90

다이얼로그 표시하기 88

커스텀(Custom) 다이얼로그 생성 99

단일 액티비티 스크린에 오버로드를 주지 마라 456

데이터를 누락하지 마라 454

데이터베이스 218

데이터 저장공간 216

디바이스 배터리 458

ㄹ

런치(Launch) 모드 33

레이아웃(Layout) 59

레이아웃 선언하기 65

XML 리소스 로드(Load) 68

XML 작성 67

레이아웃 파라미터(Layout Parameters) 70

레이아웃 포지션(Position) 71

애트리뷰트(Attribute) 68

크기(Size), 패딩(Padding) 그리고 마진(Margin) 72

로케이션과 맵 358

로케이션 서비스 358

리소스와 국제화 157

국제화 및 지역화(Internationalization and Localization) 175

리소스 사용(Using Resources) 160

리소스 생성(Creating Resources) 158

소개(Introduction) 157

용 어(Teminology) 171

택일적(Alternate) 리소스 택일하기 165

리소스와 에셋(Asset) 156

리소스 타입 175

간단한 값(Simple Values) 175

드로어블(Drawables) 182

레이아웃(Layout) 191

메뉴(Menus) 189

스타일과 테마(Styles and Themes) 197

애니메이션(Animation) 185

ㅁ

매니페스트 27

메뉴(Menu) 63

메뉴 생성 73

XML로 메뉴 정의하기 79

서브메뉴(Submenu) 73, 78

아이콘 메뉴(Icon Menu) 73

옵션(Option) 메뉴 73, 74

컨텍스트(Context) 메뉴 73, 76

확장 메뉴(Expanded Menu) 73

메뉴 특성(Feature) 81

단일 메뉴 아이템에 인텐트 설정하기 84

동적으로 인텐트 추가하기 85

메뉴 그룹 81

메뉴 아이템 인텐트 84

숏컷(Shortcut, 단축) 키 83

체크가능한(Checkable) 메뉴 아이템 81

무결절성(Seamlessness) 452

무결절성을 위한 설계 452

미디어 포맷 353

T-Mobile G1 미디어 포맷 355

코어 미디어 포맷(Core Media Formats) 354

ㅂ

배포하기 위한 준비 416

버추얼(Virtual) 433, 434

보안 및 퍼미션(Permission) 242

보안 아키텍처 242

부동소수점 441

뷰 계층구조(Hierarchy) 58

브로드캐스트 리시버 24

ㅅ

사용자를 방해하지 마라 455

사용자에게 통보하기 109

상수(Constants) 436

상태바(Status Bar) 노티피케이션 109, 110

상태바(Status Bar) 노티피케이션 생성 115

기초(The Basics) 116

노티피케이션 관리하기(Managing your Notifications) 117

노티피케이션 생성(Creating a Notification) 118

노티피케이션 업데이트하기(Updating the notification)

119
더 많은 기능(More features) 122
발광 빛 추가하기(Adding flashing lights) 122
사운드 추가하기(Adding a sound) 120
진동 추가하기(Adding vibration) 121
커스텀 확장 뷰 생성하기(Creating a Custom Expanded View) 124
서비스 23
성능(Performance) 430
성능을 위한 설계 430
약간의 샘플 성능 숫자들 442
스타일과 테마(Theme) 64, 126
스타일 127
테마(Theme) 128
매니페스트에서의 테마 설정(Set the theme in the manifest) 129
시스템 테마 확장 457
애플리케이션에서의 테마 설정(Set the theme from the application) 130
쓰레드(Thread) 39
쓰레드-세이프 메쏘드(Thread-safe method) 42
쓰레드와 무결절성 455
프로세스와 쓰레드 38

ㅇ

안드로이드가 뷰를 그리는 방법 151
애플리케이션 배포 424
배포 준비 416
애플리케이션 릴리즈 준비 이전 단계 417
애플리케이션의 최종 컴파일 이전 단계 419
애플리케이션 컴파일 이후 단계 422
안드로이드 마켓에 배포 424
애플리케이션 버전 부여 410
애플리케이션 버전 설정 411
최소 시스템 API 버전 지정하기 412
애플리케이션 사인하기(Signing) 243, 394
개인 키 안전하게 하기 407
공개 릴리즈를 위해 사인하기 400
디버그 모드에서 사인하기 398
사인을 위한 기본적 셋업 397
사인 전략 396
애플리케이션 컴포넌트 21
브로드캐스트 리시버(Broadcast receiver) 24
서비스(Service) 23
액티비티(Activity) 21
컨텐트 프로바이더(Content provider) 24
액티비티 21
액티비티 생명주기 43
액티비티와 태스크 30
런치(Launch) 모드 33
스택(Stack) 제거 36
어피니티(Affinity, 친화력)와 신규 태스크 32
태스크 시작 37
어댑터(Adapter) 63
어댑터뷰를 사용한 데이터 바인딩 142
어피니티(Affinity, 친화력)
액티비티 Affinity 263
어피니티와 신규 태스크 32
오디오 및 비디오 348
오디오 및 비디오 재생 348
제트 컨텐트 재생하기 350
오디오 캡쳐 351
오브젝트 생성을 피하라 432
용어집 462
원격 프로시저 호출(Remote procedure call) 40
원시 데이터를 드러내지 마라 454
위젯(Widget) 61
유저 ID와 파일 접근 243
응답성(Responsiveness) 446
응답성 강화하기 449
인터페이스(Interface) 433
인텐트 25
인텐트 오브젝트 201
데이터(Data) 203

액션(Action) 202

인텐트 오브젝트

엑스트라(Extras) 204

카테고리(Category) 203

컴포넌트 이름(Component name) 201

플래그(Flags) 205

인텐트와 인텐트 필터 200

인텐트 필터(Intent filters) 28, 206

인텐트 확정 205

명시적인(Explicit) 인텐트 205

암시적인(Implicit) 인텐트 205

인텐트 매칭 사용하기(Using intent matching) 213

인텐트 필터(Intent filters) 206

데이터 테스트(Data test) 209

액션 테스트(Action test) 207

카테고리 테스트(Category test) 208

일반적 경우(Common cases) 211

일반적인 레이아웃 오브젝트 145

FrameLayout 145

LinearLayout 145

RelativeLayout 148

TableLayout 147

주요(Important) 뷰 그룹 정리 150

ㅈ

정적(Static) 434

ㅋ

커스텀 컴포넌트 제작 131

기본적 접근법(The Basic Approach) 132

완전하게 커스터마이즈된 컴포넌트 133

이미 존재하는 뷰 타입 수정하기 139

합성(Compound) 컨트롤 137

컨텐트 URI 요약 239

컨텐트 프로바이더 24, 222

컨텐트 프로바이더 기초 222

URIs 224

데이터 모델(The data model) 223

컨텐트 프로바이더 데이터 수정 230

레코드 배치 업데이트 233

레코드 삭제 233

레코드 추가(Adding records) 230

새로운 값 추가(Adding new values) 231

컨텐트 프로바이더 생성 234

컨텐트 프로바이더 선언 237

컨텐트 프로바이더 클래스 확장 234

컨텐트 프로바이더 쿼리 225

얻어온 데이터 읽기 228

쿼리는 무엇을 리턴하는가 228

쿼리 만들기(Making the query) 225

컴포넌트 생명주기 43

브로드캐스트 리시버 생명주기 52

서비스 생명주기 49

액티비티 생명주기 43

컴포넌트 종료하기 26

ㅌ

태스크(Task) 30

테마(Theme) 128

토스트(Toast) 노티피케이션 생성 111

기초(The Basics) 112

커스텀 토스트 뷰 생성하기(Creating a Custom Toast View) 113

토스트 위치 부여하기(Positioning your Toast) 113

토스트 노티피케이션(Toast Notification) 109, 110

ㅍ

파일 218

퍼미션(Permission) 242

퍼미션 사용하기 244

퍼미션 선언 및 강제 245

AndroidManifest.xml 에서 퍼미션 강제 247

URI 퍼미션(URI Permissions) 249

그 밖의 퍼미션 강제 249

브로드캐스트 발송 시 보안 강제 249

프레퍼런스 217

프로세스 38

프로세스와 생명주기 53

백그라운드(background) 프로세스 55

비저블(visible) 프로세스 54

서비스(service) 프로세스 54

엠프티(empty) 프로세스 55

포어그라운드(foreground) 프로세스 53

프로세스와 쓰레드 38

쓰레드(Thread) 39

쓰레드-세이프 메쏘드(Thread-safe method) 42

원격 프로시저 호출(Remote procedure call) 40

프로세스 38

Android, It's Morning Again in Mobile.
안드로이드, 모바일에 다시 온 아침.

2007년. 구글의 야심작, Android는
폐쇄적인 모바일 에코 시스템에 새로운 아침을 알렸다.
그 푸른 미명은 전세계 단말기 제조업체와 통신사들을 긴장시켰고,
개발자들을 열광시켰다.
아이폰으로 언급되는 스마트폰의 대표선수를 갈아 치울 수 있는
막강·최강 대항마가 등장했기 때문이었다.
그리고 2009년, 안드로이드를 탑재한 모바일 단말기가
국·내외 곳곳에서 출시될 예정이다.
모바일 단말기 뿐 아니라, 각종 전자제품과 자동차에까지
안드로이드가 탑재될 것이라고 전문가들은 예견하고 있다.

이 책은
기본 중의 기본, 핵심 중의 핵심을
구글이 직접 안내한 **The Developer's Guide**를
꼼꼼하게 읽고 정리한 유일한 책이다.

넓게 펼쳐본 안드로이드 설계도면. 안드로이드로 가는 길.
기본을 이해하고 기술을 이해하게 하는 Basic&Classic Power!!
기본을 충실히 이해하고 기술을 연마하고자 하는 개발자를 위한
안드로이드 안내서가 될 것이다.

더구나 **한국 내 안드로이드 개발자 커뮤니티 No.1 사이트**
kandroid의 운영진이
그 동안 해온 안드로이드 개발자 교육 자료와 노하우를 총망라한
Lecture Note 성격을 띄고 있는 책이기 때문에
개발자들에게는 더더구나 실질적인 도움이 될 것으로 기대한다.

국내 No.1 안드로이드 개발자 커뮤니티 kandroid 운영진 **인베인 임근식** 드림

www.kandroid.org

값 30,000원

ISBN 978-89-962353-0-9