

Алексей Голощапов

Google Android

**Создание приложений
для смартфонов и планшетных ПК**

Санкт-Петербург

«БХВ-Петербург»

2013

УДК 681.3.06
ББК 32.973.26-018.2
Г61

Голощапов А. Л.

Г61 Google Android. Создание приложений для смартфонов и планшетных ПК. — СПб.: БХВ-Петербург, 2013. — 832 с.: ил. — (В подлиннике)

ISBN 978-5-9775-0880-3

Книга посвящена разработке приложений для мобильных устройств и планшетных ПК под управлением операционной системы Google Android. Приведены общие сведения о платформе. Описано создание различных типов приложений и использование системных компонентов и служб Android. Рассмотрено управление сетевыми соединениями и связь через сотовую сеть, мобильный Интернет и Wi-Fi. Уделено внимание использованию графических ресурсов и созданию анимации. Описана разработка пользовательского интерфейса и служб. Показано применение в приложениях сетевых сервисов Google. Описано взаимодействие с аппаратными компонентами мобильного устройства под управлением Android. Книга сопровождается большим количеством примеров, которые располагаются на сайте издательства.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Екатерина Капальгина</i>
Редактор	<i>Екатерина Капальгина</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Марины Дамбиевой</i>

Подписано в печать 31.08.12.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 67,08.

Тираж 1500 экз. Заказ №

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0880-3

© Голощапов А. Л., 2013

© Оформление, издательство "БХВ-Петербург", 2013

Оглавление

- Введение 15**
 - На кого рассчитана эта книга 15
 - Краткое описание глав 17
 - Исходные коды примеров..... 23
 - Благодарности..... 24
- ЧАСТЬ I. ОСНОВЫ ANDROID 25**
 - Глава 1. Архитектура и базовые сведения о платформе Android..... 27**
 - Архитектура Android 27
 - Уровень ядра 28
 - Уровень библиотек 28
 - Dalvik Virtual Machine 29
 - Уровень каркаса приложений..... 30
 - Уровень приложений..... 30
 - Как программировать под Android..... 30
 - Компоненты Android-приложения 31
 - Activity 31
 - Service 31
 - Broadcast Receiver 31
 - Content Provider 32
 - Объекты Intent..... 32
 - Резюме 32
 - Глава 2. Установка и настройка среды разработки..... 35**
 - Создание среды разработки 35
 - Установка JDK 36
 - Установка Eclipse..... 36
 - Установка Android SDK 36
 - Установка Android Development Tools 37
 - Версии SDK и Android API Level 39
 - Обзор Android SDK 40
 - Инструменты для разработки и отладки приложений..... 41
 - Создание переменных окружения 42

Android Virtual Device	43
Конфигурирование AVD	43
Сочетания клавиш	48
Неподдерживаемая функциональность	48
Резюме	49
Глава 3. Первое приложение Android	51
Создание проекта в Eclipse	51
Структура проекта	57
Каталоги ресурсов	58
Подкаталог res/layout/	58
Подкаталоги res/drawable/	60
Подкаталог res/values/	60
Файл R.java	61
Файл окна приложения FirstActivity.java	62
Файл AndroidManifest.xml	63
Общая структура манифеста	64
Структура элемента <code><application></code>	68
Резюме	71
Глава 4. Отладка приложений	73
Отладка в среде Eclipse	73
Использование DDMS	74
Запись в журнал событий	75
Журнал событий Logcat	76
Настройка мобильного устройства Android для отладки приложения	77
Установка режима отладки на мобильном телефоне	77
Установка драйвера USB	78
Взаимодействие устройства Android с DDMS	78
Запуск приложения на мобильном устройстве	79
Резюме	80
ЧАСТЬ II. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ	81
Глава 5. Компоновка элементов управления	83
Формирование графического интерфейса пользователя	83
Создание компоновки	85
XML-файл компоновки	85
Создание компоновки в Layout Editor	87
Типы компоновок	87
<i>FrameLayout</i>	88
<i>LinearLayout</i>	90
<i>TableLayout</i>	95
<i>RelativeLayout</i>	99
Отладка пользовательского интерфейса с помощью Hierarchy Viewer	101
Резюме	104

Глава 6. Базовые виджеты	105
Текстовые поля	105
<i>TextView</i>	106
<i>EditText</i>	110
Тип ввода текста	112
Режимы отображения клавиатуры	114
Полосы прокрутки	119
Отображение графики — <i>ImageView</i>	121
Резюме	123
Глава 7. Командные элементы управления и обработка событий	125
Обработка событий	125
Кнопки и флажки	126
<i>Button</i>	126
<i>RadioButton</i> и <i>RadioGroup</i>	134
<i>CheckBox</i>	136
<i>ToggleButton</i>	139
<i>ImageButton</i>	141
Закладки	143
Динамическое создание элементов управления	147
Резюме	150
Глава 8. Отображение длительно выполняющихся задач	151
Создание фоновых потоков	151
Виджет <i>ProgressBar</i>	152
Расширения класса <i>ProgressBar</i>	155
<i>SeekBar</i>	156
<i>RatingBar</i>	159
Использование системных таймеров и отображение системного времени	162
Резюме	166
Глава 9. Уведомления	167
Всплывающие уведомления	167
Создание уведомлений с заданной компоновкой	170
Резюме	174
Глава 10. Диалоговые окна	175
Типы диалогов	175
Создание диалоговых окон	176
<i>AlertDialog</i>	177
<i>AlertDialog</i> с кнопками	177
Добавление в <i>AlertDialog</i> значка и заголовка	180
<i>AlertDialog</i> со списком	181
<i>AlertDialog</i> с переключателями	183
<i>AlertDialog</i> с флажками	186
<i>ProgressDialog</i>	189
<i>DatePickerDialog</i>	193
<i>TimePickerDialog</i>	197
Создание собственных диалогов	200
Резюме	203

Глава 11. Меню	205
Меню выбора опций	205
Меню со значками	210
Расширенное меню	212
Контекстное меню	215
Подменю	219
Добавление флажков и переключателей в меню	222
Резюме	226
Глава 12. Activity	227
Процессы в системе Android.....	227
Состояния Activity	229
Запуск Activity с использованием объектов Intent	231
Intent-фильтры и запуск заданий	231
Запуск Activity с помощью явного объекта Intent	232
Стек Activity	239
Обмен данными между Activity.....	240
Вызов Activity из другого приложения	248
Вызов стандартных Activity	250
Резюме	253
Глава 13. Доступ к компонентам через разрешения	255
Вызов Activity с использованием разрешений	255
Установка разрешений в файле манифеста	256
Резюме	259
Глава 14. Фрагменты.....	261
Классы фрагментов	261
Создание фрагментов	262
Динамическое добавление фрагментов	266
Передача данных между фрагментами	269
<i>DialogFragment</i>	272
Резюме	275
ЧАСТЬ III. РЕСУРСЫ, ГРАФИКА И ОБРАБОТКА ДАННЫХ.....	277
Глава 15. Использование ресурсов	279
Доступные типы ресурсов.....	279
Создание ресурсов	280
Ссылки на ресурсы	281
Использование ресурсов в коде программы.....	281
Загрузка простых типов из ресурсов	282
Загрузка файлов произвольного типа.....	287
Создание меню в XML	289
Загрузка XML-документов.....	293
Стили и темы.....	296
Стили.....	296
Темы.....	297
Определение собственных стилей и тем.....	298

Активы	300
Резюме	304
Глава 16. Файловая система и карта памяти	305
Подключение карты памяти в эмуляторе	305
Файловая система Android	305
Стандартные директории Android	307
Проверка состояния карты памяти	310
Чтение и запись файлов	310
Сохранение и чтение файлов с SD-карты	315
Резюме	324
Глава 17. Адаптеры данных и компоненты для отображения данных	325
Отображение текстовых данных в списках	325
Адаптеры данных	326
<i>ListView</i>	328
Загрузка нескольких источников данных в список	331
Список с единичным и множественным выбором	334
Создание списка с нестандартной компоновкой	336
<i>ListFragment</i>	339
<i>GridView</i>	343
Отображение графики в списках	347
Отображение графики в <i>GridView</i>	347
Загрузка изображений и текста в список	350
<i>Gallery</i>	353
<i>SlidingDrawer</i>	356
Выпадающий список	361
Текстовые поля с автозаполнением	364
<i>AutoCompleteTextView</i>	364
<i>MultiAutoCompleteTextView</i>	367
Резюме	369
Глава 18. База данных SQLite	371
Встроенные базы данных в Android	371
Инструменты для работы с базами данных на Android-телефоне	373
Инструмент <i>sqlite3</i>	373
Использование инструментов сторонних разработчиков для работы с SQLite	374
Создание базы данных: класс <i>SQLiteOpenHelper</i>	375
Резюме	382
Глава 19. Content Provider	383
Создание компонента Content Provider	383
Расширение класса <i>ContentProvider</i>	383
URI	385
Управление базой данных из приложения	386
Чтение данных	387
Добавление записей	388
Обновление записей	389
Удаление записей	389

Декларирование компонента Content Provider в файле манифеста приложения	390
Запросы к Content Provider.....	391
Чтение возвращаемых значений.....	391
Позиционирование курсора	392
Добавление записей	393
Изменение записи	393
Удаление записей.....	393
Клиентское приложение для работы с базой данных	394
Резюме	401
Глава 20. Сохранение пользовательских настроек	403
Пользовательские настройки в Android	403
Доступ к настройкам	404
<i>CheckBoxPreference</i>	405
<i>EditTextPreference</i>	411
<i>ListPreference</i>	413
<i>RingtonePreference</i>	415
<i>PreferenceCategory</i>	417
<i>PreferenceScreen</i>	418
Резюме	421
Глава 21. Локализация приложений	423
Ресурсы, заданные по умолчанию.....	423
Создание локализованных ресурсов	423
Резюме	426
Глава 22. Графика	427
Объект <i>Drawable</i>	427
Создание объектов <i>Drawable</i> в коде программы	428
Класс <i>TransitionDrawable</i>	429
Класс <i>ShapeDrawable</i>	432
Рисование на канве	438
Резюме	441
Глава 23. Создание анимации	443
Tween Animation	443
Создание анимации в XML-файле.....	444
Элемент <i><set></i>	445
Элемент <i><alpha></i>	446
Элемент <i><scale></i>	446
Элемент <i><translate></i>	446
Элемент <i><rotate></i>	446
Анимация для графических примитивов	447
Анимация для графических файлов	453
Анимация для группы объектов	456
Frame Animation	461
Создание анимации в XML	461
Создание анимации в коде программы	464
Резюме	467

ЧАСТЬ IV. СИСТЕМНЫЕ СЛУЖБЫ.....	469
Глава 24. Компонент Service	471
Работа служб в Android	471
Создание службы	472
Вызов службы	473
Доступ к системным и сетевым сервисам	477
Резюме	480
Глава 25. Broadcast Receiver.....	481
Класс <i>BroadcastReceiver</i>	481
Прослушивание событий компонентом Broadcast Receiver	483
Пример приложения с Broadcast Receiver	484
Пример приложения-передатчика события	486
Резюме	487
Глава 26. Home Screen	489
Обои для домашнего экрана	489
Виджеты для домашнего экрана	491
Создание виджета	491
Установка виджета	495
Удаление виджета.....	495
Работа с классом <i>AppWidgetProvider</i>	495
Резюме	499
Глава 27. Уведомления в строке состояния.....	501
Менеджер уведомлений	501
Создание уведомления	502
Резюме	507
Глава 28. Action Bar	509
Управление видимостью Action Bar	509
Добавление опций меню в Action Bar.....	512
Добавление текста в меню	515
Резюме	516
Глава 29. Служба оповещений.....	517
Менеджер оповещений	517
Использование оповещений	518
Резюме	524
Глава 30. Буфер обмена и API для работы с текстом.....	525
Менеджер буфера обмена	525
Синтез речи на основе текста	528
Резюме	533
ЧАСТЬ V. СЕТЕВЫЕ СЕРВИСЫ	535
Глава 31. Получение информации о телефоне и сети сотовой связи	537
Информация о телефоне	537
Определение типа телефона и сети сотовой связи.....	537

Определение базовой станции сотовой связи	539
Определение состояния вызова	540
Получение информации о роуминге	540
Использование класса <i>TelephonyManager</i>	540
Доступ к SIM-карте	546
Состояние SIM-карты	547
Доступ к SIM-карте из приложения	547
Перехват изменений состояния параметров телефона	549
Запуск и остановка прослушивания изменений состояния сотовой сети	550
Изменение уровня сигнала	551
Изменение базовой станции сотовой связи	552
Мониторинг состояния подключения к сервису	552
Приложение для прослушивания изменений состояния сотовой сети	553
Использование эмулятора для тестирования приложений	557
Резюме	560
Глава 32. Обработка телефонных вызовов	561
Использование эмулятора для тестирования обработки телефонных вызовов	561
Имитация телефонного вызова из DDMS	561
Имитация телефонного вызова между двумя эмуляторами Android	562
Установка разрешений	564
Использование объектов <i>Intent</i> для создания телефонных вызовов	564
Вызов телефонного абонента из приложения	565
Перехват исходящих звонков	569
Резюме	572
Глава 33. Отправка и получение SMS	573
Использование эмулятора для отправки SMS	573
Отправка SMS из приложения	574
Отправка SMS с данными	576
Деление SMS на фрагменты	576
Установка разрешений для работы SMS	576
Приложение для отправки SMS	577
Структура SMS-сообщения	581
Перехват входящих SMS-сообщений приложением	582
Хранение SMS на мобильном устройстве	585
Доступ к каталогам SMS	585
Доступ к полям SMS-сообщения	590
Резюме	593
Глава 34. Мобильный Интернет	595
Создание сетевых соединений	595
Менеджер сетевых соединений	595
Характеристики мобильной сети	596
Получение информации о сети в приложении	596
Мониторинг сетевого трафика	599
Получение информации о трафике	599
Приложение для мониторинга сетевого трафика	600
Встроенный браузер	601
Виджет <i>WebView</i>	602

Использование виджета <i>WebView</i>	602
Загрузка данных в виджет <i>WebView</i>	605
Сохранение пользовательских настроек.....	606
Резюме.....	615
Глава 35. Управление Wi-Fi-соединениями.....	617
Управление соединением Wi-Fi.....	617
Менеджер Wi-Fi-соединений.....	617
Разрешения.....	618
Состояние соединения.....	618
Отслеживание состояния соединения.....	618
Управление подключением Wi-Fi и отслеживание состояния соединения из приложения.....	620
Управление настройками Wi-Fi-соединения.....	625
Характеристики соединения.....	627
IP-адресация.....	627
Получение информации о сети Wi-Fi в приложении.....	628
Конфигурация Wi-Fi-соединения.....	632
Сканирование точек доступа.....	636
Мониторинг уровня сигнала и скорости передачи данных в приложении.....	641
Резюме.....	645
Глава 36. Определение местоположения.....	647
Использование Google API в эмуляторе.....	647
Сервисы и провайдеры местоположения.....	647
Типы провайдеров местоположения.....	648
Разрешения для работы с провайдерами местоположения.....	650
Приложение для поиска доступных провайдеров.....	650
Определение лучшего провайдера.....	652
Критерии для определения лучшего провайдера.....	652
Поиск и определение лучшего провайдера в приложении.....	654
Использование эмулятора Android для тестирования приложений.....	656
Определение координат.....	658
Обновление местоположения.....	658
Приложение для мониторинга изменений координат и состояния провайдера.....	660
Резюме.....	662
Глава 37. Сервис Geocoding.....	663
Использование Geocoding.....	663
Reverse Geocoding.....	664
Отображение местоположения на карте.....	668
Forward Geocoding.....	672
Резюме.....	677
Глава 38. Использование карт Google Maps в приложениях.....	679
Получение ключа Maps API Key.....	679
Базовые классы.....	681
Виджет <i>MapView</i>	682
Класс <i>MapActivity</i>	683

Класс <i>MapController</i>	683
Класс <i>GeoPoint</i>	684
Использование <i>MapView</i> в приложении	685
Управление масштабированием карты	690
Добавление маркера	691
Изменение масштаба карты с помощью виджета <i>SeekBar</i>	692
Резюме	698

ЧАСТЬ VI. РАБОТА С ОБОРУДОВАНИЕМ..... 699

Глава 39. Использование видеокамеры..... 701

Работа с камерой в приложении	701
Параметры камеры	702
Получение параметров камеры в приложении	702
Поддержка различных режимов камерой	704
Использование объектов <i>Intent</i> для открытия камеры	708
Встраивание камеры в приложения	712
Управление работой камеры	715
Добавление оверлеев	719
Захват изображения	723
Использование автофокуса	728
Резюме	732

Глава 40. Встроенные датчики..... 733

Библиотека для работы с датчиками	733
Управление датчиками	733
Поиск доступных датчиков на мобильном устройстве	735
Отслеживание изменений, измеряемых датчиками значений	737
Работа с датчиками в приложении	739
Датчик освещенности	739
Датчик расстояния	743
Датчик ориентации	744
Акселерометр	749
Датчик уровня магнитного поля	753
Другие датчики, доступные на мобильных устройствах Android	754
Имитация работы сенсоров для эмулятора Android	755
Резюме	756

Глава 41. Управление дисплеем..... 757

Программный доступ к дисплею	757
Менеджер окон	757
Параметры дисплея мобильного устройства	757
Управление яркостью экрана	760
Резюме	765

Глава 42. Доступ к аккумуляторной батарее..... 767

Менеджер источника питания	767
Отображение статистики использования батареи	775
Резюме	776

Глава 43. Управление энергопотреблением телефона.....	779
Менеджер энергопотребления.....	779
Управление энергопотреблением и блокировки.....	779
Резюме	785
Глава 44. Получение информации о системе.....	787
Класс <i>ActivityManager</i>	787
Информация о конфигурации устройства	793
Информация о системе.....	797
Доступная память устройства.....	797
Выполняющиеся процессы	798
Выполняющиеся службы	800
Выполняющиеся задания	801
Последние выполненные задания.....	802
Процессы в состоянии ошибки.....	804
Терминал в системе Android.....	806
Резюме	812
Приложение. Описание электронного архива и установка примеров	813
Электронный архив	813
Установка примеров	813
Предметный указатель	817

Введение

На момент написания этой книги платформа Android уже представляет собой заметное явление в области программного обеспечения для мобильных устройств. Новой платформой заинтересовались ведущие мировые производители мобильной электроники и сотовые операторы, а многие из них уже выпускают большой ассортимент мобильных устройств, работающих под управлением Android.

В чем же заключается уникальность платформы Android? Основная идея Google состоит в том, что компания предлагает в открытый доступ исходные коды своей операционной системы, предоставляет набор удобных инструментов для разработки и хорошо документированный комплект SDK, что должно со временем привести к появлению большого количества программного обеспечения для этой платформы. За несколько лет Android стал самым успешным проектом для мобильных телефонов. Android захватывает рынок мобильных телефонов, постепенно вытесняя с него общепризнанных лидеров. Система Android устанавливается теперь не только на смартфоны, данная платформа была адаптирована для планшетов и нетбуков.

Большим шагом в развитии Google Android стало открытие в октябре 2008 года онлайн-магазина приложений — Android Market, в котором можно приобрести программы и другой софт для устройств на базе новой платформы. Кроме того, для разработчиков программного обеспечения появилась возможность брать плату за свои приложения в Android Market, что делает разработку приложений под эту платформу еще более привлекательной.

На кого рассчитана эта книга

Поскольку эта книга о программировании приложений для мобильных устройств на платформе Android, необходимое условие для работы с книгой — наличие базовых навыков программирования на языке Java, который нужен для написания приложений с использованием Android SDK.

Книга предназначена в первую очередь для разработчиков на Java, уже имеющих опыт программирования для мобильных устройств на платформе Android. Читатель должен иметь представление об архитектуре системы, ее фундаментальных компонентах и их взаимодействии между собой и, конечно, иметь опыт создания приложений для Android в интегрированной среде разработки Eclipse.

Для тех читателей, которые не работали до этого момента на Java, но использовали другие объектно-ориентированные языки (типа C#.NET), переход на платформу Android также не вызовет больших затруднений. Таким образом, отсутствие опыта программирования в Java не будет большим недостатком при работе с книгой и освоении платформы Android. Необходимые навыки для программирования на Java можно приобрести постепенно, параллельно с изучением платформы Android.

В целом эта книга предназначена для двух разных групп программистов:

- ❑ традиционных разработчиков программного обеспечения, которые имеют опыт работы на языках Java или C#.NET и желают перейти на разработку приложений для мобильных телефонов на базе ОС Android;
- ❑ разработчиков, уже имеющих опыт программирования мобильных устройств на iPhone, Windows Mobile, Symbian и Java ME, которые хотят программировать на платформе Android.

По мере изучения книги вы будете создавать приложения, принимающие и отправляющие SMS-сообщения, телефонные звонки, управляющие сетевыми соединениями мобильного телефона, использующие возможности систем глобального позиционирования и сетевых сервисов Google для определения местоположения. Вы сможете управлять из программного кода "железом" мобильного телефона: встроенными датчиками, картой памяти, видеокамерой, процессором и дисплеем, а также использовать в своих приложениях много другой полезной и интересной функциональности, предоставляемой мобильным телефоном с системой Android.

Еще одно требование, предъявляемое к читателю, — наличие мобильного телефона с системой Android, однако он потребуется при изучении второй половины книги. Где большинство примеров приложений будет рассчитано на использование в реальном мобильном устройстве. Конечно, там где есть возможность, для тестирования и запуска приложений используется эмулятор Android, но все же одного эмулятора для работы с приложениями, рассматриваемыми в этой книге, будет недостаточно.

Примеры приложений рассчитаны на версию Android 4.0.3 (API Level 15). Однако совсем необязательно иметь устройство с версией Android 4.0.3. Можно использовать и более ранние версии, например Android 2.1 и даже Android 1.6, но необходимо учитывать, что при перекомпиляции приложений в более раннюю версию Android, возможно, придется из кода приложения убрать некоторую функциональность, предоставляемую библиотеками Android SDK, которая стала доступной только в поздних версиях системы.

Желательно, чтобы эта книга была полезной и ценной любому человеку, заинтересованному в разработке приложений для Android. Люди, увлеченные программированием, найдут здесь основу для своих будущих приложений. Прикладные программисты изучат основные функциональные возможности платформы, которые смогут использовать в своих профессиональных разработках. Короче говоря, эта книга содержит много информации, которая пригодится вам независимо от вашего опыта и профессиональных интересов.

Краткое описание глав

Книга состоит из 6 частей, которые содержат 44 главы, и одного приложения. Далее приводится краткое описание каждой из глав.

□ Часть I. Основы Android

В этой части рассказывается о настройке среды разработки и мобильного телефона для тестирования приложений, даются общие сведения о системных компонентах и службах Android, с которыми мы будем работать на протяжении этой книги.

- *Глава 1. Архитектура и базовые сведения о платформе Android*

Описывается архитектура и программный интерфейс операционной системы Android. Приводится информация о составе и функциональных возможностях библиотек Android, базовых классах и интерфейсах, входящих в состав библиотек и пакетов Android SDK. Дается понятие программного стека Android, принципы работы Dalvik Virtual Machine. Приводятся базовые понятия об основных компонентах Android-приложений — Activity, Service, Broadcast Receiver и Content Provider.

- *Глава 2. Установка и настройка среды разработки*

Глава посвящена установке на компьютер необходимого программного обеспечения, требуемого для разработки приложений под Android: Java Development Kit, Eclipse, Android SDK, Android Development Tools, а также настройке среды разработки для написания программ для Android. Описывается инструментарий, входящий в состав Android SDK, — различные средства отладки, компоновки, упаковки и инсталляции приложений на эмулятор и мобильное устройство. Приводятся инструкции по конфигурации и работе с Android Virtual Device — эмулятором мобильного устройства.

- *Глава 3. Первое приложение Android*

Рассматривается создание первой программы под Android, запуск и работа программы в эмуляторе мобильного устройства. Будет детально изучена структура проекта, содержимое файлов проекта и работа с ними в интегрированной среде разработки Eclipse. Рассматривается внутренняя архитектура файла манифеста Android-приложения, который предоставляет основную информацию о компонентах приложения и требуемых разрешениях для взаимодействия с системой.

- *Глава 4. Отладка приложений*

Рассматривается использование интегрированной среды разработки Eclipse и инструментов из состава Android SDK для отладки и тестирования приложений. Описываются конфигурирование и подключение мобильного устройства к компьютеру для организации взаимодействия устройства с инструментами, входящими в состав Android SDK, и со средой разработки Eclipse.

□ Часть II. Графический интерфейс пользователя

Здесь мы будем учиться проектировать графический интерфейс пользователя приложения — общую компоновку элементов управления на экране. Мы изучим работу с основными элементами пользовательского интерфейса, обработку событий пользовательского взаимодействия, создание меню, уведомлений и диалоговых окон.

Также мы рассмотрим приложения с несколькими окнами и организацию взаимодействия между окнами в таких приложениях.

- *Глава 5. Компоновка элементов управления*

Эта глава дает базовые понятия о графическом интерфейсе Android и знакомит с принципами экранной иерархии элементов графического интерфейса. Рассматриваются вопросы компоновки экранных элементов (в Android их принято называть виджетами) и создания разметки для окон приложений, которые читатель будет разрабатывать и использовать в следующих главах для создания профессионального пользовательского интерфейса в своих приложениях.

- *Глава 6. Базовые виджеты*

Глава знакомит читателя с основными элементами графического интерфейса пользователя — текстовыми полями и виджетами для отображения графики. Эти виджеты являются базовыми для большинства остальных элементов управления пользовательским интерфейсом Android-приложения.

- *Глава 7. Командные элементы управления и обработка событий*

В этой главе будет рассмотрено использование командных кнопок, флажков и переключателей, а также обработка событий, возникающих при взаимодействии пользователя с приложением. Затем мы опишем контейнерные виджеты для группировки переключателей и создания закладок.

- *Глава 8. Отображение длительно выполняющихся задач*

В этой главе мы рассмотрим индикаторы, слайдеры и компоненты отображения времени. Некоторые операции требуют для своего выполнения длительного времени, и для отображения степени завершенности этих операций (например, загрузка файла из сети) обычно используются индикаторы. Такие операции требуется выполнять в фоновом потоке, поэтому в данной главе мы также рассмотрим создание фоновых потоков в приложении.

- *Глава 9. Уведомления*

Рассматривается создание и вызов уведомлений из приложения. При работе пользователя с приложением могут возникать различные ситуации, о которых необходимо уведомить пользователя. Читатель познакомится с созданием механизма оповещения пользователя приложения.

- *Глава 10. Диалоговые окна*

В данной главе рассказывается о создании и использовании диалоговых окон для Android. Диалоги обычно применяются для сообщений и коротких действий, которые непосредственно касаются событий, возникающих в процессе работы приложения. Помимо использования стандартных диалогов читатель научится разрабатывать собственный дизайн диалоговых окон.

- *Глава 11. Меню*

Android SDK предлагает обширную поддержку меню в приложениях. Читатель научится работать с несколькими типами меню, поддерживаемых Android, включая контекстные меню, меню с иконками, всплывающие меню и альтернативные меню, сможет встраивать меню в свои приложения.

- *Глава 12. Activity*

Рассматривается управление и взаимодействие Activity — окон приложения. Дается представление о жизненном цикле Activity в Android-приложении и стеке Activity. Обсуждаются способы обмена данными между Activity. Также рассматривается одна из интересных функциональностей Android — Intent, которая обеспечивает динамическое связывание между компонентами приложений. Вместо статического соединения программного кода в Android используется система обмена сообщениями, которая выполняет позднее связывание (late bound).

- *Глава 13. Доступ к компонентам через разрешения*

В системе Android доступ почти ко всем системным компонентам ограничен для внешних приложений. Поэтому если приложению требуется использовать какую-либо системную службу, например необходимо послать сообщение SMS, подключиться к мобильному Интернету или к удаленному сетевому сервису, нужно задать для приложения соответствующие разрешения.

- *Глава 14. Фрагменты*

Фрагмент представляет часть пользовательского интерфейса в Activity и позволяет строить более гибкий пользовательский интерфейс. По сути, они представляют собой части Activity. Можно объединять несколько фрагментов в одном Activity и повторно использовать эти же фрагменты в других Activity.

■ Часть III. Ресурсы, графика и обработка данных

В этой части мы будем работать с различными типами ресурсов, доступными на платформе Android, и использованием встроенной базы данных SQLite для создания приложений для работы с данными.

- *Глава 15. Использование ресурсов*

Ресурсы и активы — это неотъемлемая часть любого Android-приложения. Мы рассмотрим способы доступа Android-приложения к различным типам ресурсов, находящихся во внешних файлах. Здесь мы будем учиться загружать в программу изображения, строки, разметки, стили, темы, XML-документы и т. д.

- *Глава 16. Файловая система и карта памяти*

Рассматривается организация файловой системы на карте памяти и мобильном устройстве, создание каталогов, сохранение и чтение файлов с карты памяти. В этой главе даются сведения о получении доступа к карте памяти, информации о ее состоянии.

- *Глава 17. Адаптеры данных и компоненты для отображения данных*

Отображение данных различного типа необходимо для большинства приложений. В этой главе рассматриваются виджеты-списки, выводящие на экран текстовую и графическую информацию, которая может быть связана с внутренним или внешним источником данных, и адаптеры данных — компоненты-посредники между набором данных и элементом пользовательского интерфейса для их отображения.

- *Глава 18. База данных SQLite*

В данной главе рассматривается база данных SQLite. В этой главе читатель научится создавать базы данных, заполнять их данными и управлять ими из кода клиентского приложения.

- *Глава 19. Content Provider*

Компонент Content Provider (Контент-провайдер) служит удобным механизмом для сохранения и обмена данными между приложениями, а также может создавать контент-провайдеры, которые способны управлять данными — добавлять, удалять и модифицировать данные любых других приложений (если они предоставляют соответствующие разрешения) из своего приложения.

- *Глава 20. Сохранение пользовательских настроек*

Рассматривается механизм предпочтений — сохранение пользовательских настроек приложения, а также чтение и запись файлов и управление файловым вводом-выводом из приложения.

- *Глава 21. Локализация приложений*

В этой главе рассматривается создание локализованных приложений. Android-приложение может работать на многих устройствах во многих регионах мира и должно соответствовать настройкам и языкам того региона, где оно будет использоваться.

- *Глава 22. Графика*

Обсуждаются различные варианты использования графических ресурсов в Android-приложении. Рассматривается рисование графики и загрузка графики из ресурсов или XML-документов при создании интерфейсов. Примеры приложений в этой главе показывают использование собственной графики и анимации в приложениях с применением API-библиотек для работы с графикой.

- *Глава 23. Создание анимации*

Рассматривается создание анимации для разработки визуально привлекательных Android-приложений с использованием возможностей Android SDK, который предоставляет двумерную графическую библиотеку анимации.

□ **Часть IV. Системные службы**

В этой части даются сведения о системных службах Android. Их широкие функциональные возможности можно применять в собственных приложениях. Здесь описывается работа с различными системными сервисами платформы Android, предназначенными для управления и мониторинга работы системы, отправкой уведомлений для интерактивного взаимодействия с пользователем, а также служб, предоставляющих различную дополнительную функциональность для пользователя мобильного телефона.

- *Глава 24. Компонент Service*

Рассматривается создание компонента Service (Служба), который позволяет приложению работать в фоновом режиме без использования интерфейса пользователя. Описываются общие принципы доступа приложений к системным компонентам, локальным службам на платформе Android и сетевым сервисам, предоставляемым Google для пользователей мобильных телефонов.

- *Глава 25. Broadcast Receiver*

Эта глава научит созданию компонента Broadcast Receiver для приема и обработки событий, происходящих в системе. Broadcast Receiver используется, когда

возникает необходимость в том, чтобы приложение или служба реагировали на внешние события, которые могут инициализировать другие приложения и службы.

- *Глава 26. Home screen*

Здесь мы рассмотрим управление домашним экраном устройства Android — программную смену обоев и создание виджетов для домашнего экрана — специальных программ, отображаемых и запускаемых пользователем из домашнего экрана устройства.

- *Глава 27. Уведомления в строке состояния*

В этой главе описывается использование менеджера уведомлений для создания приложений, осуществляющих интерактивное взаимодействие с пользователем мобильного устройства.

- *Глава 28. Action Bar*

Action Bar — это новая функциональность, появившаяся в Android 3.0 и предназначенная для планшетных ПК. Вместо традиционной строки заголовка, расположенной наверху экрана устройства, Action Bar выводит на экран значок приложения вместе с заголовком действия и может также отображать ярлыки для пунктов меню приложения.

- *Глава 29. Служба оповещений*

В данной главе рассматривается использование менеджера оповещений. На основе этой службы можно создавать различные планировщики заданий, органайзеры, будильники, таймеры и другие приложения для оповещения пользователя в заданные моменты времени.

- *Глава 30. Буфер обмена и API для работы с текстом*

Глава посвящена использованию менеджера системного буфера обмена в приложениях для копирования и вставки данных, а также применению Text To Speech — библиотеки для синтеза речи на основе текста.

□ **Часть V. Сетевые сервисы**

Эта часть описывает работу с основными функциями мобильного телефона Android и сетями сотовой связи: управление телефонными вызовами, отправка и получение SMS-сообщений. Рассматривается работа с мобильным Интернетом и сетями Wi-Fi, провайдерами местоположения и сетевыми сервисами Google Maps и Geocoding.

- *Глава 31. Получение информации о телефоне и сети сотовой связи*

В этой главе даются базовые понятия о типах сетей сотовой связи. Рассматривается использование менеджера телефонных соединений для получения информации о сети сотовой связи и операторе, для управления и конфигурирования сети.

- *Глава 32. Обработка телефонных вызовов*

В этой главе даются сведения о том, как производить телефонные вызовы непосредственно из кода приложения. Рассматривается создание служб, работающих в фоновом режиме, и приложений для отслеживания телефонных вызовов на мобильном устройстве.

- *Глава 33. Отправка и получение SMS*

В этой главе рассматриваются возможности приложений, работающих с SMS-сообщениями. Система Android предоставляет полный доступ к функциональным возможностям SMS, позволяя отправлять и получать SMS-сообщения из приложений.

- *Глава 34. Мобильный Интернет*

В этой главе рассматриваются подключение и конфигурация сети мобильного Интернета на мобильном устройстве. Описывается мониторинг сетевого трафика из приложения. Рассматривается функциональность встроенного веб-браузера WebKit для использования его в собственных приложениях.

- *Глава 35. Управление Wi-Fi-соединениями*

В этой главе даются сведения об организации сети Wi-Fi и взаимодействие с ними мобильного устройства. Рассматривается создание приложений, которые могут осуществлять сканирование точек доступа и подключение к ним мобильного устройства, читать характеристики сети, отслеживать изменение уровня сигнала и конфигурацию сетей Wi-Fi.

- *Глава 36. Определение местоположения*

Глава посвящена работе с менеджером местоположения — Location Manager для определения и отслеживания изменений текущих координат пользователя мобильного устройства. Рассматриваются также поиск доступных провайдеров местоположения, автоматическое определение лучшего провайдера, подключение и использование провайдера приложением.

- *Глава 37. Сервис Geocoding*

В этой главе описывается Geocoding — сетевая служба, которая выполняет сопоставление географических координат карты и физического адреса объекта. С этой службой можно взаимодействовать из приложения, установленного на мобильном устройстве.

- *Глава 38. Использование карт Google Maps в приложениях*

В этой главе рассматривается использование внешних библиотек Google API в своих приложениях, а также получение и регистрация Maps API Key для возможности встраивания карт Google Maps в приложение, работа с виджетом MapView — специализированным виджетом для навигации и отображения местоположения пользователя на карте.

■ Часть VI. Работа с оборудованием

Эта часть описывает доступ к аппаратной платформе мобильного устройства и управление встроенным оборудованием: картой памяти, видеокамерой, датчиками, аккумуляторной батареей и другим "железом" мобильного телефона.

- *Глава 39. Использование видеокамеры*

Описываются доступ к видеокамере мобильного устройства, использование и управление камерой в приложениях: управление автофокусом, масштабирование изображений, сохранение видео и фотографий.

- *Глава 40. Встроенные датчики*

В этой главе даются сведения о встроенных датчиках, доступных в системе Android. Набор встроенных датчиков зависит от модели мобильного устройства. Телефон, в зависимости от производителя и модели, может иметь встроенные датчики температуры, давления, ориентации, ускорения, измерители уровня освещенности, магнитного поля и др. Здесь читатель научится создавать приложения, получающие доступ к встроенным датчикам мобильного устройства, определяющие их наличие на устройстве и использующие их возможности.

- *Глава 41. Управление дисплеем*

В этой главе рассматривается получение и использование информации о дисплее мобильного устройства — получение метрик и физических свойств дисплея для адаптации приложения на конкретной модели мобильного устройства. Также рассматривается управление яркостью дисплея из программного кода.

- *Глава 42. Доступ к аккумуляторной батарее*

В этой главе рассматриваются использование менеджера источника питания для доступа к аккумуляторной батарее, получение приложением информации о состоянии источника питания — уровне напряжения, подключении зарядного устройства, температуре и других параметрах.

- *Глава 43. Управление энергопотреблением телефона*

В этой главе рассматривается использование менеджера энергопотребления мобильного телефона. Также даются сведения о возможностях применения блокировок для управления процессором, подсветкой экрана и клавиатуры мобильного устройства с целью уменьшения потребления энергии мобильным телефоном.

- *Глава 44. Получение информации о системе*

В этой главе рассматриваются способы получения информации о системе Android: выполняющихся в данный момент процессах, службах и заданиях, потреблении и доступности памяти. Также рассматриваются использование встроенного терминала Android, вызов системных команд из приложений для управления системой и журналирование системных событий.

Исходные коды примеров

Архив с материалами к книге выложен на FTP сайте издательства. В архиве даны все исходные коды примеров, приведенных в книге. Установка примеров описана в *приложении*. Все примеры используют функциональность Android SDK версии 4.0.3 и скомпилированы в среде Eclipse. Установка и настройка среды разработки подробно описана в *главе 2*. В архиве также находятся файлы ресурсов — графика, иконки, шрифты, используемые в примерах приложений, приведенных в книге.

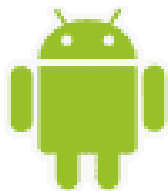
Книга содержит полные исходные коды всех программ, имеющихся в архиве, однако некоторые листинги программ для экономии места и во избежание ненужного дублирования информации содержат только изменения программного кода относительно предыдущих листингов. Такое сокращение позволяет не только экономить место в кни-

ге, но и улучшить понимание программного кода, делая акцент только на новой функциональности.

Все примеры приложений сделаны по возможности компактными и реализуют только конкретную функциональность, рассматриваемую в данной теме. После того как вы прочитаете книгу, вы можете использовать код примеров как справочную информацию и ресурс при создании собственных приложений для Android. Примеры приложений в архиве структурированы по темам и позволяют, при необходимости, легко отыскать реализацию нужной вам функциональности.

Благодарности

В первую очередь хочу поблагодарить своих родных и близких за оказанную моральную поддержку в процессе написания книги, а также всех сотрудников издательства "БХВ-Петербург", которые помогали мне в ее создании и без помощи которых эта книга не увидела бы свет и не попала бы в руки читателей.



ЧАСТЬ I

Основы Android

Глава 1. Архитектура и базовые сведения о платформе Android

Глава 2. Установка и настройка среды разработки

Глава 3. Первое приложение Android

Глава 4. Отладка приложений



ГЛАВА 1

Архитектура и базовые сведения о платформе Android

Перед тем как приступить к разработке приложений для Android, хотелось бы вкратце познакомить читателя с архитектурой системы и основными особенностями этой платформы.

Архитектура Android

Платформа Android представляет собой программный стек для мобильных устройств, который включает операционную систему, программное обеспечение промежуточного слоя (middleware), а также основные пользовательские приложения, входящие в состав мобильного телефона (или КПК), такие как календарь, браузер, базы данных контактов, сообщений SMS и др.

Архитектуру Android принято делить на четыре уровня:

- уровень ядра;
- уровень библиотек и среды выполнения;
- уровень каркаса приложений;
- уровень приложений.

На рис. 1.1 показаны основные компоненты операционной системы Android и их взаимодействие.

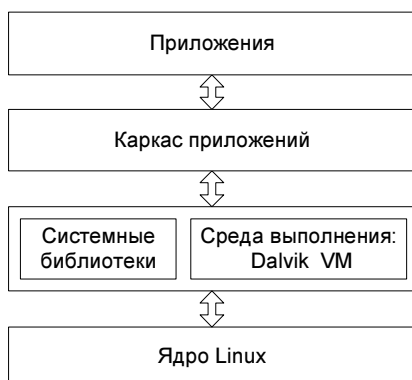


Рис. 1.1. Архитектура системы Android

Уровень ядра

Ядро является слоем абстракции между оборудованием и остальной частью программного стека. На этом уровне располагаются основные службы типа управления процессами, распределения памяти и управления файловой системой.

Android основан на ядре Linux версии 2.6, но сама система Android не является Linux-системой в чистом виде. Система Android имеет некоторые отличия и содержит дополнительные расширения ядра Linux, специфичные для Android, — свои механизмы распределения памяти, взаимодействие между процессами и др.

Приложения и службы могут работать в защищенных отдельных процессах, которые должны общаться между собой и иметь доступ к общим данным. Платформа Android поддерживает механизм IPC (Inter-Process Communication), который является основным механизмом взаимодействия процессов. Драйвер IPC обеспечивает взаимодействие процессов, создание и обработку пулов потоков в процессах, подсчет и отображение ссылок на объекты в других процессах и синхронные запросы между процессами.

Поскольку Android является платформой для мобильных устройств и должна обеспечивать экономный расход аккумуляторной батареи телефона, важную роль выполняет система управления энергопотреблением — Android Power Management. Она разработана на основе стандартного драйвера управления питанием Linux, но оптимизирована для мобильных устройств с учетом их специфических особенностей. Драйвер переводит систему в "спящий режим" с минимальным потреблением мощности процессором, если приложения и службы не используются.

Программный стек Android разработан с учетом необходимой гибкости, включая работу со многими дополнительными компонентами, имеющимися в мобильных устройствах. Эти компоненты в значительной степени полагаются на доступность определенных аппаратных средств на данном устройстве. Они предоставляют дополнительную функциональность для мобильных устройств (сенсорный экран, камера, GPS, акселерометр и т. д.).

На этом уровне также расположен набор драйверов для обеспечения работы с оборудованием мобильного устройства. Набор драйверов может отличаться в зависимости от производителя и модели устройства. Поскольку новое вспомогательное оборудование для мобильных устройств постоянно появляется на рынке, драйверы для них должны быть написаны на уровне ядра Linux для обеспечения поддержки оборудования, так же как и для настольных Linux-систем.

Преимущество использования ядра Linux как основы Android в том, что ядро системы позволяет верхним уровням программного стека оставаться неизменными, несмотря на различия в используемом оборудовании. Конечно, хорошая практика программирования требует, чтобы пользовательские приложения корректно завершали свою работу в случае вызова ресурса, являющегося недоступным, например встроенной видекамеры или сенсора, не присутствующего в данной модели телефона.

Уровень библиотек

Следующий уровень над ядром Linux включает набор библиотек C/C++, используемых различными компонентами ОС.

Библиотеки этого уровня по своему функциональному назначению можно разделить на две группы:

- системная библиотека C;
- функциональные библиотеки C/C++.

Системная библиотека базируется на Berkeley Software Distribution (BSD). Компания Google разработала собственную версию системной библиотеки *libc* — Bionic специально для мобильных устройств на основе Linux. Это было необходимо для обеспечения быстрой загрузки библиотеки в каждый процесс, и следовательно, библиотека должна была иметь маленький размер. Библиотека Bionic имеет размер около 200 Кбайт, что в два раза меньше размера стандартной библиотеки Linux *glibc*. Кроме того, необходимо было учитывать ограниченную мощность центрального процессора мобильного устройства. Это означает, что библиотека должна быть оптимизирована для максимального быстродействия. Конечно, сейчас это уже не актуально, современные мобильные устройства практически сравнялись по мощности процессора с нетбуками, но еще несколько лет назад это являлось серьезной проблемой.

Библиотека Bionic имеет встроенную поддержку важных для Android системных служб и регистрацию системных событий, но в то же время она не поддерживает некоторую функциональность, например исключения C++, и несовместима с GNU *libc* и стандартом POSIX.

Функциональные библиотеки представляют собой набор библиотек C/C++ типа OpenGL, WebKit, FreeType, SSL, базы данных SQLite и библиотек мультимедиа (Media Framework). Для разработчиков доступ к функциям этих библиотек реализован через использование Application Framework — каркаса приложений.

Dalvik Virtual Machine

Среда выполнения обеспечивает библиотеки ядра Dalvik Virtual Machine (виртуальная машина Dalvik), которые предоставляют требуемую функциональность для Java-приложений.

Прикладное программное обеспечение, запускаемое на мобильном устройстве, исполняет виртуальная машина Dalvik, которая хоть и является аналогом виртуальной машины Java, существенно от нее отличается. Dalvik относится к классу регистровых машин (регистры процессора используются как первичные модули хранения данных), идеально подходящих для работы на процессорах RISC-архитектуры, к которым относятся и процессоры ARM, применяемые в мобильных устройствах, тогда как стандартная виртуальная машина Java компании Sun Microsystems — стековая. В результате использования регистровой виртуальной машины Google надеется на 30 процентов уменьшить количество команд по сравнению со стековыми машинами.

Созданные с помощью стандартного Java-компилятора class-файлы преобразуются в байт-код Dalvik (*.dex) транслятором dx, входящим в состав SDK. Изнутри работающий Android выглядит как набор виртуальных машин Dalvik, в каждой из которых исполняется прикладная задача.

Виртуальная машина Dalvik, на которой построена вся операционная система Google Android, дает разработчикам удобный механизм для написания приложений, которым не принципиален объем используемой памяти и мощность процессора.

Уровень каркаса приложений

Уровень каркаса приложений находится на вершине системных библиотек, функциональных библиотек и Dalvik VM. На этом уровне находятся основные службы Android для управления жизненным циклом приложений, пакетами, ресурсами и т. д.

Программист имеет полный доступ к тем же API, которые используются основными приложениями. Архитектура этих приложений разработана с целью упрощения многократного использования компонентов. Любое разрабатываемое приложение может использовать возможности базовых приложений и, соответственно, любое другое стороннее приложение может использовать возможности вашего приложения (с учетом установленных разрешений). Этот же самый механизм позволяет многократно использовать уже разработанные компоненты.

Уровень приложений

Мобильное устройство Android поставляется с набором основных приложений, включая почтового клиента, программу для работы с SMS, календарь, навигационные карты, браузер, контакты и др.

Что интересно, платформа Android не делает разницы между основными приложениями, входящими в комплект мобильного телефона, и сторонним программным обеспечением — таким образом, ключевые приложения, входящие в стандартный набор программного обеспечения, можно заменить при желании альтернативными приложениями.

При разработке приложений программисты имеют полный доступ ко всей функциональности операционной системы. Архитектура приложений построена так, чтобы было легко использовать основные компоненты, предоставляемые системой. Также есть возможность создавать свои компоненты и предоставлять их в открытое использование.

Как программировать под Android

Большинство из вас, возможно, думают, что программы для Android можно писать на языке Java. Это не совсем так — писать можно еще и на C или C++. Все зависит от того, для какого уровня вы пишете программное обеспечение. В данной книге рассматривается программирование для верхнего уровня — уровня приложений. Программирование на этом уровне осуществляется на языке Java, а при разработке приложений вы пользуетесь функциональностью, предоставляемой библиотеками, находящимися на уровне каркаса приложений, которые инкапсулируют нижние слои архитектуры.

Если вы программируете на C/C++, можно также заниматься разработкой библиотек уровня каркаса приложений и системных библиотек, добавляя функциональность, которой пока нет в стандартной системе Android. Поскольку Android является системой с открытым исходным кодом, вы имеете полный доступ к любому уровню системной архитектуры и полную свободу в усовершенствовании этой системы.

Кроме разработки пользовательских приложений, вы можете разрабатывать драйверы на уровне ядра или портировать эту систему на другие аппаратные платформы, если

она еще не была на них портирована, но это уже тема отдельной книги, здесь мы будем работать на Java, создавая программы для Application Level (уровня приложений).

Компоненты Android-приложения

Приложения для Android состоят из компонентов, которые система может запускать и управлять ими так, как ей необходимо.

Всего в Android-приложениях существует четыре типа компонентов:

- ☐ Activity;
- ☐ Service;
- ☐ Broadcast Receiver;
- ☐ Content Provider.

Взаимодействие этих компонентов осуществляется с помощью объектов Intent. Сейчас мы кратко рассмотрим компоненты и их взаимодействие.

Activity

Компонент Activity представляет собой визуальный пользовательский интерфейс для приложения — окно. Как правило, окно полностью заполняет экран мобильного устройства, но может иметь размеры меньше, чем у экрана. Activity может также использовать дополнительные окна, например всплывающее диалоговое окно, которое запрашивает пользовательский ответ для основного Activity, или окно уведомления о каком-либо событии в приложении или системе.

Все Activity реализуются как подкласс базового класса Activity. Приложение может содержать несколько Activity. Каждый Activity независим от других. При открытии нового Activity работа предыдущего Activity приостанавливается, а сам он вносится и сохраняется в стек Activity (стек и взаимодействие Activity будут рассмотрены в главе 12).

Service

Компонент Service не имеет визуального интерфейса пользователя и выполняется в фоновом режиме в течение неопределенного периода времени, пока не завершит свою работу. Этот компонент аналогичен службам в настольных операционных системах.

Приложения могут подключаться к компоненту Service или запускать его, если он не запущен, а также останавливать уже запущенные компоненты. Подключившись к Service, вы можете обращаться к функциям компонента через предоставляемый этим компонентом интерфейс.

Broadcast Receiver

Компонент Broadcast Receiver — компонент для получения внешних событий, происходящих в системе, и формирования реакции на эти события. Инициализировать передачи могут другие приложения или сама система Android. Приложение может иметь

несколько компонентов Broadcast Receiver, чтобы отлавливать происходящие события, которые оно считает важными для своей работы.

Компоненты Broadcast Receiver не имеют пользовательского интерфейса. Однако они могут запустить другой компонент — Activity или службу, обработать поступившую информацию или показать уведомление на экране мобильного устройства, чтобы предупредить пользователя о наступившем событии.

Content Provider

Компонент Content Provider (контент-провайдер) делает определенный набор данных, используемых приложением, доступным для других приложений. Этот компонент является своеобразным посредником между хранилищем данных и клиентским приложением.

Данные в Android могут быть сохранены различными способами: в файловой системе, в базе данных SQLite или любым другим способом. Content Provider для безопасного доступа к данным используют механизм разрешений. Это означает, что вы можете сконфигурировать собственный Content Provider, чтобы разрешить доступ к своим данным из других приложений, а также использовать Content Provider другого приложения для обращения к его хранилищу данных.

Объекты Intent

Главная особенность платформы Android состоит в том, что одно приложение может использовать элементы других приложений при условии, что эти приложения разрешают использовать свои компоненты. При этом ваше приложение не включает код другого приложения или ссылки на него, а просто запускает нужный элемент другого приложения.

Поэтому, в отличие от приложений в большинстве других систем, у приложений Android нет единственной точки входа для запуска всего приложения, аналогичной, например, функции `main()` в С-подобных языках программирования. Программа может иметь несколько точек входа.

Для реализации такого использования компонентов других приложений система должна быть в состоянии запустить процесс для приложения, в котором находится требуемый компонент, и инициализировать нужные ей объекты. Для запуска компонентов используются объекты Intent (намерения), которые определяют имя запускаемого компонента и, при необходимости, могут передавать набор параметров, определяющих условия запуска этого компонента.

В системе Android все коммуникации между приложениями, службами и отдельными компонентами происходят только с использованием объектов Intent.

Резюме

В этой главе мы кратко познакомились с архитектурой системы Android. Как вы увидели, система Android состоит из четырех уровней, таких как: ядро, системные библиотеки, каркас приложений и сам уровень приложений, для которого мы и будем разрабатывать программы в этой книге.

Также мы кратко рассмотрели фундаментальные компоненты приложений Android (Activity, Service, Broadcast Receiver и Content Provider) и их взаимодействие в системе. Пока вы получили только начальные сведения о назначении компонентов, с которыми будете работать на протяжении всей книги.

В следующей главе мы займемся инсталляцией и настройкой инструментов, необходимых для разработки приложений под Android, а также рассмотрим работу с эмулятором устройства Android.



ГЛАВА 2

Установка и настройка среды разработки

Чтобы писать приложения для Android, необходимо установить среду разработки. В этой главе мы установим Java Development Kit, интегрированную среду разработки Eclipse, Android SDK и Android Development Tools, а также сконфигурируем Eclipse для разработки приложений под Android.

Создание среды разработки

Все инструменты, которые нам потребуются для разработки приложений для платформы Android, доступны и абсолютно бесплатны. Google предлагает для свободного скачивания набор библиотек и инструментов для разработки приложений — Android SDK (Software Development Kit), который предназначен для x86-машин, работающих под операционными системами Windows XP, Mac OS и Linux.

Перед началом работы по созданию Android-приложений нам необходимо загрузить и установить следующее программное обеспечение:

- ☐ Java Development Kit (JDK);
- ☐ Eclipse IDE;
- ☐ Android SDK;
- ☐ Android Development Tools (ADT) — плагин для Eclipse.

ПРИМЕЧАНИЕ

В принципе, вместо Eclipse можно использовать и другую среду разработки, например NetBeans IDE, но возможны некоторые неудобства, о которых будет рассказано далее.

Поскольку среда разработки не зависит от операционной системы и Android-приложения в настольных операционных системах запускаются в эмуляторе мобильного устройства, необходимые инструменты для разработки можно установить на любую из систем: Windows, Linux или Mac OS.

Android SDK включает эмулятор устройства Android для всех трех операционных систем, и, поскольку Android-приложения выполняются на виртуальной машине, нет никакого преимущества для разработки приложений в любой из этих ОС.

Установка JDK

Для разработки программ на языке Java нам потребуется специальное программное обеспечение. Самые новые версии системного программного обеспечения, необходимого для поддержки, можно загрузить с сайта компании Oracle.

Для запуска и исполнения программ необходима Java Runtime Environment (JRE) — среда выполнения Java. Для разработки программ также требуется комплект разработки программного обеспечения — Java Development Kit (JDK). Java Development Kit — это комплект разработчика приложений на языке Java, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и уже включающий в себя Java Runtime Environment. Java Development Kit доступен для свободной загрузки на сайте Oracle по адресу **<http://oracle.com>**. После загрузки JDK сделайте установку с параметрами по умолчанию, предлагаемыми мастером установки.

Однако в состав JDK не входит интегрированная среда разработки на Java (IDE), поэтому для разработки приложений необходимо установить Eclipse.

Установка Eclipse

Итак, наш следующий шаг — загрузка интегрированной среды разработки Eclipse. Eclipse доступна для загрузки по адресу **<http://www.eclipse.org/downloads/>**. После того как вы загрузили Eclipse, распакуйте архив и запустите файл eclipse.exe. Обычно Eclipse устанавливают в свой пользовательский каталог (в MS Windows), но вы можете установить ее в каталог Program Files или любой другой.

Несмотря на то что для разработки можно использовать и другие IDE, есть несколько причин, почему именно Eclipse рекомендуется для разработки Android-приложений. Eclipse — это наиболее полно документированная, свободная и доступная интегрированная среда разработки для Java. Eclipse также очень проста в изучении — освоение ее займет минимальное время. Это делает Eclipse очень удобной средой для разработки приложений под Android.

Кроме того, компания Google выпустила для Eclipse плагин ADT — Android Development Tools, который позволяет создавать Android-проекты, компилировать и использовать эмулятор мобильного Android-устройства для их запуска и отладки. Плагин Android Development Tools автоматически создает в Eclipse необходимую структуру Android-проекта и устанавливает требуемые параметры настройки компилятора.

Установка Android SDK

Чтобы разрабатывать приложения для Android, необходим Android SDK. SDK включает эмулятор, так что нет необходимости в мобильном устройстве с ОС Android, чтобы разрабатывать приложения для Android. Последняя версия на момент написания книги — Android SDK v3.2. Android SDK доступен для свободного скачивания на официальном сайте Android по адресу **<http://developer.android.com/sdk/index.html>**.

После загрузки распакуйте файл в выбранную вами директорию. В ранних версиях SDK архив содержал полный комплект компонентов текущей платформы Android. Начиная с версии 2.0 архив Android SDK содержит стартовый набор файлов и инструмен-

тальные средства. В версии 2.0 и выше используется специальный инструмент Android SDK and AVD Manager для установки и обновления компонентов Android SDK — библиотек, инструментов и документации.

Чтобы разрабатывать приложения, необходимо установить не менее одной версии платформы Android, используя Android SDK and AVD Manager. Это требует подключения к Интернету, т. к. все необходимые для загрузки и обновления компоненты SDK находятся в репозитории на сервере Google.

Чтобы открыть Android SDK and AVD Manager, запустите файл SDK Manager.exe в корневом каталоге SDK. После установки соединения с репозиторием Google в окне менеджера будет отображен список доступных пакетов, как показано на рис. 2.1.

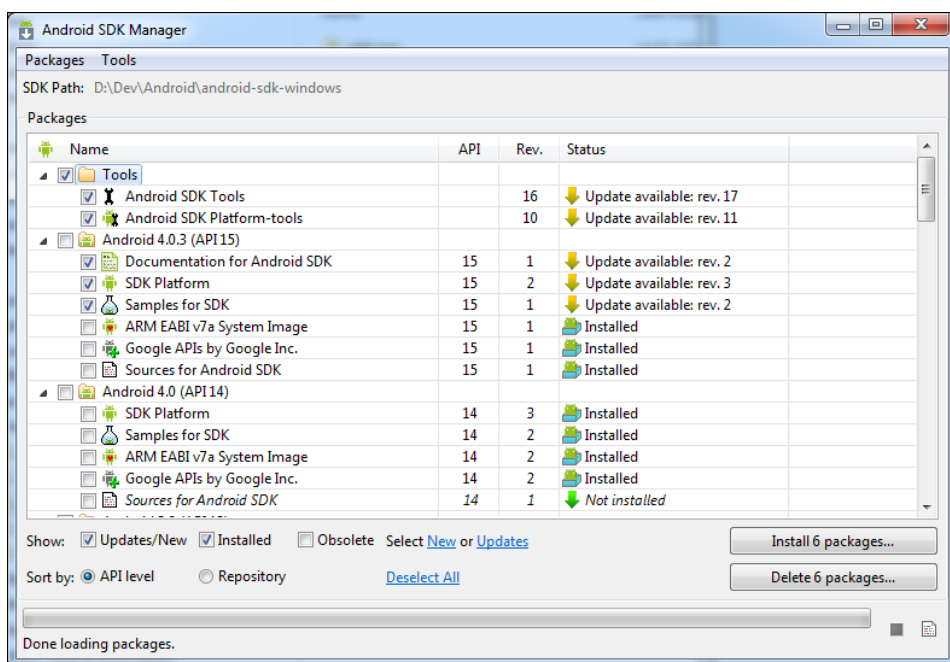


Рис. 2.1. Выбор пакетов для инсталляции

Выбрав необходимые пакеты, нажмите кнопку **Install** и далее, следуя инструкциям, установите компоненты SDK. Кроме библиотек из репозитория Google, можно также дополнительно устанавливать библиотеки сторонних разработчиков.

После успешной установки Android SDK можно приступать к установке ADT-плагина для Eclipse.

Установка Android Development Tools

Далее нам надо установить плагин Android Development Tools (ADT). Для его установки сначала запустите Eclipse, затем выберите в главном меню пункт **Help | Install New Software**. В появившемся диалоговом окне нажмите кнопку **Add**. После установки соединения пометьте устанавливаемые компоненты ADT, как показано на рис. 2.2.

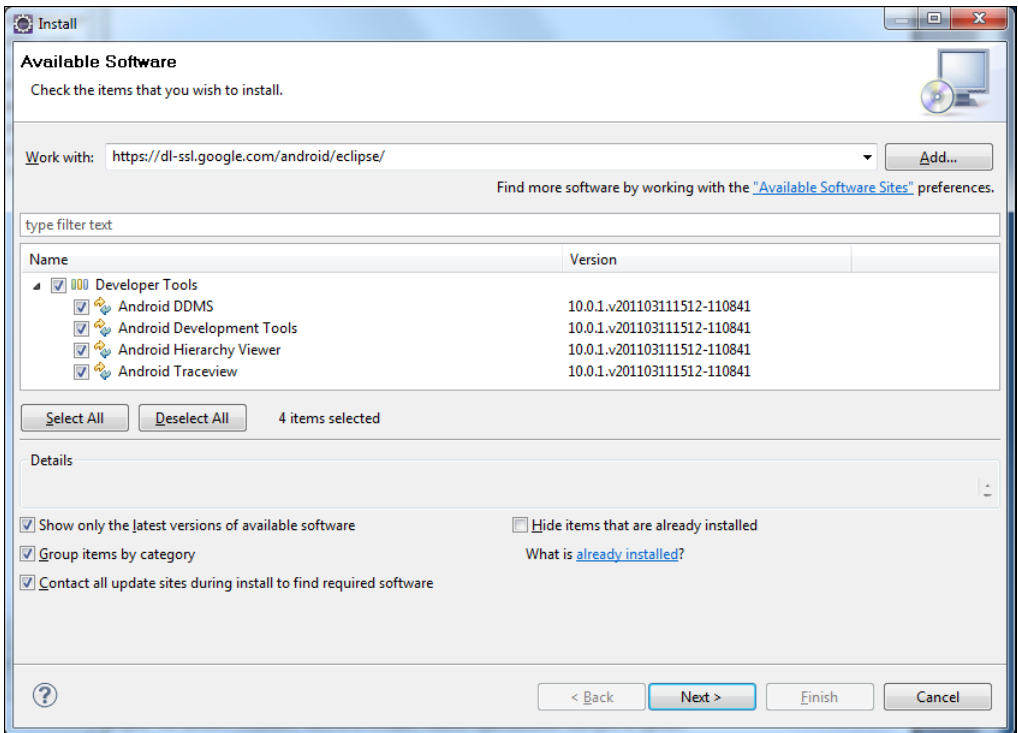


Рис. 2.2. Установка компонентов ADT

После выполнения всех инструкций по установке перезапустите среду Eclipse.

Теперь нам необходимо связать среду разработки Eclipse с каталогом Android SDK. Для этого выберите в главном меню Eclipse пункт **Window | Preferences**, чтобы открыть диалоговое окно **Preferences**. Выберите на левой панели пункт **Android**. В поле **SDK Location** на основной панели необходимо указать каталог, в котором расположен Android SDK. Для этого нажмите кнопку **Browse** и установите путь к каталогу Android SDK, как показано на рис. 2.3.

Нажмите кнопку **Apply**, затем **OK**. Среда Eclipse теперь "видит" библиотеки и инструменты Android SDK, и можно начинать разрабатывать приложения для Android.

Плагин ADT для Eclipse автоматизирует процесс построения приложений для Android, интегрируя инструменты разработки непосредственно в среду разработки Eclipse, что делает создание, запуск и отладку ваших приложений быстрее и проще. Плагин ADT также добавляет в среду Eclipse несколько дополнительных компонентов, облегчающих создание приложений для Android:

- ☐ мастер создания проекта — New Project Wizard, который упрощает создание новых Android-проектов и формирует шаблон проекта;
- ☐ редактор Layout Editor — для разработки графического интерфейса приложения;
- ☐ различные редакторы ресурсов для создания, редактирования и проверки правильности XML-ресурсов разработчика.

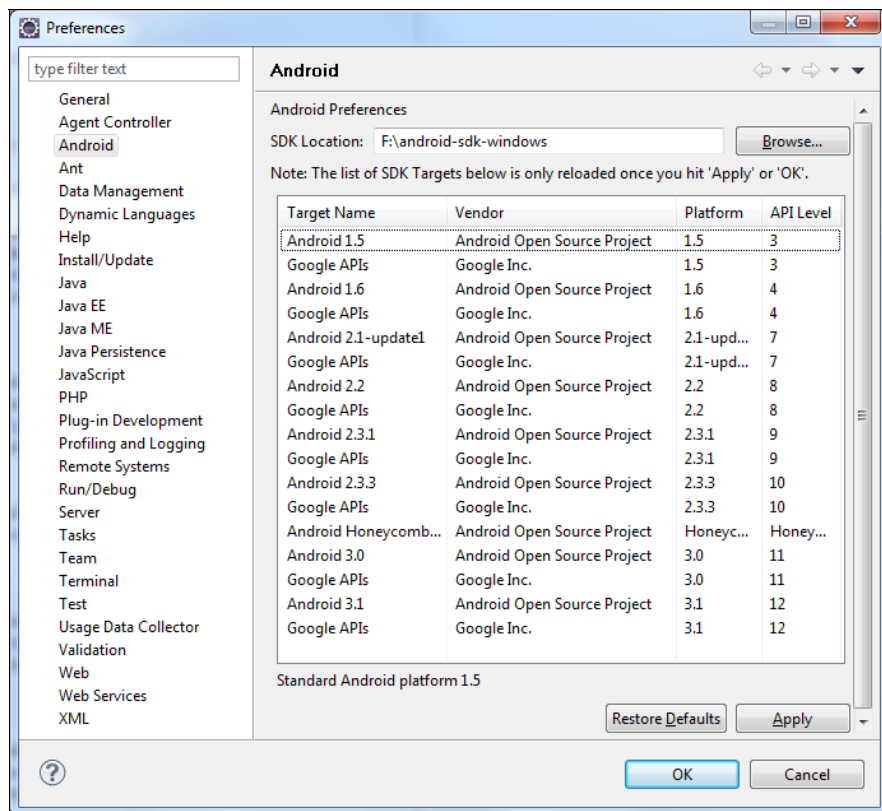


Рис. 2.3. Связывание среды Eclipse с Android SDK

Плагин ADT для Eclipse также предоставляет доступ к инструментам из комплекта Android SDK напрямую из среды Eclipse. Например, ADT позволяет запускать эмулятор мобильного устройства, получить доступ к Dalvik Debug Monitor Service (DDMS) — инструмента SDK для управления и отладки приложений на мобильном устройстве, настройки контрольных точек (breakpoints), просмотра информации о потоках и процессах непосредственно из среды Eclipse. Подробнее инструменты для разработки будут рассмотрены позднее.

Версии SDK и Android API Level

Перед началом разработки приложений для Android полезно понять общий подход платформы к управлению изменением API. Также важно понять Android API Level (идентификатор уровня API) и его роль в обеспечении совместимости вашего приложения с устройствами, на которых оно будет устанавливаться.

Уровень API — целочисленное значение, которое однозначно определяет версию API платформы Android. Платформа обеспечивает структуры API, которые приложения могут использовать для взаимодействия с системой Android. Каждая следующая версия платформы Android может включать обновления API.

Обновления API-структуры разработаны так, чтобы новый API оставался совместимым с более ранними версиями API. Таким образом, большинство изменений в API является

совокупным и вводит новые функциональные возможности или исправляет предыдущие. Поскольку часть API постоянно обновляется, устаревшие API не рекомендуются к использованию, но не удаляются из соображений совместимости с имеющимися приложениями.

Уровень API, который использует Android-приложение, определяется целочисленным идентификатором, который указывается в файле конфигурации каждого Android-приложения. На момент написания этой книги уже существует 15 уровней API.

Таблица 2.1 определяет соответствие уровня API и версии платформы Android.

Таблица 2.1. Соответствие версии платформы и уровня API

Версия платформы	Уровень API
Android 4.0.3	15
Android 4.0, 4.0.1, 4.0.2	14
Android 3.2	13
Android 3.1	12
Android 3.0	11
Android 2.3.3, 2.3.4	10
Android 2.3	9
Android 2.2	8
Android 2.1	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

Обзор Android SDK

Android SDK включает в себя разнообразные библиотеки, документацию и инструменты, которые помогают разрабатывать мобильные приложения для платформы Android (рис. 2.4).

- docs/ — документация SDK с обширной справочной информацией, детализирующей, что включено в каждый пакет и класс и как это использовать при разработке приложений.
- platforms/ — содержит набор версий платформ Android (которые вы скачали, ведь вам обязательно скачивать все старые версии).
- tools/ — содержит инструменты для отладки и тестирования, а также еще некоторый набор дополнительных программ для разработки приложений Android.

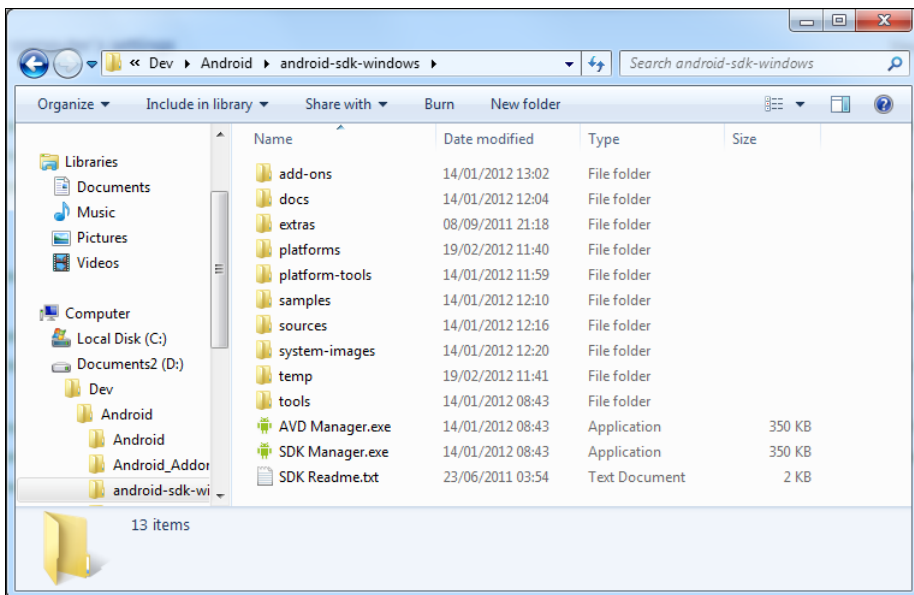


Рис. 2.4. Содержимое каталога Android SDK после закичивания обновлений

- ☐ `platform-tools/` — несколько платформенно-зависимых инструментальных средств для разработки, которые позволяют компилировать и отлаживать создаваемые приложения, но этот набор инструментов обновляется с выходом новой версии Android SDK.
- ☐ `add-ons/` — каталог, который содержит аддоны (дополнения) для среды разработки Android.
- ☐ `samples/` — примеры типовых приложений, демонстрирующие некоторые из возможностей Android, и простые программы, которые показывают, как использовать индивидуальные особенности API в вашем коде.
- ☐ `system-images/` — содержит системные образы Android, отдельные для каждой закачанной вами версии.

Инструменты для разработки и отладки приложений

Android SDK также включает множество других инструментальных средств для отладки и установки создаваемых приложений. Если вы разрабатываете приложения для Android с помощью IDE Eclipse, многие инструменты командной строки, входящие в состав SDK, уже используются при сборке и компиляции проекта. Однако кроме них SDK содержит еще ряд полезных инструментов для разработки и отладки приложений, перечисленных ниже.

- ☐ *android* — важный инструмент разработки, запускаемый из командной строки, который позволяет создавать, удалять и конфигурировать виртуальные устройства, создавать и обновлять Android-проекты (при работе вне среды Eclipse) и обновлять Android SDK новыми платформами, дополнениями и документацией.
- ☐ *Dalvik Debug Monitor Service (DDMS)* — этот инструмент интегрирован с Dalvik Virtual Machine, стандартной виртуальной машиной платформы Android, он позво-

ляет управлять процессами на эмуляторе или устройстве, а также помогает в отладке приложений. Вы можете использовать этот сервис для завершения процессов, выбора определенного процесса для отладки, генерирования трассировочных данных, мониторинга использования памяти мобильного устройства, информации о потоках, делать скриншоты эмулятора или устройства и многое другое.

- ❑ *Hierarchy Viewer* — визуальный инструмент, который позволяет отлаживать и оптимизировать пользовательский интерфейс разрабатываемого приложения. Он показывает визуальное дерево иерархии классов `View`, анализирует быстродействие перерисовки графических изображений на экране и может выполнять еще много других функций для анализа графического интерфейса приложений.
- ❑ *Layoutopt* — инструмент командной строки, который помогает оптимизировать схемы разметки и иерархии разметок в создаваемом приложении. Необходим для решения проблем при создании сложных графических интерфейсов, которые могут затрагивать производительность приложения.
- ❑ *Draw 9-patch* — графический редактор, который позволяет легко создавать Nine-Patch графику для интерфейса разрабатываемых приложений.

ПРИМЕЧАНИЕ

Nine-Patch (еще ее называют 9-patch) графика — это файл, аналогичный файлам формата PNG, но имеющий специальную служебную рамку толщиной в 1 пиксел. Эта рамка остается неизменной при изменении размеров, масштабировании файла. Данный формат используется в Android при создании скинов для элементов управления — кнопок, текстовых полей и т. д. Файлы Nine-patch имеют расширение *.9.png.

- ❑ *sqlite3* — инструмент командной строки для доступа к файлам данных SQLite, созданных и используемых Android-приложениями.
- ❑ *Traceview* — этот инструмент выдает графический анализ трассировочных логов, которые можно генерировать из приложений.
- ❑ *mksdcard* — инструмент для создания образа диска, который вы можете использовать в эмуляторе для симуляции наличия внешней карты памяти (например, SD-карты).

Далее в книге, при разработке учебных приложений, мы рассмотрим некоторые из перечисленных инструментов. Наиболее важный из них — эмулятор мобильного устройства, однако в состав SDK входят и другие инструменты для отладки, упаковки и инсталляции приложений на эмулятор.

Создание переменных окружения

Для использования инструментов, предоставляемых Android SDK, необходимо настроить переменные окружения (для OS Windows).

Откройте окно **System Properties**, перейдите на вкладку **Advanced** и щелкните по кнопке **Environment Variables**. В группе **System variables** выберите переменную **Path** и щелкните по кнопке **Edit**. Откроется окно **Edit System Variable**. Добавьте в конец списка переменных полные пути до двух подкаталогов из вашего каталога Android SDK (например: `F:\android-sdk-windows\tools` и `F:\android-sdk-windows\platform-tools`), как показано на рис. 2.5.

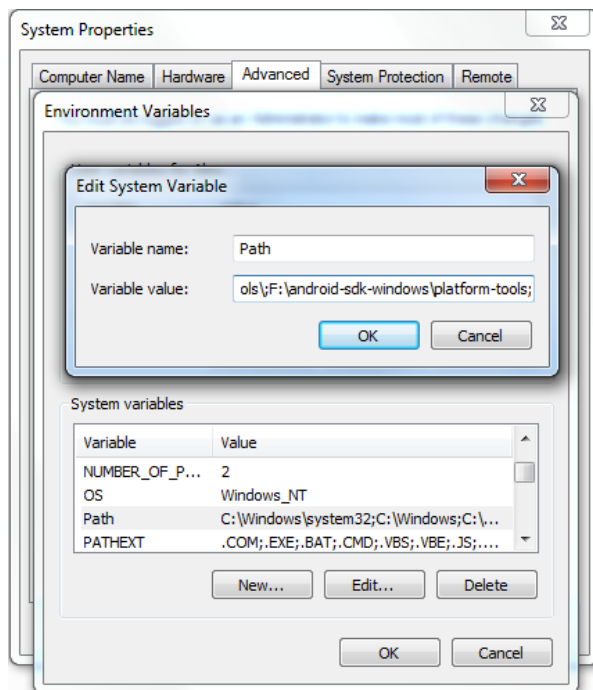


Рис. 2.5. Добавление переменных окружения

Android Virtual Device

Android Virtual Device (Виртуальное устройство Android) — это эмулятор, который запускается на обычном компьютере. Эмулятор используется для проектирования, отладки и тестирования приложений в реальной среде выполнения.

Прежде чем мы начнем писать программы под Android в Eclipse, необходимо создать экземпляр Android Virtual Device (AVD), в котором будут запускаться и проверяться наши программы. AVD определяет системное изображение и параметры настройки устройства, используемые эмулятором.

Создать экземпляр AVD можно двумя способами:

1. В командной строке утилитой `android`, доступной в каталоге, куда вы установили Android SDK, в папке `tools`.
2. С помощью визуального инструмента **Android Virtual Device Manager**. Его можно запустить из корневого каталога Android SDK или в среде Eclipse, выбрав пункт меню **Window | AVD Manager**. При этом появится окно **Android Virtual Device Manager**, с помощью которого можно создавать и конфигурировать эмуляторы мобильного устройства, а также загружать обновления Android SDK (рис. 2.6).

Конфигурирование AVD

В правой части панели **List of existing Android Virtual Devices** нажмите кнопку **New**, при этом откроется окно **Create new Android Virtual Device (AVD)** (рис. 2.7).

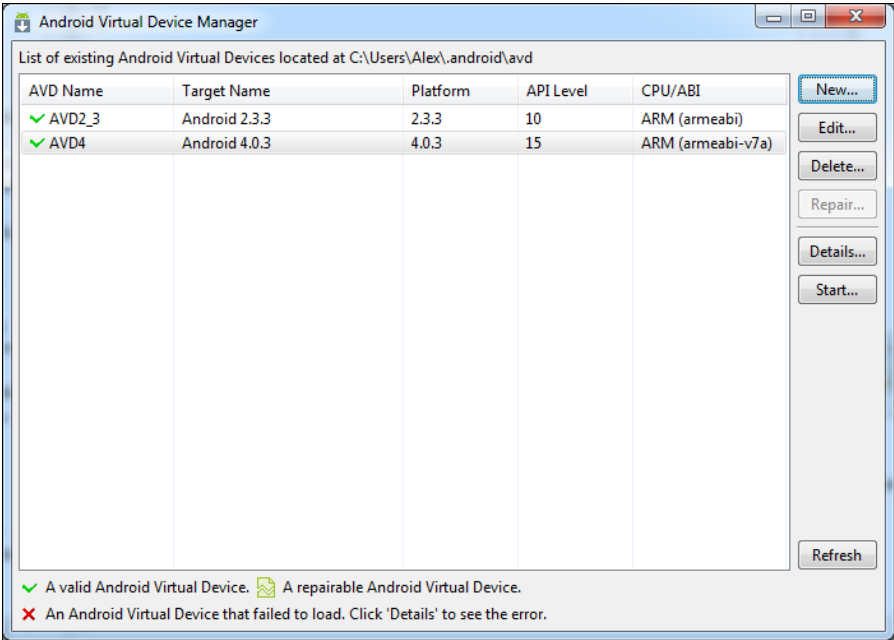


Рис. 2.6. Окно Android Virtual Device Manager

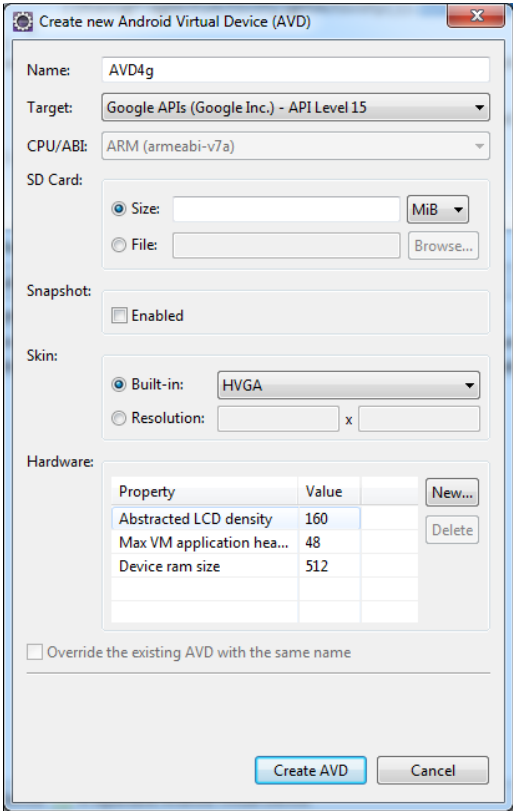


Рис. 2.7. Диалоговое окно создания нового AVD

В этом окне задайте нужную конфигурацию для создаваемого эмулятора устройства:

- ☐ **Name** — имя создаваемого устройства;
- ☐ **Target** — версия Android SDK, поддерживаемая эмулятором Android-устройства. Устройство имеет обратную совместимость со старыми версиями SDK, т. е. если выбрана версия Android 2.3.3, эмулятор будет поддерживать версии SDK 2.3, 2.1, 1.6, 1.5 и т. д.;
- ☐ **SD Card** — устанавливает виртуальную карту памяти, для которой можно в текстовом поле **Size** задать требуемую емкость;
- ☐ **Skin** — тип экрана устройства. Загружаемая платформа включает ряд скинов для эмулятора, которые можно использовать для моделирования работы приложения в устройствах с разными размерами и разрешением экрана. Набор скинов для эмулятора в зависимости от установленной версии SDK, указанной в поле **Target**, содержит различные типы и размеры экрана, например:
 - **HVGA** (Half-size VGA Video Graphics Array), размер 320×480, средняя плотность, нормальный экран;
 - **WVGA800** (Wide Video Graphics Array), размер 480×800, высокая плотность, нормальный экран;
 - **WVGA854** (Wide Video Graphics Array), 480×854, высокая плотность, нормальный экран;
 - **QVGA** (Quarter Video Graphics Array), размер 240×320, низкая плотность, малый экран;
 - **WQVGA400** (Wide Quarter Video Graphics Array), размер 240×400, низкая плотность, нормальный экран;
 - **WQVGA432** — то же, что и предыдущий, но с размером 240×432.

В дополнение к использованию встроенного разрешения экрана можно также указать свое собственное разрешение, если в этом есть необходимость;

- ☐ **Hardware** — имитация оборудования, установленного на устройстве. При необходимости нажатием кнопки **New** можно вызвать окно для добавления дополнительного виртуального оборудования (рис. 2.8).

После задания конфигурации и нажатия кнопки **Create AVD** (см. рис. 2.8) менеджер создаст новое виртуальное устройство, название и версия API которого появятся в списке **List of existing Android Virtual Devices** (см. рис. 2.6).

ПРИМЕЧАНИЕ

Для более тонкой настройки лучше использовать инструмент командной строки `android.exe`. Он имеет более широкие возможности, чем визуальный AVD Manager, и удобен для конфигурации сети, портов и виртуального оборудования эмулятора. К сожалению, из-за ограниченного объема книги нет возможности рассмотреть подробнее этот инструмент.

В зависимости от поддерживаемой версии SDK внешний вид виртуального устройства будет отличаться. Для версий 4.0.3 и разрешения экрана HVGA устройство примет вид, показанный на рис. 2.9, для более старых версий SDK внешний вид устройства будет другим.

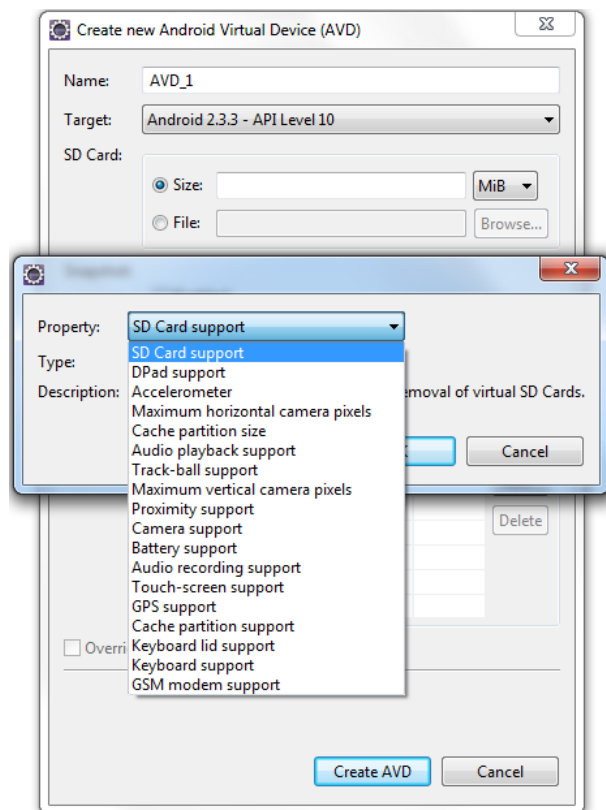


Рис. 2.8. Окно для добавления дополнительного виртуального оборудования



Рис. 2.9. Внешний вид AVD версии 4.0.3 с разрешением HVGA

Окно эмулятора оформлено в виде телефона с дополнительной клавиатурой. На рис. 2.9 показана загруженная операционная система. Эмулятор имитирует сенсорный экран реального мобильного устройства — в эмуляторе на экран нажимают левой кнопкой мыши.

В эмуляторе три виртуальных рабочих стола, перемещение по которым осуществляется с помощью кнопок со стрелками на навигационной панели устройства или передвижением курсора при нажатой левой кнопке мыши (в реальном устройстве — перемещая палец по экрану). Кроме ярлыков программы на рабочем столе можно размещать виджеты.

Если вам требуется создавать и тестировать приложения, специфичные для планшетных ПК, вы можете создать еще один вариант эмулятора, например, для версии 3.2 и разрешения WXGA (рис. 2.10)

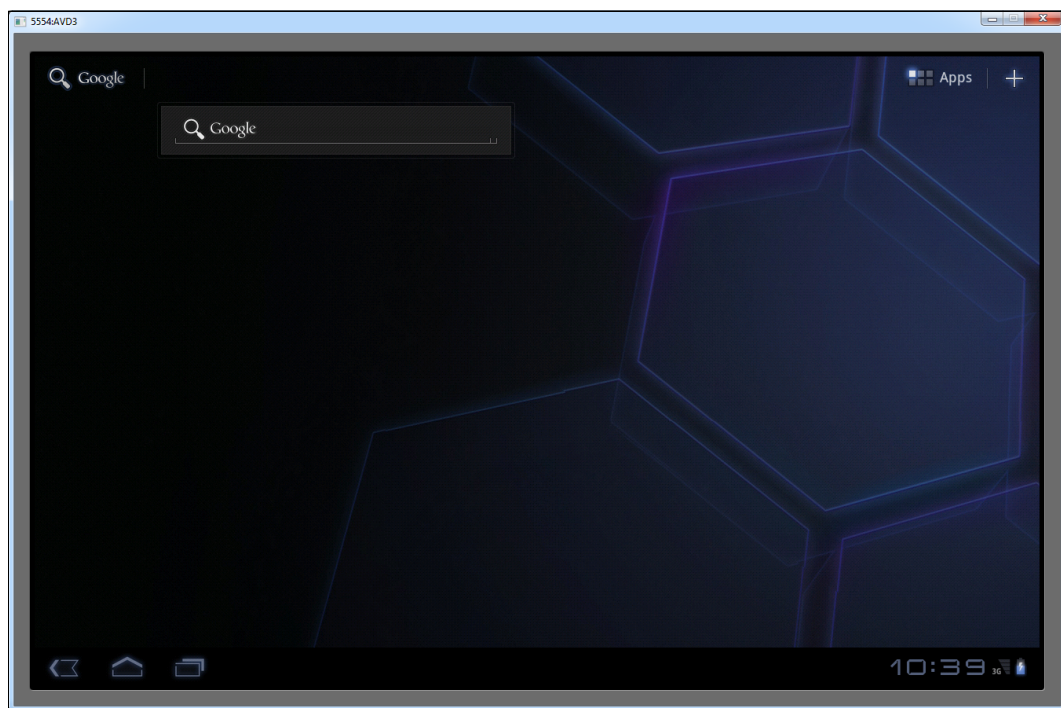


Рис. 2.10. Вариант эмулятора для планшетного ПК с разрешением экрана WXGA

Однако при использовании эмуляторов с большим разрешением экрана необходимо учитывать, что такой эмулятор на вашем рабочем компьютере будет работать гораздо медленнее, чем эмулятор с маленьким экраном. Это связано с тем, что эмулятор требует больших ресурсов для эмуляции архитектуры и работы процессоров ARM и для поддержки графики эмулируемого устройства. Поэтому, если нет необходимости при отладке и тестировании программ в большом разрешении экрана, используйте меньшее разрешение, например HVGA — наиболее популярное разрешение для мобильных устройств Android.

Если нет необходимости использовать самую последнюю версию Android SDK в приложении, можно запускать его на более старой версии эмулятора, например 2.1 или 2.3.3, которая также будет работать быстрее, чем эмулятор версии 4.0.3.

Вы можете создать несколько эмуляторов.

Сочетания клавиш

Эмулятор Android поддерживает несколько сочетаний клавиш, которые позволяют управлять различными режимами работы устройства. Вот некоторые из сочетаний, которые можно использовать с эмулятором:

- ☐ <Esc> — навигация назад;
- ☐ <Home> — переход на Home Screen (домашний экран);
- ☐ <F2> — вызов контекстного меню (Toggles context-sensitive menu);
- ☐ <F3> — вызов Call Log (журнала вызовов);
- ☐ <F4> — "поднять" или "положить" трубку;
- ☐ <F7> — кнопка включения/выключения питания устройства;
- ☐ <F5> — поиск;
- ☐ <F6> — режим трекбола (trackball mode);
- ☐ <Ctrl>+<F5> — увеличение громкости звонка;
- ☐ <Ctrl>+<F6> — уменьшение громкости звонка;
- ☐ <Ctrl>+<F11>/<Ctrl>+<F12> — изменение ориентации экрана (горизонтальная или вертикальная).

Например, для тестирования внешнего вида создаваемого приложения при разных положениях экрана комбинацией клавиш <Ctrl>+<F11> можно изменять расположение экрана с вертикального на горизонтальный и наоборот.

Неподдерживаемая функциональность

Эмулятор, тем не менее, не поддерживает некоторые функции, доступные на реальных устройствах, такие как:

- ☐ входящие и исходящие сообщения. Однако можно моделировать обращения по телефону через интерфейс эмулятора с помощью инструмента DDMS;
- ☐ соединение через USB;
- ☐ видеокамера (однако есть имитатор работы видеокамеры, хотя работу с видеокамерой лучше тестировать на реальном Android-устройстве);
- ☐ подключение наушников;
- ☐ определение статуса соединения;
- ☐ определение уровня заряда аккумуляторной батареи;
- ☐ определение вставки или изъятия карты памяти;
- ☐ соединение по Bluetooth.

Конечно, реальные Android-устройства несколько отличаются от эмулятора, но в целом AVD разработан очень качественно и близок по функциональности к реальному устройству. Для работы большинства (но не всех) примеров приложений, приведенных в этой книге, эмулятора будет достаточно.

Резюме

В этой главе мы рассмотрели создание среды разработки для Android-приложений. Также мы создали экземпляр эмулятора мобильного Android-устройства, на котором будем запускать и тестировать разрабатываемые нами приложения.

В следующей главе мы приступим к созданию Android-приложений. Мы спроектируем простейшее приложение и детально изучим его структуру, принципы работы и развертывание приложения на эмулятор и мобильное Android-устройство.



ГЛАВА 3

Первое приложение Android

Итак, после того как мы установили и настроили среду разработки и эмулятор мобильного устройства, можно приступить к созданию своего первого приложения. В этой главе мы вначале сформируем простейшее Android-приложение и запустим его на эмуляторе. Затем мы познакомимся со структурой проекта, его каталогами и файлами, которые среда разработки генерирует при создании Android-проекта.

Создание проекта в Eclipse

Создавать проект для платформы Android довольно легко, особенно при использовании в качестве среды разработки Eclipse. Для создания проекта Android откройте Eclipse и выберите **File | New | Android**. Затем выберите опцию **Android Project** и нажмите кнопку **Next** (рис. 3.1).

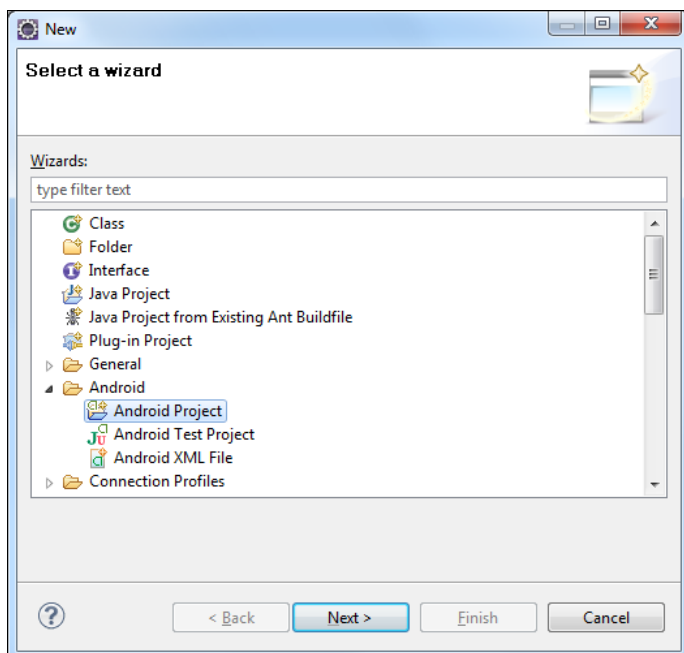


Рис. 3.1. Диалоговое окно создания нового проекта

Откроется страница **Create Android Project** мастера создания нового проекта **New Android Project** (рис. 3.2). В этом окне заполните поле **Project Name**: `FirstAndroidApp` — это будет название нашего проекта и имя каталога, который будет содержать проектные файлы.

Флажок **Use default location**, включенный по умолчанию, позволяет изменять местоположение на диске, где файлы проекта будут сгенерированы и сохранены.

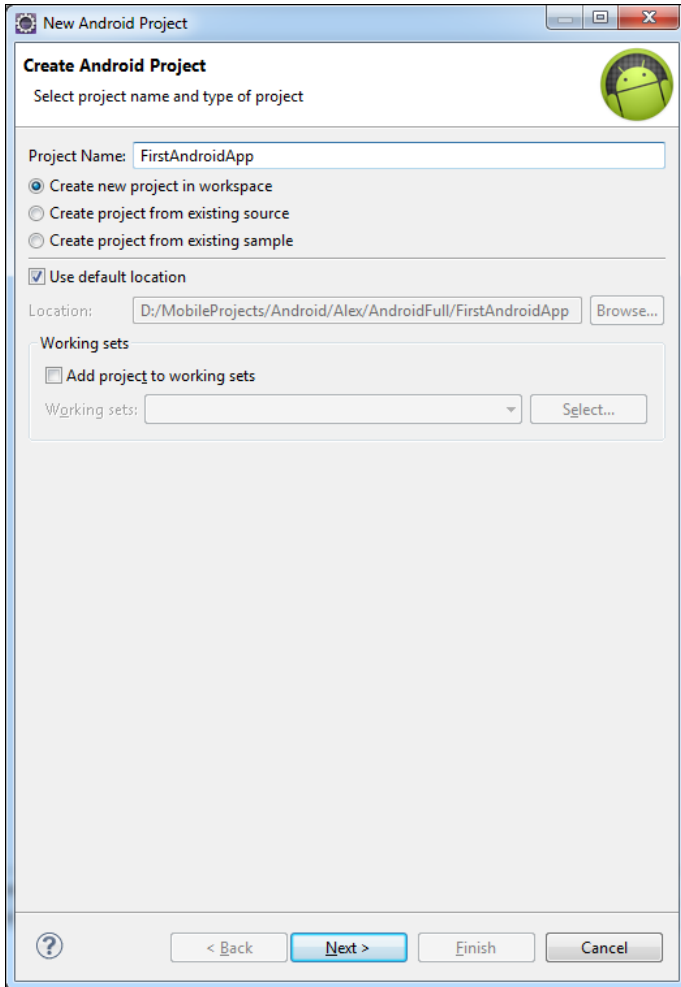


Рис. 3.2. Диалоговое окно **New Android Project**

Далее мы переходим на страницу **Select Build Target**. На этой странице мы должны определить целевую версию платформы Android для нашего приложения. Если вы установили все пакеты с репозитория Google (не включая пакеты от сторонних разработчиков), то у вас эта страница будет выглядеть так, как на рис. 3.3.

- ❑ Страница **Select Build Target** дает указание компилятору собрать приложение для выбранного уровня API. В поле **Build Target** выберем платформу Android 4.0.3. Это означает, что ваше приложение будет скомпилировано с использованием библиотек

Android 4.0.3. Android-приложения совместимы снизу вверх, так что приложения, собранные, например, на версии библиотек API 1.5, будут работать на платформах 1.6, 2.1, 2.3.3 и вплоть до 4.0.3.

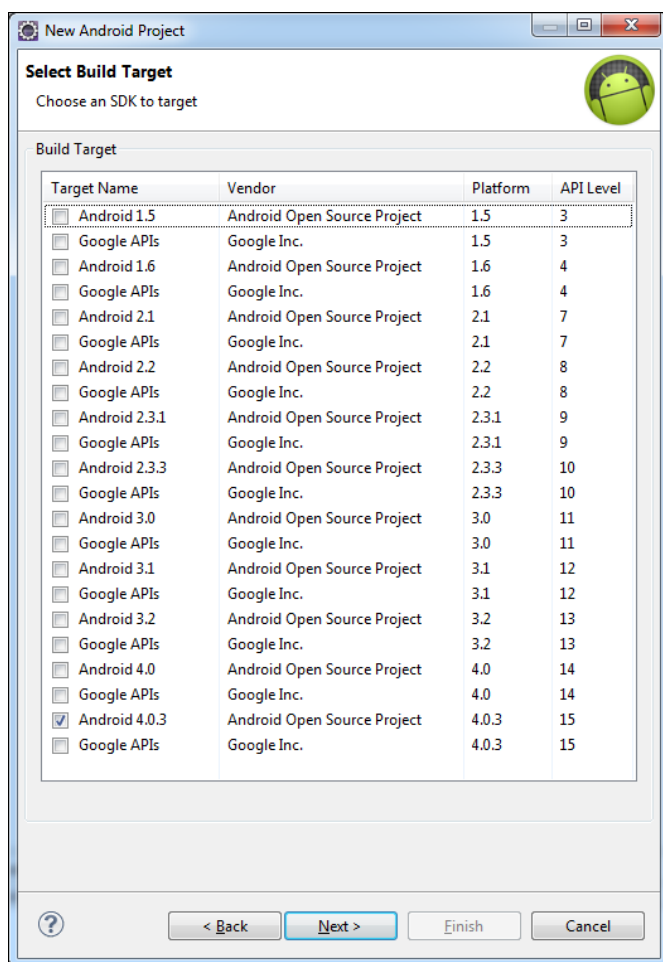


Рис. 3.3. Окно выбора целевой версии

ОБРАТИТЕ ВНИМАНИЕ

При выборе уровня API необязательно устанавливать самый последний из доступных на данный момент уровней. Необходимо учитывать требования к функциональности приложения и поддерживаемый на реальном устройстве уровень API. Если приложение требует уровень API выше, чем уровень API, поддерживаемый устройством, то приложение не будет установлено.

После выбора целевой версии вам откроется последнее окно мастера — **Application Info** (рис. 3.4), где необходимо ввести информацию о приложении:

- ❑ **Application Name** — заголовок приложения — имя, которое появится в заголовке окна приложения в верхней части экрана. Введите значение `First Android Application`;

- ❑ **Package Name** — имя пакета, которое имеет такие же правила построения и именования пакетов, как и для приложений на Java. Введите значение `com.samples.intro.firstapp`;

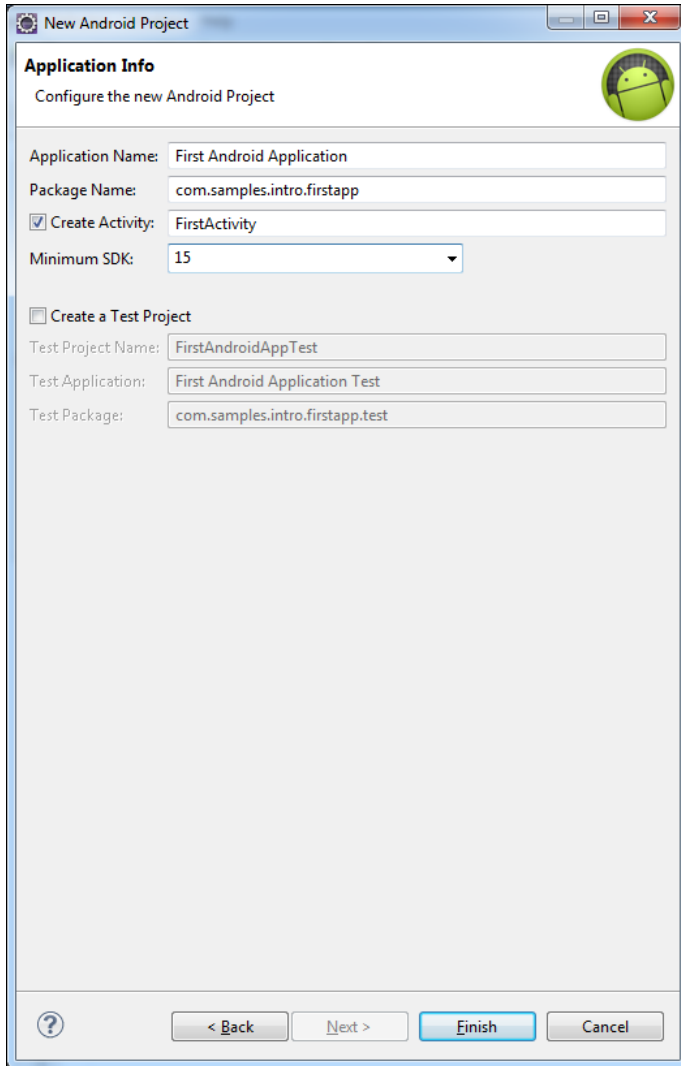


Рис. 3.4. Окно ввода информации о приложении

ОБРАТИТЕ ВНИМАНИЕ

Имя пакета должно быть уникально по отношению ко всем пакетам, установленным на мобильном Android-устройстве. По этой причине очень важно использовать для своих приложений стандартный пакет доменного стиля.

- ❑ **Create Activity** — имя для заглушки класса, которая будет сгенерирована плагином. Заглушка является подклассом класса `Activity`, который будет представлять окно нашего приложения. Переключатель около поля ввода предполагает, что создание

класса является необязательным, но `Activity` почти всегда используется как основа для приложения. Введите в поле значение `FirstActivity`;

- ❑ **Minimum SDK** — минимальный уровень API, требуемый для приложения. У нас будет значение 15.

Нажмите кнопку **Finish**. Мастер New Android Project сгенерирует шаблон для проекта Android-приложения. Это должно быть видно в окне **Package Explorer** слева.

По умолчанию плагин ADT генерирует проект с единственным окном и текстовым полем с надписью "Hello, World" + имя класса `Activity` (главного окна приложения), которое вы определили в поле **Create Activity**. При желании можете поменять надпись, например, на "Hello, Android!". Для этого откройте файл `strings.xml`, находящийся в каталоге `res/values/`, и в появившемся окне редактора ресурсов Resource Editor перейдите в режим текстового редактирования XML и измените значение для элемента `<string name="hello">`:

```
<string name="hello">Hello, Android!</string>
```

Нажмите кнопку **Run** (или используйте комбинацию клавиш `<Ctrl>+<F11>`). Появится окно **Run As** запуска проекта на выполнение. Выберите из списка **Select a way to run 'FirstAndroidApp'** опцию **Android Application** и нажмите кнопку **OK** (рис. 3.5).

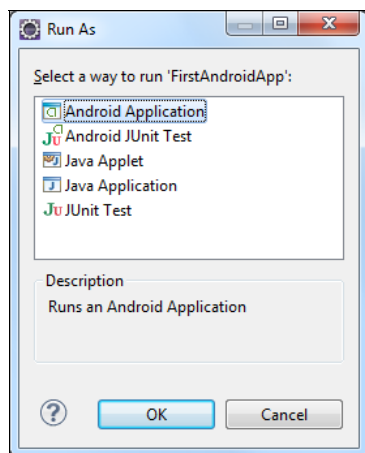


Рис. 3.5. Окно компиляции и запуска проекта

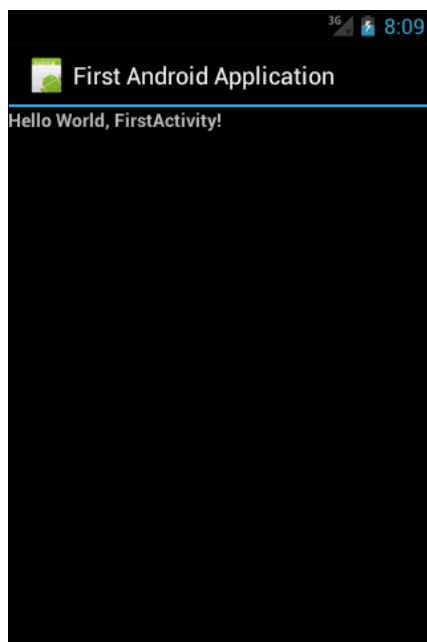


Рис. 3.6. Окно программы First Android Application

Плагин ADT автоматически создаст новую исполняемую конфигурацию для проекта, которая автоматически загрузится и запустится в эмуляторе мобильного устройства. После запуска эмулятора и загрузки операционной системы разблокируйте его, нажав кнопку **MENU**. В окне эмулятора появится наше приложение (рис. 3.6).

В окне программы мы увидим две строки. Текст "First Android Application", который виден в серой области вверху, — это заголовок приложения (его имя мы задавали в поле **Application Name** окна **New Android Project**). Текст ниже заголовка — это текст из файла строковых ресурсов `strings.xml`, автоматически сгенерированный средой разработки. По умолчанию при создании шаблона приложения среда разработки создает окно приложения с текстовым полем, содержимое которого состоит из текста "Hello World," плюс имя класса главного окна приложения, в нашем случае — `FirstActivity`.

При выполнении команды **Run** среда разработки автоматически устанавливает его на Android-эмулятор (или мобильное устройство).

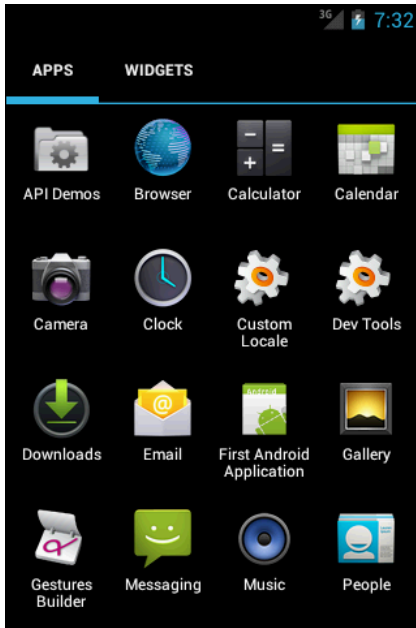


Рис. 3.7. Отображение приложения в окне **Application Launcher**

Приложения, установленные на эмулятор, сохраняются на нем и при его закрытии. Если необходимо заново запустить созданное приложение, это можно сделать из окна **Application Launcher** Android-эмулятора (рис. 3.7).

При компиляции приложения в каталоге проекта создается папка `bin/`. Скомпилированный код Java-классов вместе с файлами данных и ресурсов помещается в архивный файл с расширением `APK` в каталоге `bin/`. Этот файл используется для распространения приложения и установки его на мобильных устройствах. Файлы `APK` на Android-устройстве (или эмуляторе) помещаются в каталог `data/app/`.

Целевую платформу (версию Android SDK), на которую рассчитан проект, всегда можно поменять в большую или меньшую сторону. Например, у вас есть старый проект, созданный для версии Android 1.6, а вы хотите добавить в него новую функциональность, которая появилась только в версии 4.0.3. Для этого выберите в меню пункт **Project | Properties** и в открывшемся окне **Properties for FirstAndroidApp** на левой панели выберите узел **Android**. В правой части окна отобразится панель **Android**, где в группе **Project Build Target** можно будет выбрать другую целевую платформу (рис. 3.8).

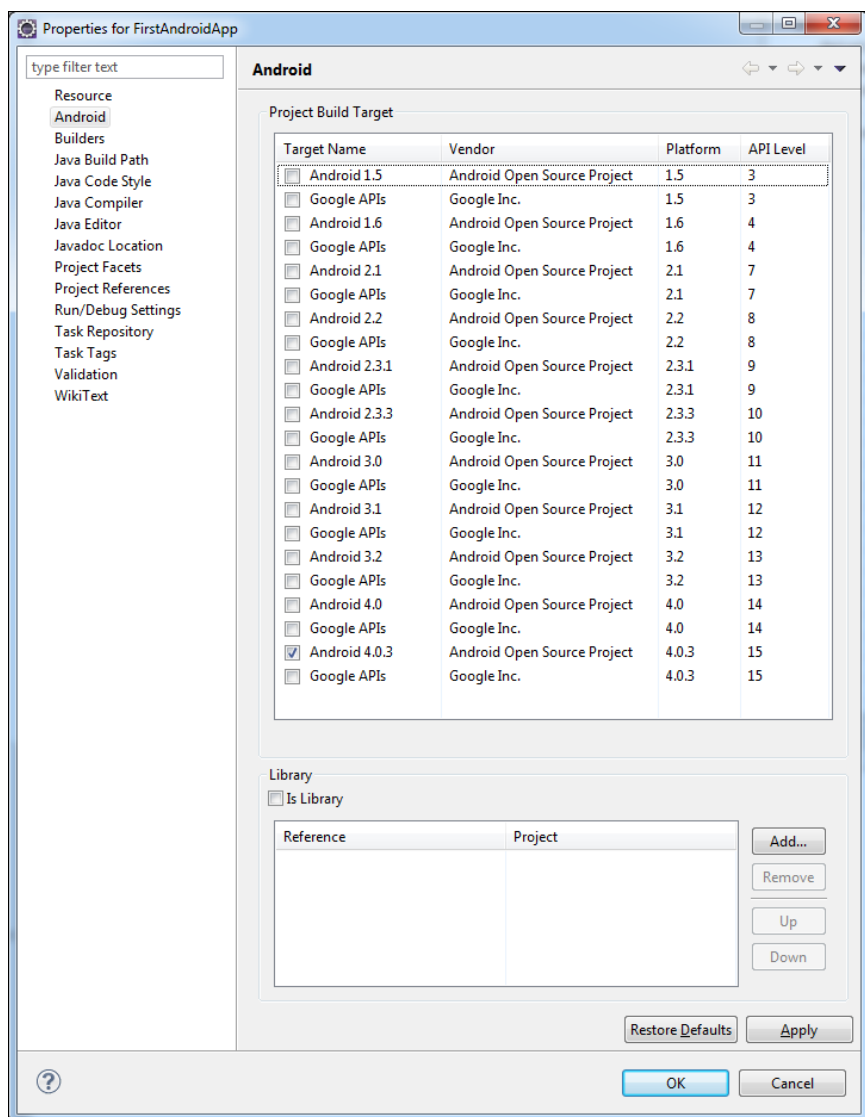


Рис. 3.8. Окно свойств проекта

Структура проекта

Рассмотрим сейчас файлы и каталоги, которые были созданы для нас средой разработки. ADT-плагин при создании Android-проекта организует структуру в виде дерева каталогов, как и любой другой Java-проект. В среде Eclipse, в окне **Package Explorer**, можно видеть структуру созданного проекта (рис. 3.9).

ПРИМЕЧАНИЕ

Структура файлов и каталогов проекта может меняться в зависимости от уровня API, установленного для проекта. Например, начиная с уровня 7 (версия Android 2.1-update1) и выше

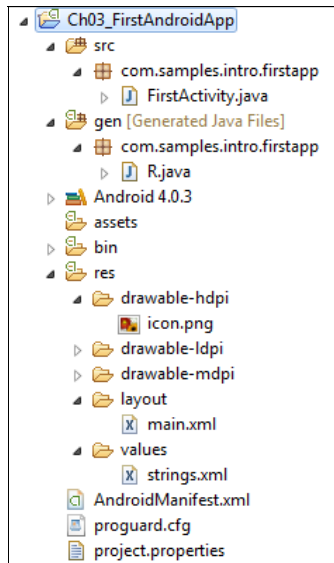
в каталоге `res/` создаются три подкаталога: `drawable-hdpi/`, `drawable-mdpi/`, `drawable-ldpi/` с иконками для разных типов экрана мобильного устройства. Для более старых версий Android SDK среда разработки создает только один каталог `drawable/`.

Рассмотрим теперь подробнее каталоги и файлы, созданные в проекте.

ПРИМЕЧАНИЕ

Исходные коды приложения находятся в каталоге `Ch03_FirstAndroidApp`.

Рис. 3.9. Структура проекта `FirstAndroidApp` в окне **Package Explorer**



Каталоги ресурсов

В этом каталоге хранятся используемые в приложении статические файлы ресурсов: изображения, строки, анимация и др. Некоторые из подкаталогов создаются ADT-плагином, другие необходимо добавлять самостоятельно, используя предопределенные имена. Обычно в ресурсы включают следующие подкаталоги:

- ❑ `res/drawable-hdpi/`, `res/drawable-ldpi/`, `res/drawable-mdpi/` — для изображений (PNG, JPEG и т. д.). Изображения в каждой папке рассчитаны на соответствующее разрешение экрана мобильного устройства;
- ❑ `res/layout/` — для XML-файлов компоновки (компоновка графических элементов управления для окон приложения);
- ❑ `res/menu/` — для XML-файлов меню;
- ❑ `res/values/` — для строковых ресурсов, массивов и т. д.;
- ❑ `res/xml/` — для других XML-файлов, которые понадобятся для разработки приложения.

Здесь следует отметить несколько ограничений относительно создания папок файлов ресурсов проекта. Во-первых, Android поддерживает только линейный список файлов в пределах предопределенных папок под каталогом `res/`. Например, он не поддерживает вложенные папки под папкой для XML-файлов компоновки (или другими папками в каталоге `res/`). Более подробно эти каталоги мы будем рассматривать в *главе 17*, когда будем изучать работу с ресурсами.

Подкаталог `res/layout/`

В подкаталог `res/layout/` помещаются файлы компоновки в формате XML, которые определяют внешний вид окна `Activity` и расположение на нем элементов управления. Каждый файл компоновки представляет собой окно приложения. В нашем проекте он

единственный и по умолчанию называется `main.xml`. Если щелкнуть мышью по файлу `main.xml`, откроется Layout Editor — редактор компоновки (рис. 3.10).

Плагин ADT по умолчанию генерирует базовую компоновку для главного окна приложения с текстовым полем и надписью "Hello World, <Имя_класса_Activity>!". Редактор компоновки имеет два режима отображения: графический и текстовый, которые переключаются закладками в нижней части окна. В текстовом виде файл `main.xml` показан в листинге 3.1.

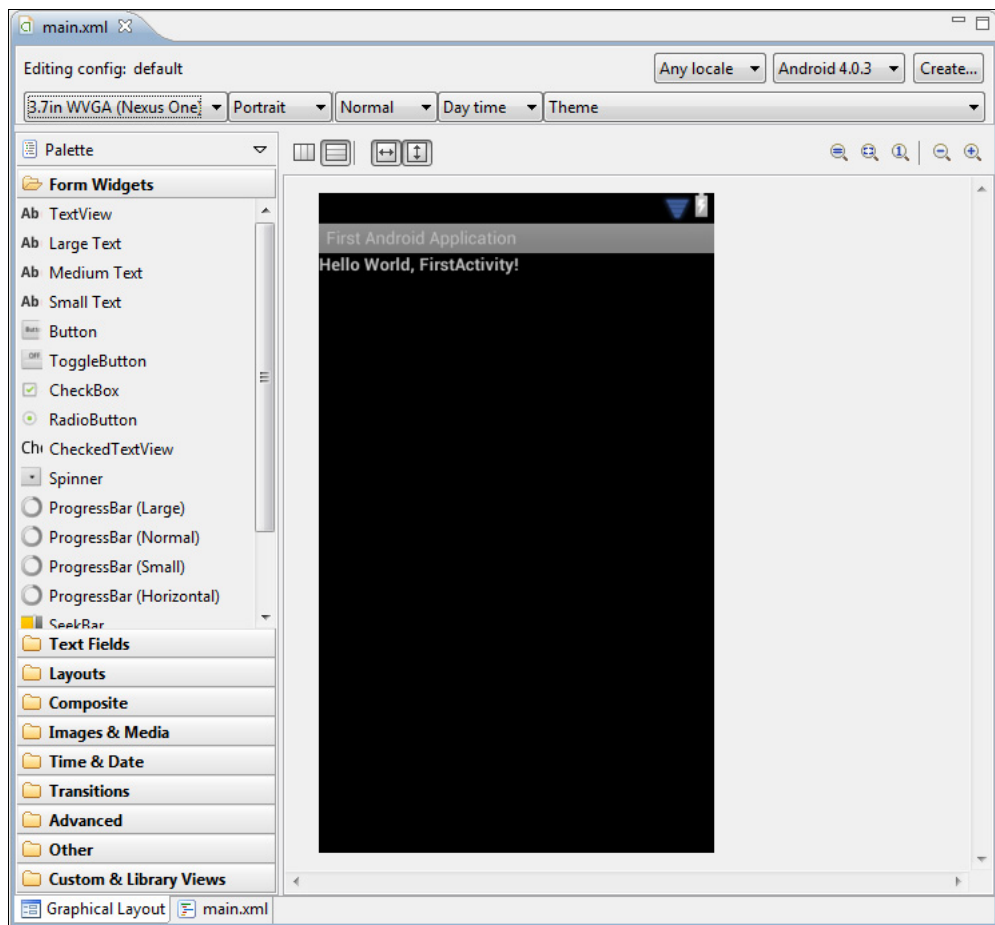


Рис. 3.10. Редактор компоновки

Листинг 3.1. Файл компоновки `main.xml`

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

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:textStyle="bold"/>
</LinearLayout>
```

Сейчас мы не будем изучать содержимое этого файла. Создание файлов компоновки будет рассматриваться далее, в *главе 5*, а элементов управления — в *главах 6—8*.

Подкаталоги res/drawable/

В подкаталогах res/drawable-hdpi, res/drawable-ldpi, res/drawable-mdpi, которые предусмотрены для различных разрешений экрана, размещаются все графические файлы, используемые в приложении. На данный момент в этих подкаталогах содержится только файл icon.png — значок приложения, по умолчанию устанавливаемый для приложения мастером создания проекта и отображаемый в меню запуска установленных на телефоне приложений (**Application Launcher**).

Подкаталог res/values/

В подкаталоге res/values расположены XML-файлы, в которых хранятся общие константы для всего приложения: текст, используемый элементами управления, цвета, стили и т. д. Например, если мы хотим вывести "Hello, Android!" в текстовое поле, можно это сделать двумя способами:

- ❑ написать явно в файле компоновки или в файле манифеста;
- ❑ создать в strings.xml константу hello со значением "Hello, Android!", а в файле компоновки в атрибуте android:text для элемента TextView указать ссылку на ресурс в strings.xml, как в листинге 3.1:

```
android:text="@string/hello"
```

Если щелкнуть мышью по файлу strings.xml, откроется редактор ресурсов, представленный на рис. 3.11.

Файл ресурсов, так же как и файлы компоновки, можно редактировать в графическом и текстовом режимах. Текстовое представление файла строковых ресурсов для нашего приложения приведено в листинге 3.2.

Листинг 3.2. Файл ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, FirstActivity!</string>
    <string name="app_name">First Android Application</string>
</resources>
```

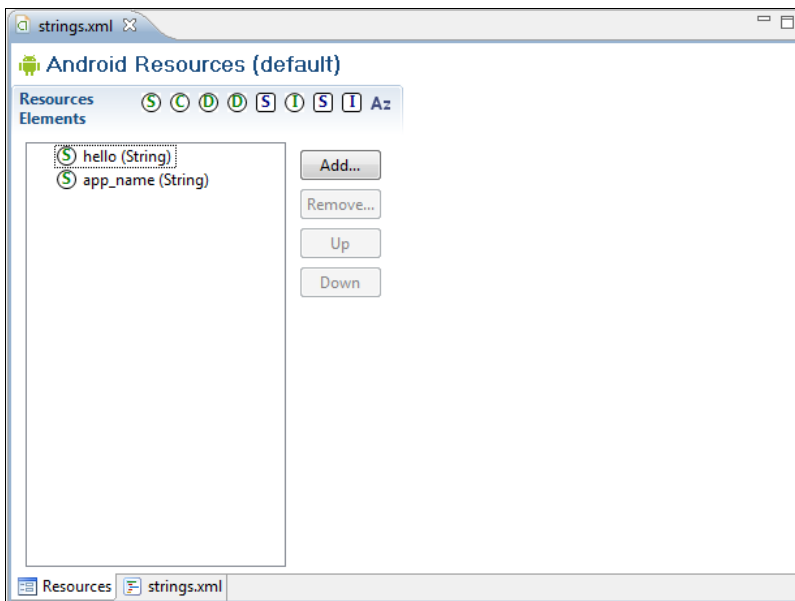



Рис. 3.11. Редактор ресурсов

Файл R.java

В каталоге `gen/имя_пакета/` (в нашем случае — `gen/com.samples.firstapp/`) расположен файл `R.java`. Когда проект компилируется первый раз, среда разработки создает класс `R` и помещает его в файл `R.java`. Этот класс используется в коде программы для обращения к ресурсам, которые находятся в каталоге `res/`.

Пример файла `R.java` для нашего приложения показан в листинге 3.3.

Листинг 3.3. Файл класса `R.java`

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */
/* АВТОМАТИЧЕСКИ СГЕНЕРИРОВАННЫЙ ФАЙЛ. НЕ МОДИФИЦИРОВАТЬ.
 *
 * Этот класс сгенерирован автоматически
 * инструментом aapt на основе созданных ресурсов.
 * Не модифицировать этот файл вручную.
 */
package com.samples.intro.helloandroid;

public final class R {
    public static final class attr {
    }
```

```

public static final class drawable {
    public static final int icon=0x7f020000;
}
public static final class layout {
    public static final int main=0x7f030000;
}
public static final class string {
    public static final int app_name=0x7f040001;
    public static final int hello=0x7f040000;
}
}

```

Класс `R` содержит набор внутренних классов с идентификаторами ресурсов, которые он создает в зависимости от внутреннего содержимого каталога `res/`:

- ▢ `drawable` — для каталога `res/drawable/`;
- ▢ `layout` — для каталога `res/layout/`. Содержит идентификаторы файлов компоновки. В нашем приложении только один файл компоновки — `main.xml`, сгенерированный мастером создания проекта. Если в приложении определены несколько `Activity`, для каждого из них необходимо будет определять файл компоновки;
- ▢ `string` — для идентификаторов строк в файле `strings.xml`;
- ▢ `attr` — для дополнительных атрибутов, определяемых во внешнем XML-файле.

При добавлении в ресурсы других файлов среда сгенерирует новый класс `R`, добавив в него дополнительные вложенные классы, например `id` — для идентификаторов элементов компоновки (о них мы расскажем в *главе 5* и далее).

ОБРАТИТЕ ВНИМАНИЕ

Вы никогда не должны редактировать этот файл вручную, т. к. при компиляции проекта среда разработки все равно его перезапишет.

Файл окна приложения `FirstActivity.java`

В каталоге `src/имя_пакета/` (в нашем случае — `src/com.samples.firstapp/`) находится файл `HelloAndroidActivity.java` — это класс, автоматически генерируемый ADT-плагином для главного `Activity` (окна) приложения.

Плагин определяет в классе метод обратного вызова `onCreate()`, который вызывается системой для прорисовывания окна `Activity` на экране устройства. В этот класс разработчик может добавлять код, реализующий логику работы приложения в данном `Activity`.

По умолчанию класс, создаваемый средой разработки, является расширением класса `Activity`, для нашего приложения показан в листинге 3.4.

Листинг 3.4. Файл окна приложения `FirstActivity.java`

```

package com.samples.intro.firstapp;

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

```

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

Если приложение будет иметь несколько окон, для каждого из них будет создан отдельный класс, наследуемый от базового класса `Activity`. В следующих главах будет подробно рассмотрено написание классов, реализующих `Activity` в приложении.

Файл `AndroidManifest.xml`

Файл манифеста приложения — структурный XML-файл, который всегда имеет название `AndroidManifest.xml` для всех приложений. Он задает конфигурацию приложения: объявляет компоненты приложения, перечисляет любые библиотеки, связанные с приложением (помимо библиотек Android, связанных по умолчанию), и объявляет разрешения, которые требуются для работы приложения (например, доступ в сеть, разрешение на отправку SMS и т. д.).

Код файла манифеста приведен в листинге 3.5.

Листинг 3.5. `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.intro.firstapp"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".FirstActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Например, атрибут `android:name` элемента `<activity>` вызывает подкласс `Activity`, который реализует `Activity` (окно) в приложении. Атрибуты `icon` и `label` прикрепляют файл-

лы ресурсов, содержащих значок и текст, которые могут быть отображены пользователю.

Аналогичным способом объявляются и другие компоненты приложения. В следующей главе будет подробно рассказано о работе с файлом манифеста и о том, как требуемую функциональность Android-приложения можно объявить в файле манифеста.

Прежде чем система Android запустит компонент приложения, она должна узнать, что этот компонент существует. Поэтому приложения объявляют свои компоненты в файле манифеста `AndroidManifest.xml`, который предоставляет основную информацию системе. Каждое приложение должно иметь свой файл `AndroidManifest.xml`.

Файл манифеста приложения выполняет следующие функции:

- ❑ объявляет имя Java-пакета данного приложения. Имя пакета служит уникальным идентификатором для приложения;
- ❑ описывает компоненты приложения — `Activity`, `Service`, `Broadcast Receiver` и `Content Provider`, из которых состоит данное приложение. Эти объявления позволяют системе Android знать, чем компоненты являются и при каких условиях они могут быть запущены;
- ❑ объявляет, какие разрешения должно иметь приложение для обращения к защищенным системным службам и взаимодействия с компонентами других приложений;
- ❑ объявляет разрешения, которые сторонние приложения обязаны иметь, чтобы взаимодействовать с компонентами данного приложения;
- ❑ объявляет минимальный уровень API Android, которого требует приложение;
- ❑ перечисляет библиотеки, с которыми приложение должно быть связано.

Редактировать файл манифеста можно вручную, записывая XML-код непосредственно в файл, или через визуальный редактор. Для работы с файлом манифеста в Eclipse есть отдельный инструмент — `Manifest Editor` (Редактор файла манифеста), который позволяет визуальное и текстовое редактирование файла манифеста приложения, как показано на рис. 3.12.

Общая структура манифеста

Файл манифеста инкапсулирует всю архитектуру Android-приложения, его функциональные возможности и конфигурацию. В процессе разработки приложения вам придется постоянно редактировать файл, изменяя его структуру и дополняя его новыми элементами и атрибутами по мере усложнения разрабатываемого приложения, поэтому важно хорошо ориентироваться во внутренней структуре манифеста и назначении его элементов и атрибутов.

На рис. 3.13 приведена общая структура файла манифеста и элементов, которые содержатся в нем, и назначение каждого из элементов.

Порядок расположения элементов, находящихся на одном уровне, произвольный. Все значения устанавливаются через атрибуты элементов. Элемент `<application>` является основным элементом манифеста и содержит множество дочерних элементов, определяющих структуру и работу приложения. Элементы `<manifest>`, `<application>` и `<uses-sdk>`

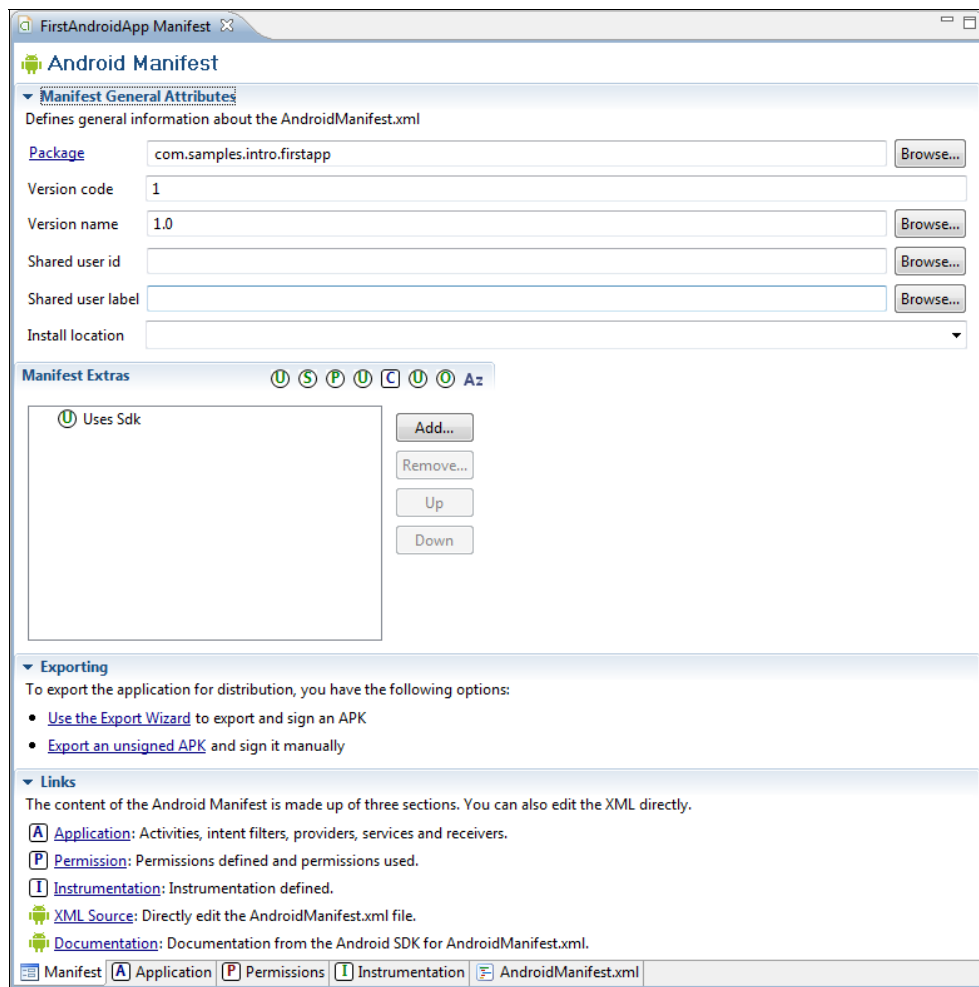


Рис. 3.12. Редактор файла манифеста

являются обязательными. Другие элементы необязательные и используются в манифесте при необходимости.

Элемент `<manifest>` является корневым элементом файла `AndroidManifest.xml`. По умолчанию мастер создания Android-проекта в Eclipse создает элемент с четырьмя атрибутами:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.programmingandroid.helloandroid"
    android:versionCode="1"
    android:versionName="1.0">
```

Эти атрибуты обязательны для любого Android-приложения и имеют следующее назначение:

- `xmlns:android` — пространство имен Android. Это значение всегда неизменно для всех приложений;



Рис. 3.13. Структура файла манифеста

- ☐ `package` — имя пакета приложения, которое вы определили при создании проекта приложения;
- ☐ `android:versionCode` — внутренний номер версии;
- ☐ `android:versionName` — номер пользовательской версии. Этот атрибут может быть установлен как строка или как ссылка на строковый ресурс.

Элемент `<permission>` объявляет разрешение, которое используется для ограничения доступа к определенным компонентам или функциональности данного приложения. В этой секции описываются права, которые должны запросить другие приложения для получения доступа к вашему приложению.

Приложение может также защитить свои собственные компоненты (Activity, Service, Broadcast Receiver и Content Provider) разрешениями. Оно может использовать любое из системных разрешений, определенных Android (перечисленных в `android.Manifest.permission`) или объявленных другими приложениями, а также может определить свои собственные разрешения. Новое разрешение должно быть объявлено в атрибуте `android:name` элемента `<permission>` следующим образом:

```
permission android:name="com.samples.custom_permission"
```

Кроме того, используются дополнительные атрибуты:

- ☐ `android:label` — имя разрешения, отображаемое пользователю;
- ☐ `android:description` — описание;
- ☐ `android:icon` — иконка, представляющая разрешение;
- ☐ `android:permissionGroup` — определяет принадлежность к группе разрешений;
- ☐ `android:protectionLevel` — уровень защиты.

Элемент `<uses-permission>` запрашивает разрешения, которые приложению должны быть предоставлены системой для его нормального функционирования. Разрешения предоставляются во время установки приложения, а не во время его работы.

Этот элемент имеет единственный атрибут — с именем разрешения — `android:name`. Это может быть разрешение, определенное в элементе `<permission>` данного приложения, разрешение, определенное в другом приложении, или одно из стандартных системных разрешений, например, если требуется получить доступ к базе данных контактов:

```
android:name="android.permission.READ_CONTACTS"
```

Элемент `<permission-tree>` объявляет базовое имя для дерева разрешений. Этот элемент объявляет не само разрешение, а только пространство имен, в которое могут быть помещены дальнейшие разрешения.

Элемент `<permission-group>` определяет имя для набора логически связанных разрешений. Это могут быть как объявленные в этом же манифесте с элементом `<permission>` разрешения, так и объявленные в другом месте. Этот элемент не объявляет разрешение непосредственно, только категорию, в которую могут быть помещены разрешения. Разрешение можно поместить в группу, назначив имя группы в атрибуте `permissionGroup` элемента `<permission>`.

Элемент `<instrumentation>` объявляет объект `Instrumentation`, который дает возможность контролировать взаимодействие приложения с системой. Обычно используется при отладке и тестировании приложения и удаляется из release-версии приложения.

Элемент `<uses-sdk>` позволяет объявлять совместимость приложения с указанной версией (или более новыми версиями API) платформы Android. Уровень API, объявленный приложением, сравнивается с уровнем API системы мобильного устройства, на который устанавливается данное приложение.

Основной используемый в элементе атрибут — `minSdkVersion`, определяющий минимальный уровень API, требуемый для работы приложения. Система Android будет препятствовать тому, чтобы пользователь установил приложение, если уровень API системы будет ниже, чем значение, определенное в этом атрибуте. Желательно всегда объявлять этот атрибут, например:

```
<uses-sdk android:minSdkVersion="10"/>
```

Попробуйте в нашем тестовом проекте уменьшить значение `minSdkVersion` до 10 и перекомпилируйте проект. При запуске его в эмуляторе вы увидите, что окно приложения изменилось — исчез `Action Bar` (значок приложения и надпись) и вместо него появился заголовок окна только с надписью `First Android Application` (рис. 3.14).

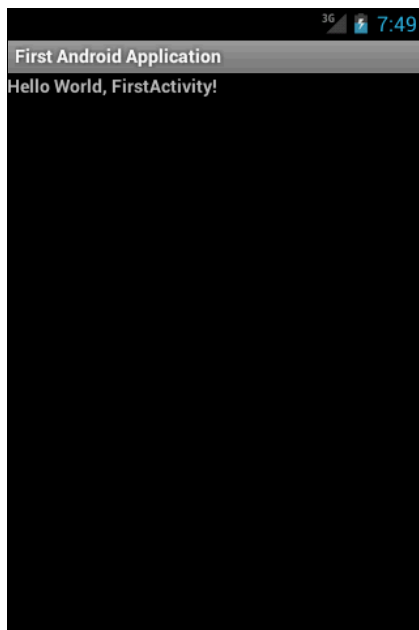


Рис. 3.14. Внешний вид приложения с измененной версией `minSdkVersion`

Action Bar стал доступен только с версии 3.0 (API Level 11), предназначенной для планшетных ПК. Поэтому при установке значения `android:minSdkVersion` меньше 11 Action Bar будет скрыт, несмотря на то, что мы используем эмулятор версии 4.03 (API Level 15).

Элемент `<uses-configuration>` указывает требуемую для приложения аппаратную и программную конфигурацию мобильного устройства. Например, приложение могло бы определить требование обязательного наличия на устройстве физической клавиатуры или порта USB. Спецификация используется, чтобы избежать инсталляции приложения на устройствах, которые не поддерживают требуемую конфигурацию.

Если приложение может работать с различными конфигурациями устройства, необходимо включить в манифест отдельные элементы `<uses-configuration>` для каждой конфигурации.

Элемент `<uses-feature>` объявляет определенную функциональность, требующуюся для работы приложения. Таким образом, приложение не будет установлено на устройствах, которые не имеют требуемую функциональность.

Например, приложение могло бы определить, что оно требует камеру с автофокусом. Если устройство не имеет встроенную камеру с автофокусом, приложение не будет инсталлировано.

Элемент `<supports-screens>` определяет разрешение экрана, требуемое для функционирования устройства (для старых версий Android-устройств). По умолчанию современное приложение с уровнем API 4 и выше поддерживает все размеры экрана и должно игнорировать этот элемент.

Элемент `<application>` имеет гораздо более сложную структуру, чем все перечисленные элементы, поэтому мы его рассмотрим отдельно.

Структура элемента `<application>`

Элемент `<application>` — это очень важный элемент манифеста, содержащий описание компонентов приложения, доступных в пакете. Этот элемент содержит дочерние элементы, которые объявляют каждый из компонентов, входящих в состав приложения Android, описание и принципы работы которых были кратко представлены в *главе 1* (эти компоненты мы будем подробно рассматривать в следующих главах).

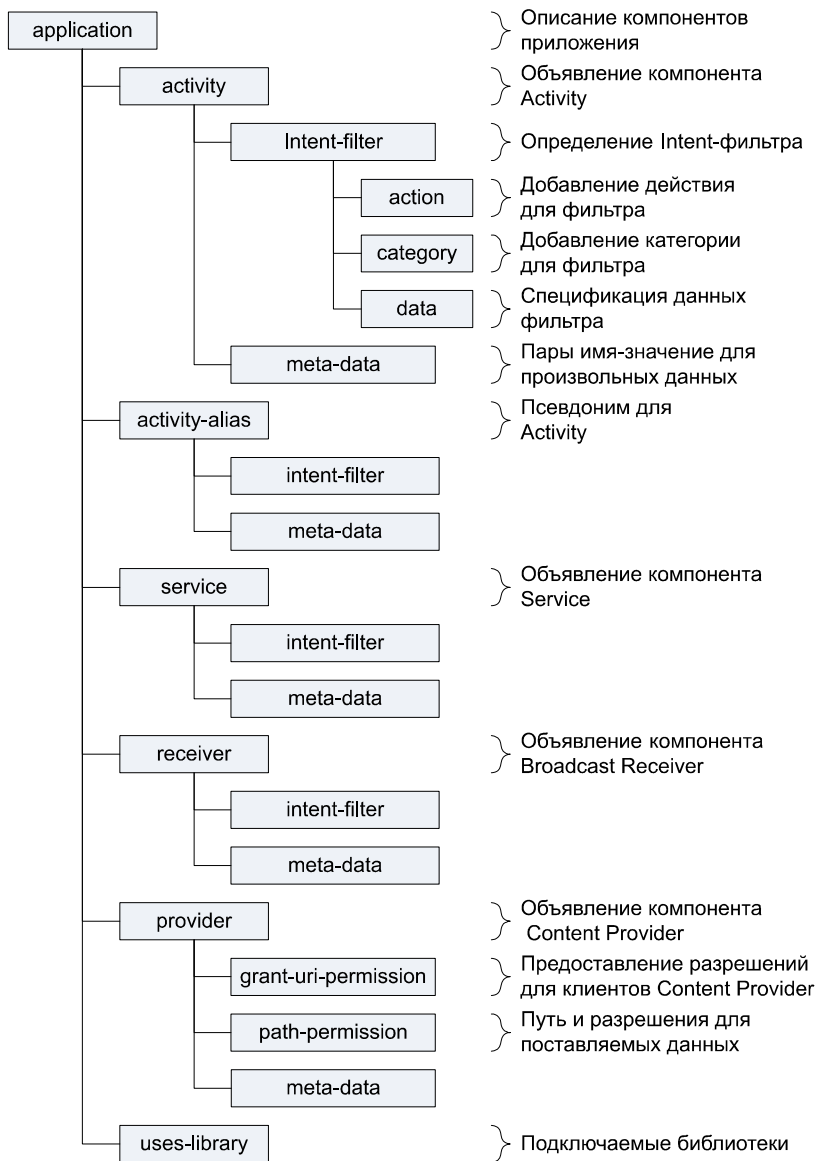
Структура элемента `<application>` и назначение его дочерних элементов представлены на рис. 3.15.

Дочерний элемент `<activity>` объявляет компонент Activity. Все компоненты Activity должны быть явно представлены отдельными элементами `<activity>` в файле манифеста, например:

```
<activity android:name="com.samples.helloandroid.HelloAndroid"
        android:label="@string/app_name">
```

Атрибуты элемента `<activity>` являются обязательными и определяют следующее:

- ❑ `android:name` — имя класса. Имя должно включать полное обозначение пакета, но так как имя пакета уже определено в корневом элементе `<manifest>`, имя класса, реализующего Activity, можно записывать в сокращенном виде, опуская имя пакета:
`android:name="com.samples.helloandroid.HelloAndroid"`
- ❑ `android:label` — текстовая метка, отображаемая пользователю в заголовке Activity.

Рис. 3.15. Структура элемента `<application>`

Кроме вышеперечисленных элемент `<activity>` содержит множество атрибутов, определяющих разрешения, ориентацию экрана и т. д. Если приложение содержит несколько Activity, не забывайте объявлять их в манифесте, создавая для каждого из них свой элемент `<activity>`. Если данный Activity не объявлен в манифесте, он не будет виден системе и не будет запущен при выполнении приложения.

Элемент `<intent-filter>` определяет типы Intent, на которые могут ответить Activity, Service или Broadcast Receiver. Intent-фильтр предоставляет для компонентов-клиентов возможность получения Intent объявляемого типа, отфильтровывая те, которые не значимы для компонента, и содержит дочерние элементы `<action>`, `<category>`, `<data>`.

Элемент `<action>` добавляет действие к Intent-фильтру. Элемент `<intent-filter>` должен содержать один или более элементов `<action>`. Если в элементе `<intent-filter>` не будет этих элементов, то объекты Intent не пройдут через фильтр. Пример объявления действия:

```
<action android:name="android.intent.action.MAIN">
```

означает, что этот Activity в данном приложении будет главным и, когда система пришлет Intent для запуска приложения, этот Activity откроется по умолчанию.

Элемент `<category>` определяет категорию компонента, которую должен обработать Intent. Это строковые константы, определенные в классе Intent. Например, при создании приложения среда вставляет следующее:

```
<category android:name="android.intent.category.LAUNCHER" />
```

Этот атрибут означает, что значок приложения будет отображаться в окне запуска приложений **Application Launcher** мобильного устройства (см. рис. 3.7).

Элемент `<data>` добавляет спецификацию данных к фильтру Intent. Спецификация может быть только типом данных (атрибут `contentType`), URI или типом данных вместе с URI. Значение URI определяется отдельными атрибутами для каждой из его частей, т. е. URI делится на части: `android:scheme`, `android:host`, `android:port`, `android:path` или `android:pathPrefix`, `android:pathPattern`.

Элемент `<meta-data>` определяет пару имя-значение для элемента дополнительных произвольных данных, которыми можно снабдить родительский компонент. Составляющий элемент может содержать любое число элементов `<meta-data>`.

Элемент `<activity-alias>` — это псевдоним для Activity, определенного в атрибуте `targetActivity`. Целевой Activity должен быть в том же приложении, что и псевдоним, и должен быть объявлен перед псевдонимом Activity в манифесте. Псевдоним представляет целевой Activity как независимый объект. У элемента `<activity-alias>` может быть свой собственный набор Intent-фильтров.

Элементы `<service>`, `<receiver>` и `<provider>` объявляют соответственно компоненты Service, Broadcast Receiver и Content Provider. Не забывайте, компоненты, которые не были объявлены, не будут обнаружены системой и никогда не будут запущены. Эти элементы имеют много атрибутов, определяющих имя, доступность, разрешения, процесс и т. д.

Элемент `<uses-library>` определяет общедоступную библиотеку, с которой должно быть скомпоновано приложение. Этот элемент указывает системе на необходимость включения кода библиотеки в загрузчик классов для пакета приложения.

Каждый проект связан по умолчанию с библиотеками Android, в которые включены основные пакеты для сборки приложений (с классами общего назначения, например: Activity, Service, Intent, View, Button, Application, ContentProvider и т. д.). Однако некоторые пакеты (например, `maps` и `awt`) находятся в отдельных библиотеках, которые автоматически не компонуются с приложением. Если же приложение использует пакеты из этих библиотек или других от сторонних разработчиков, необходимо сделать явное связывание с этими библиотеками и манифест обязательно должен содержать отдельный элемент `<uses-library>`.

Резюме

Мы рассмотрели важную тему — архитектуру Android-приложения. Данная глава закладывает основу для последующей разработки Android-приложений, которую мы будем изучать на протяжении всей книги.

В следующей главе мы рассмотрим способы и инструменты для отладки программ, а также подключение и настройку мобильного устройства Android для тестирования и отладки приложений.



ГЛАВА 4

Отладка приложений

В этой главе мы рассмотрим инструменты для отладки приложений и подключение реального мобильного устройства к компьютеру для развертывания и отладки на нем Android-приложений.

Отладку приложений можно производить двумя способами: в среде Eclipse, как обычное приложение Java, и с помощью инструмента, входящего в состав Android SDK — Dalvik Debug Monitor Server (DDMS).

Отладка в среде Eclipse

Интегрированная среда разработки Eclipse поддерживает отладку приложения как на эмуляторе Android, так и на реальных устройствах Android. Если нажать клавишу <F11> (режим отладки) в Eclipse, среда Eclipse сначала определяет, работает ли экземпляр эмулятора Android уже, или реальное устройство соединяется. Если по крайней мере один эмулятор (или устройство) будет работать, то Eclipse развернет приложение на рабочий эмулятор или соединенное устройство.

Если нет запущенного экземпляра Android или устройств, которые в данный момент соединены с компьютером, Eclipse автоматически запускает экземпляр эмулятора Android и развертывает приложение на него.

Установка контрольных точек является хорошим способом временно приостановить выполнение приложения на эмуляторе и затем исследовать контент переменных и объектов.

Когда выполнение программы достигнет контрольной точки, программа остановится и выведет на экран перспективу Debug, которая будет отображать состояние переменных и много другой отладочной информации (рис. 4.1).

Я не буду здесь подробно рассматривать отладку программы с помощью контрольных точек, думаю, вы с ней все знакомы, даже если до этого не работали в Eclipse. Весь процесс аналогичен отладке в других интегрированных средах разработки — Visual Studio, Net Beans и т. д.

А вот на использовании отладочных инструментов из состава Android SDK мы остановимся более подробно. Одной из основных утилит при отладке программ для Android является Dalvik Debug Monitor Server (DDMS).

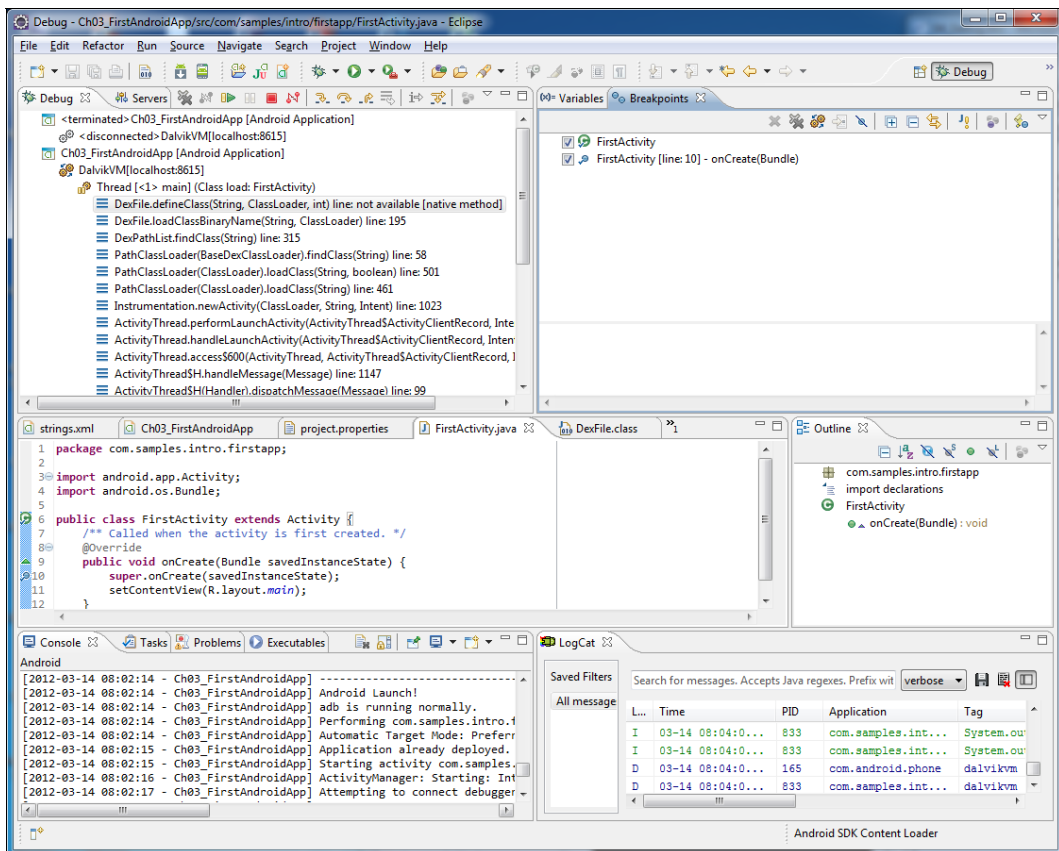


Рис. 4.1. Перспектива Debug в среде Eclipse

Использование DDMS

Запустить DDMS можно из подкаталога `tools/` вашей инсталляции SDK Android или, если вы добавили пути для инструментов Android в переменную окружения `Path` (см. главу 2, рис. 2.5), через командную строку, набрав команду `ddms`. Откроется окно **Dalvik Debug Monitor**, представленное на рис. 4.2.

В левой части окна отображаются запущенные экземпляры эмуляторов. Чтобы подключить DDMS к эмулятору, достаточно выбрать имя эмулятора на панели **Name**. При установке соединения DDMS с эмулятором на этой панели отобразится список запущенных процессов. Чтобы посмотреть информацию о нужном процессе, выделите его мышью. В правой части окна отображается набор закладок, в которых можно посмотреть информацию о процессе и системе в целом — порождаемые этим процессом потоки, работу Dalvik VM, распределение памяти и много другой полезной информации. В нижней части окна DDMS в реальном времени отображаются события, происходящие в системе.

Отдельные компоненты DDMS можно запускать напрямую из среды Eclipse, выбрав в главном меню пункт **Window | Show View | Other**. Откроется окно **Show View**, которое

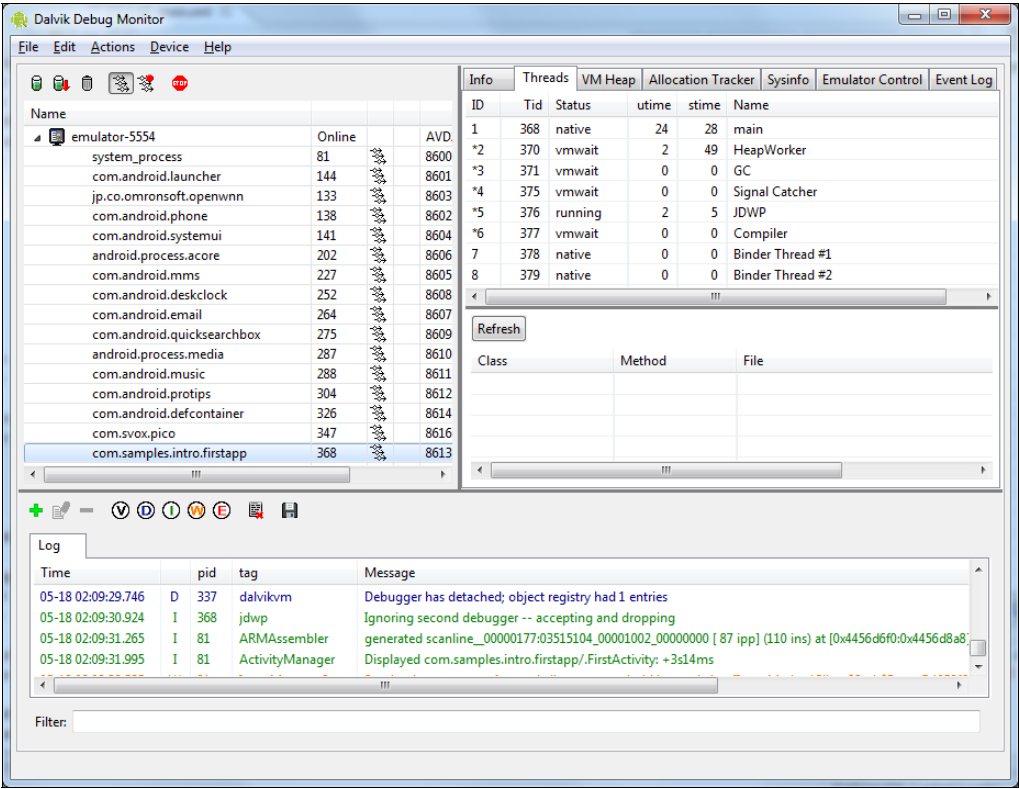


Рис. 4.2. Информация о выполняемом процессе в DDMS

содержит все представления, доступные в вашей среде разработки. Раскрыв группу **Android**, вы увидите набор представлений (рис. 4.3).

Запись в журнал событий

Журнал событий Logcat обеспечивает механизм для сбора и просмотра вывода отладочных сообщений свалки журнала системных сообщений, которые включают в себя такие вещи, как трассировки стека, ошибки, выданные эмулятором, и сообщения, которые вы написали из приложения с помощью журнала класса. Logcat можно запустить через DDMS, который позволяет читать сообщения в реальном времени.

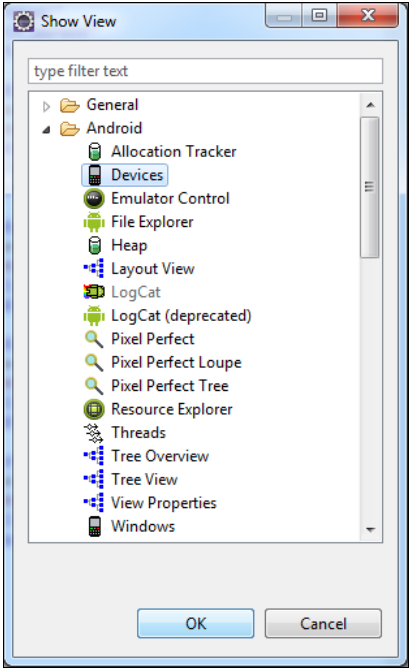


Рис. 4.3. Представления Eclipse из группы **Android**

Для генерирования сообщения в коде приложения используется класс `Log`. Этот класс содержит несколько статических методов:

- ❑ `v(String, String)` — для вывода сообщения о ходе выполнения программы (*verbose*);
- ❑ `d(String, String)` — для вывода отладочного сообщения (*debug*);
- ❑ `i(String, String)` — для вывода информационного сообщения (*information*);
- ❑ `w(String, String)` — для вывода предупреждающего сообщения (*warning*);
- ❑ `e(String, String)` — для вывода сообщения об ошибке (*error*).

Для примера можете добавить в приложение из предыдущей главы пару вызовов для вывода сообщений в журнал, как показано в листинге 4.1.

Листинг 4.1. Вызов методов класса `Log` в программе

```
public void onCreate(Bundle savedInstanceState) {  
  
    super.onCreate(savedInstanceState);  
    Log.d("FirstAndroidApp", "Call onCreate() method for super class");  
  
    setContentView(R.layout.main);  
    Log.d("FirstAndroidApp", "Finished create user interface");  
}
```

Журнал событий Logcat

Журнал событий Logcat интегрирован в DDMS и выводит сообщения, которые вы определяете с помощью методов класса `Log` наряду с другими системными сообщениями, исключениями и т. д. В утилите DDMS журнал событий расположен на нижней панели (см. рис. 4.2). Можно также запустить журнал событий в Eclipse (представление **LogCat** на рис. 4.3).

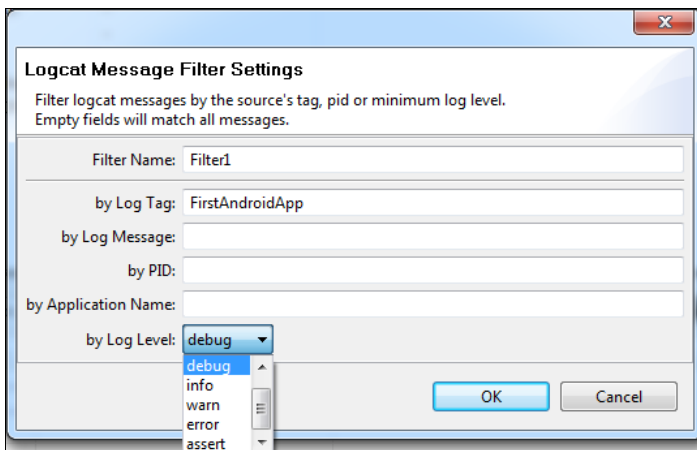


Рис. 4.4. Установка фильтра сообщений для Logcat

В журнал событий выводится много сообщений. При необходимости можно настроить свой собственный пользовательский фильтр сообщений, чтобы фильтровать по имени приложения, идентификаторам процесса или по типу сообщений, как представлено на рис. 4.4.

В результате настроек фильтра можно будет выводить только нужные вам сообщения, например, только те, которые вы определили в коде программы (см. листинг 4.1), как показано на рис. 4.5.

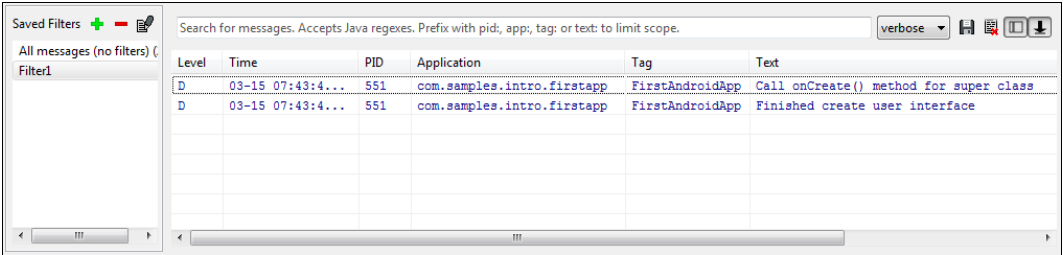


Рис. 4.5. Вывод фильтрованных сообщений для нашего приложения

Настройка мобильного устройства Android для отладки приложения

Эмулятор Android предоставляет широкие возможности для отладки и тестирования приложений. Кроме того, необходимо учитывать, что не все приложения можно отладить на эмуляторе Android. На реальном мобильном Android-устройстве можно тестировать и отлаживать создаваемые приложения так же, как и на эмуляторе Android. Для того чтобы сконфигурировать мобильное устройство для инсталляции, запуска и отладки приложений, необходимо сделать несколько шагов:

- ☐ установить на мобильном телефоне режим отладки через USB;
- ☐ установить на компьютере, где находится среда разработки, драйвер USB для вашего мобильного устройства, если вы до сих пор его не установили;
- ☐ проверить работоспособность соединения с помощью утилиты DDMS.

Установка режима отладки на мобильном телефоне

В мобильном телефоне выберите **Settings** (обычно эта опция выводится на **Home Screen**, если нет — ее можно найти в меню). Затем последовательно выберите **Applications | Development | USB debugging** и поставьте флажок **Debug mode when USB is connected**, как показано на рис. 4.6.

После того как вы выполнили эти действия, подключите ваш мобильный телефон к компьютеру, используя кабель USB, который, как правило, всегда идет в комплекте с мобильным телефоном.

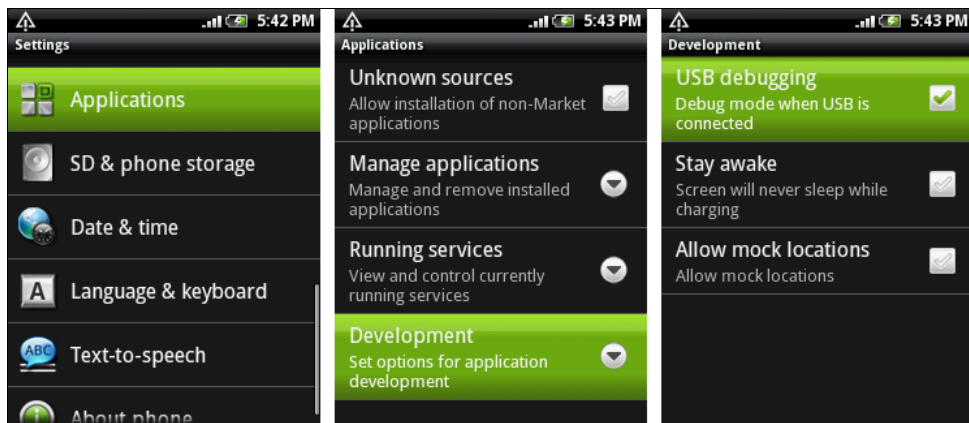


Рис. 4.6. Включение опции **USB debugging** на мобильном телефоне

Установка драйвера USB

Чтобы мобильный телефон соединить с вашим компьютером, на котором установлена среда разработки, потребуется инсталляция USB-драйвера, совместимого с моделью вашего телефона. Здесь возможны два варианта:

- ☐ использовать драйверы USB, поставляемые в составе Android SDK;
- ☐ использовать драйвер USB, предоставляемый производителем данной модели мобильного телефона.

Драйвер USB для Windows доступен для загрузки как дополнительный компонент Android SDK. Драйверы USB для мобильного телефона, поставляемые вместе с Android SDK, расположены в каталоге `\android-sdk-windows\google-usb_driver`.

Однако эти драйверы рассчитаны только на несколько типов моделей телефонов, поэтому лучше выбрать второй вариант и инсталлировать драйвер USB, предоставляемый производителем мобильного телефона. Обычно все производители предлагают отдельные драйверы или пакеты для синхронизации мобильного устройства и компьютера, в комплект которых уже входит драйвер USB.

Процесс установки драйвера USB очень простой, и я не буду его здесь рассматривать, наверняка, вы все умеете это делать.

Взаимодействие устройства Android с DDMS

После подключения телефона (или планшета) необходимо проверить взаимодействие устройства и DDMS. Для этого запустите DDMS. Если все было правильно установлено и настроено, в окне **Dalvik Debug Monitor** на панели **Name** будет отображаться подключенное внешнее мобильное устройство, как показано на рис. 4.7.

Инструмент **Dalvik Debug Monitor Server** работает с реальным мобильным устройством так же, как и с эмулятором Android. Вы можете получать информацию о запущенных процессах, системных событиях, иметь доступ к файловой системе и многим другим функциям, предоставляемым этим инструментом.

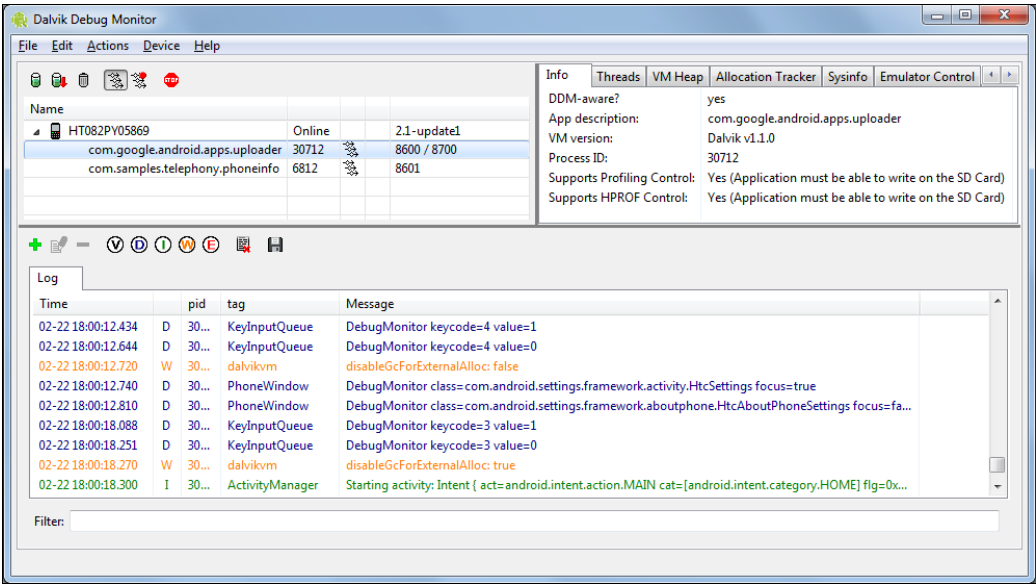


Рис. 4.7. Подключение мобильного устройства к DDMS

Запуск приложения на мобильном устройстве

Запуск приложения на мобильном устройстве ничем не отличается от запуска на эмуляторе Android. Если у вас одновременно запущен эмулятор Android, и подключен мобильный телефон, то среда Eclipse попросит вас выбрать целевой эмулятор (устройство), на котором нужно развернуть приложение. При запуске приложения появится окно **Android Device Chooser**, в котором надо будет выбрать целевое устройство (рис. 4.8).

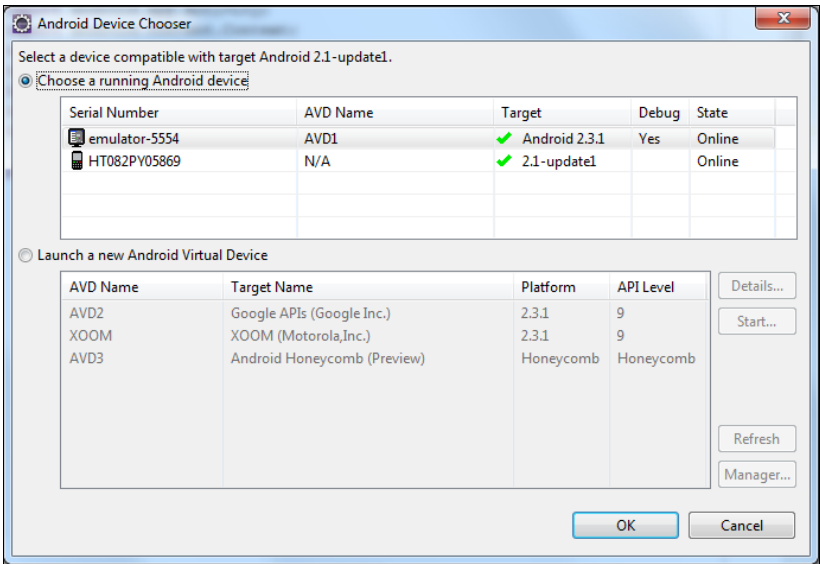


Рис. 4.8. Выбор целевого устройства для инсталляции и запуска проекта

Выберите целевое устройство, которое вы хотите использовать, и нажмите кнопку **ОК**. Программа будет инсталлирована и запущена на выбранном устройстве. Например, наше приложение на смартфоне HTC выглядит так, как показано на рис. 4.9.

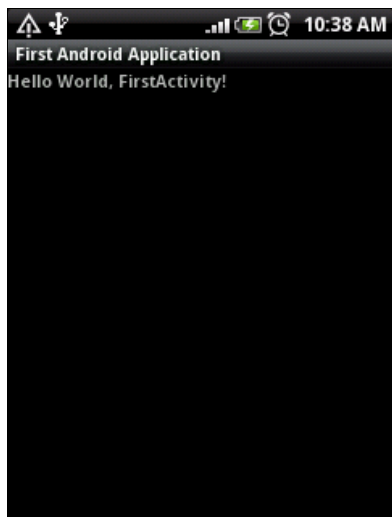


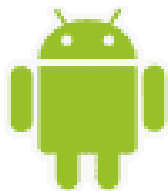
Рис. 4.9. Приложение, запущенное на мобильном устройстве

Для примеров в этой книге мобильный Android-телефон нам понадобится при рассмотрении служб и работе с оборудованием. Конечно, при разработке собственных приложений одним эмулятором ограничиваться не стоит, желательно также тестировать приложения на реальном Android-смартфоне или планшетном ПК.

Резюме

Мы рассмотрели способы отладки приложений и возможности для отладки, предоставляемые интегрированной средой разработки Eclipse и инструментом DDMS из состава Android SDK.

Далее мы переходим к новой части, в которой рассмотрим принципы создания графического интерфейса пользователя. В этой части мы будем изучать создание компоновки для окна приложения, использование элементов управления, создание диалоговых окон, меню и уведомлений.



ЧАСТЬ II

Графический интерфейс пользователя

Глава 5. Компоновка элементов управления

Глава 6. Базовые виджеты

Глава 7. Командные элементы управления и обработка событий

Глава 8. Отображение длительно выполняющихся задач

Глава 9. Уведомления

Глава 10. Диалоговые окна

Глава 11. Меню

Глава 12. Activity

Глава 13. Доступ к компонентам через разрешения

Глава 14. Фрагменты



ГЛАВА 5

Компоновка элементов управления

Компоновка — это архитектура расположения элементов интерфейса пользователя для конкретного окна, представляющего Activity. Компоновка определяет структуру расположения элементов в окне и содержит все элементы, которые отображаются пользователю программы.

Это довольно важная тема в разработке мобильных приложений, поскольку проектирование пользовательского интерфейса для мобильных телефонов и планшетов сложнее, чем для настольных систем или для Web-страниц. Экраны мобильных телефонов имеют гораздо меньшее разрешение, чем обычные мониторы. Кроме того, существует много разновидностей дисплеев для мобильных телефонов, отличающихся размерами, разрешением и плотностью пикселей.

Необходимо также учесть, что большинство экранов для мобильных телефонов сенсорные, причем они могут быть разного типа. Например, емкостный экран реагирует на касание пальцем, а для взаимодействия с резистивным экраном используется стилус. Поэтому важно правильно задавать компоновку и размеры элементов управления, чтобы пользователю было удобно управлять вашим приложением независимо от типа экрана.

В этой главе мы уделим внимание созданию общей архитектуры окна приложения, сами элементы управления мы рассмотрим в следующих главах.

Формирование графического интерфейса пользователя

В Android-приложении графический интерфейс пользователя формируется с использованием объектов View и ViewGroup. Класс View является базовым классом для ViewGroup и состоит из коллекции объектов View (рис. 5.1). Есть множество типов View и ViewGroup, каждый из которых является потомком класса View.

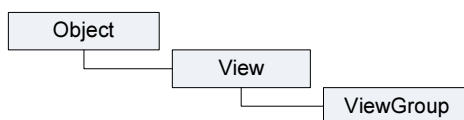


Рис. 5.1. Иерархия классов View и ViewGroup

Объекты `View` — это основные модули для создания графического интерфейса пользователя на платформе Android. Класс `View` служит базовым для классов элементов управления, называемых виджетами, — текстовых полей, кнопок и т. д. Объект `View` является структурой, свойства которой сохраняют параметры компоновки и содержание для определенной прямоугольной области экрана. В графическом интерфейсе пользователя объект `View` является точкой взаимодействия пользователя и программы.

Класс `ViewGroup` представляет собой контейнер, который служит ядром для подклассов, называемых *компоновками* (layouts). Эти классы формируют расположение элементов пользовательского интерфейса на форме и содержат дочерние элементы `View` или `ViewGroup`.

При разработке пользовательского интерфейса для Android необходимо определить компоновку для каждого `Activity` в виде дерева, используя иерархии узлов `View` и `ViewGroup`, например так, как показано на рис. 5.2. Это дерево иерархии может быть и простым, и сложным, имеющим множество дочерних узлов и большую глубину, — в зависимости от требований, предъявляемых к графическому интерфейсу создаваемого приложения.

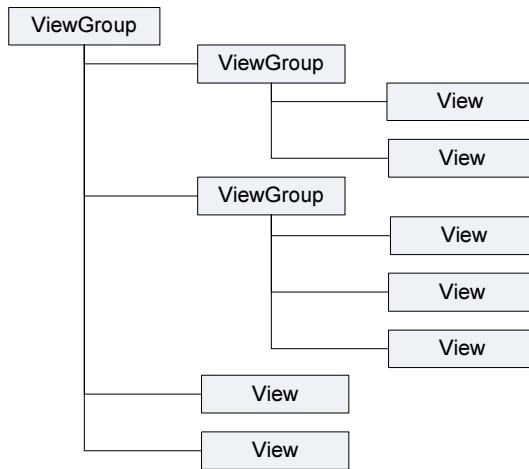


Рис. 5.2. Пример дерева узлов `View` и `ViewGroup` для окна приложения Android

При запуске программы система Android получает ссылку на корневой узел дерева компоновки и использует ее для прорисовки графического интерфейса на экране мобильного устройства. Система также анализирует элементы дерева от вершины дерева иерархии, прорисовывая дочерние объекты `View` и `ViewGroup` и добавляя их родительским элементам. Для этого в методе `onCreate()` необходимо вызвать метод `setContentView()`, передав ему в качестве параметра ссылку на ресурс компоновки в следующем виде:

```
R.layout.layout_file_name
```

Например, если компоновка находится в файле `main.xml`, ее загрузка в методе `onCreate()` происходит так, как представлено в листинге 5.1.

Листинг 5.1. Инициализация компоновки в программном коде

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

Прорисовка элементов дерева компоновки всегда начинается с корневого узла. Затем последовательно прорисовываются дочерние объекты дерева компоновки. Это означает, что родители будут прорисовываться раньше, чем их дочерние объекты, — т. е. по окончании процесса прорисовки элементов на экране родители будут находиться на заднем плане по отношению к дочерним узлам.

Создание компоновки

Компоновку можно создавать двумя способами:

- ❑ объявить элементы пользовательского интерфейса в XML-файле. Android обеспечивает прямой XML-словарь, который соответствует классам `View` и `ViewGroup`;
- ❑ создать компоновку для окна в коде программы во время выполнения — инициализировать объекты `Layout` и дочерние объекты `View`, `ViewGroup` и управлять их свойствами программно.

При разработке пользовательского интерфейса можно использовать каждый из этих методов в отдельности или оба сразу для объявления и управления пользовательским интерфейсом в приложении. Например, можно объявить заданную по умолчанию компоновку окна вашего приложения в XML-файле, включая экранные элементы, которые появятся в них, и их свойства, а затем добавить код в приложение, который во время выполнения изменит состояние объектов на экране, включая объявленные в XML-файле.

XML-файл компоновки

Самый общий способ определять компоновку и создавать иерархию элементов интерфейса — в XML-файле компоновки. XML предлагает удобную структуру для компоновки, похожую на HTML-компоновку веб-страницы.

Преимущество объявления пользовательского интерфейса в XML-файле состоит в том, что это дает возможность отделить дизайн приложения от программного кода, который управляет поведением приложения. Ваше описание пользовательского интерфейса является внешним по отношению к программному коду, что означает, что вы можете изменять пользовательский интерфейс в файле компоновки без необходимости изменения вашего программного кода.

Используя XML-словарь Android, можно быстро проектировать пользовательский интерфейс компоновки и экранные элементы, которые он содержит, тем же самым способом, которым вы создаете веб-страницы в HTML, — с рядом вложенных элементов.

Каждый файл компоновки должен содержать только один корневой элемент, который должен быть объектом `View` или `ViewGroup`. Как только вы определили корневой элемент, вы можете добавить дополнительные объекты компоновки или виджеты как дочерние элементы, чтобы постепенно формировать иерархию элементов, которую определяет создаваемая компоновка.

Самый простой способ объяснить эту концепцию состоит в том, чтобы показать образец. Например, вот этот XML-файл компоновки приложения "Hello, Android!" из главы 3 (листинг 5.2).

Листинг 5.2. Пример файла компоновки `main.xml`

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

    <TextView
        android:id="@+id/TextView01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello">

    </TextView>
</LinearLayout>
```

Общая структура XML-файла компоновки проста: это дерево XML-элементов, где каждый узел представляет имя класса `View` (в нашем примере только один элемент `View` — `TextView`). Вы можете использовать имя любого класса, производного от класса `View`, как элемент в XML-компоновках, включая ваши собственные классы. Эта структура позволяет быстро создавать пользовательский интерфейс, используя более простую структуру и синтаксис, чем при создании компоновки в программном коде. Например, в упомянутой ранее XML-компоновке есть корневой элемент `<LinearLayout>` и только один дочерний элемент `View.TextView` (текстовое поле), которые имеют следующие атрибуты:

- ❑ `xmlns:android` — декларация пространства имен XML, которая сообщает среде Android, что вы ссылаетесь на общие атрибуты, определенные в пространстве имен Android. В каждом файле компоновки у корневого элемента должен быть этот атрибут со значением `"http://schemas.android.com/apk/res/android"`;
- ❑ `android:layout_width` — атрибут определяет, сколько из доступной ширины экрана должен использовать этот объект `View` (или `ViewGroup`). В нашем случае он — единственный объект, таким образом, можно растянуть его на весь экран, которому в данном случае соответствует значение `fill_parent`;
- ❑ `android:layout_height` — аналогичен атрибуту `android:layout_width` за исключением того, что он ссылается на доступную высоту экрана.

Элементы управления в Android могут иметь множество различных атрибутов, которые определяют внешний вид и поведение данного элемента и отличаются в зависимости от

типа элемента. Мы их будем постепенно изучать при рассмотрении элементов управления в следующих главах.

Создание компоновки в Layout Editor

ADT-плагин для Eclipse предлагает удобный инструмент — визуальный редактор компоновки Layout Editor (рис. 5.3), который применяется для создания и предварительного просмотра создаваемых файлов компоновки, которые находятся в каталоге `res/layout/` проекта.

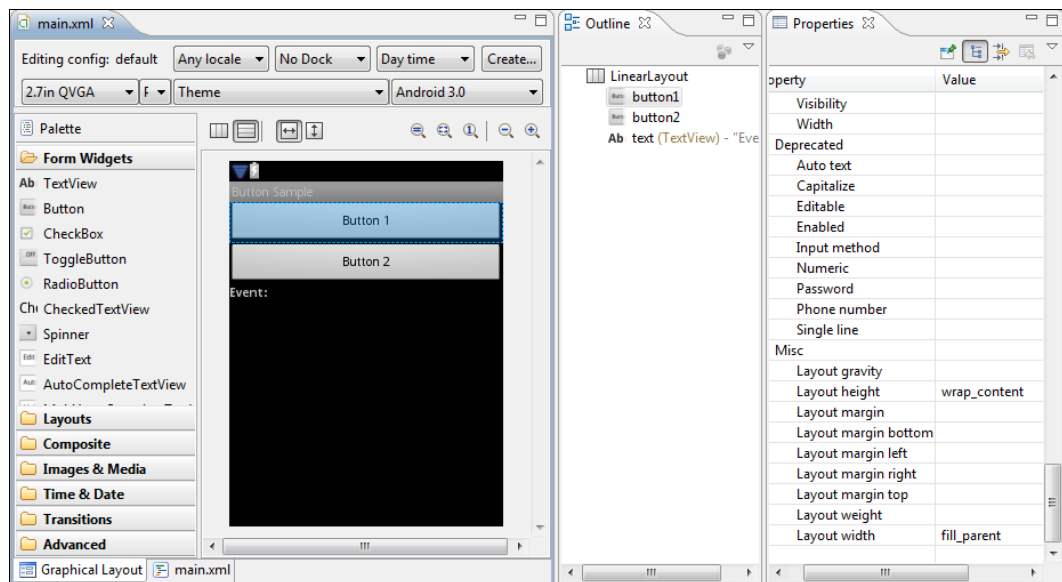


Рис. 5.3. Редактор компоновки Layout Editor

Например, можно создавать XML-компоновки для различных ориентаций экрана мобильного устройства (portrait, landscape), размеров экрана и языков интерфейса. Дополнительно, объявление компоновки в XML-файле облегчает визуализацию структуры вашего пользовательского интерфейса, что упрощает отладку приложения.

На вкладке **Outline** отображается компоновка в виде дерева. Каждый элемент в XML является объектом `View` или `ViewGroup` (или его потомком). Объекты `View` — листья дерева, объекты `ViewGroup` — ветви. Вы можете также создавать объекты `View` и `ViewGroup` в Java-коде, используя метод `addView(View)`, чтобы динамически вставлять новые объекты `View` и `ViewGroup` в существующую компоновку.

Типы компоновок

Используя различные виды `ViewGroup`, можно структурировать дочерние объекты `View` и `ViewGroup` многими способами в зависимости от требований к графическому интерфейсу приложения.

Для создания окон существует несколько стандартных типов компоновок, которые вы можете использовать в создаваемых приложениях:

- ☐ `FrameLayout`;
- ☐ `LinearLayout`;
- ☐ `TableLayout`;
- ☐ `RelativeLayout`.

Все эти компоновки являются подклассами `ViewGroup` (рис. 5.4) и наследуют свойства, определенные в классе `View`.

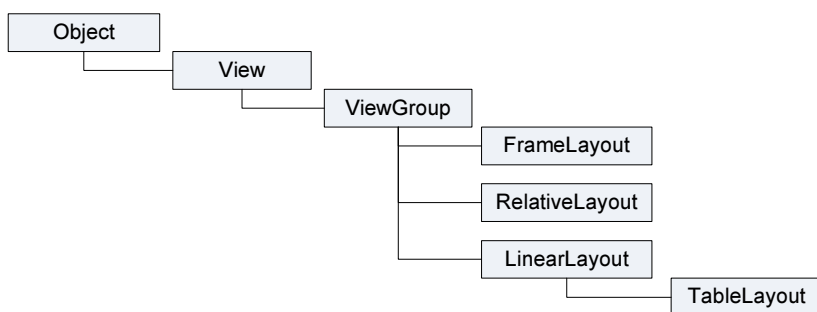


Рис. 5.4. Иерархия классов компоновок

Каждый из этих типов компоновки предлагает уникальный набор параметров, которые используются, чтобы определить позиции дочерних элементов `View` и структуру компоновки на экране. В зависимости от требований, предъявляемых к пользовательскому интерфейсу, выбирается наиболее подходящий тип компоновки. Далее мы рассмотрим все варианты компоновок и их использование.

Существует еще один тип компоновки — `AbsoluteLayout`. В этой компоновке дочерние элементы имеют жесткую пиксельную привязку координат. Компоновка `AbsoluteLayout` уже достаточно давно является устаревшей и на ее основе создавать пользовательский интерфейс не рекомендуется. В принципе, это не удивительно — при таком большом количестве вариантов разрешения экранов мобильных устройств создать удобную компоновку практически невозможно.

FrameLayout

`FrameLayout` является самым простым типом компоновки. Это в основном пустое пространство на экране, которое можно позже заполнить только единственным дочерним объектом `View` или `ViewGroup`. Все дочерние элементы `FrameLayout` прикрепляются к верхнему левому углу экрана.

Чтобы поработать с компоновкой "вживую", создайте в Eclipse новый проект и в диалоговом окне **New Android Project** введите следующие значения:

- ☐ **Project name** — `FrameLayoutApp`;
- ☐ **Application name** — `FrameLayout Sample`;

- ❑ **Package name** — `com.samples.framelayout`;
- ❑ **Create Activity** — `FrameLayoutActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch05_FrameLayout`.

Откройте файл `main.xml` и создайте компоновку, как в листинге 5.4. Компоновку можно создавать, используя `Layout Editor` или вручную, написав XML-код, приведенный в листинге 5.3.

Листинг 5.3. Файл компоновки `main.xml` для `FrameLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/FrameLayout01"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent">

    <Button
        android:text="@+id/Button1"
        android:id="@+id/Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</FrameLayout>
```

Внешний вид экрана с компоновкой `FrameLayout` должен получиться таким, как на рис. 5.5. (Можете не запускать проект в эмуляторе, посмотрите внешний вид окна в `Layout Editor`.)

В компоновке `FrameLayout` нельзя определить различное местоположение для дочернего объекта `View`. Последующие дочерние объекты `View` будут просто рисоваться поверх предыдущих, частично или полностью затеняя их, если находящийся сверху объект непрозрачен, поэтому единственный дочерний элемент для `FrameLayout` обычно растянут до размеров родительского контейнера и имеет атрибуты `layout_width="fill_parent"` и `layout_height="fill_parent"`.

Добавьте в файл разметки дополнительный элемент `Button`, как показано в листинге 5.4.

Листинг 5.4. Измененный файл компоновки `main.xml` для `FrameLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/FrameLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button
        android:text="Button1"
```

```
        android:id="@+id/Button1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>
<Button android:text="Button2"
        android:id="@+id/Button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</FrameLayout>
```

Вторая кнопка будет прорисована поверх первой и будет затенять ее, как на рис. 5.6.

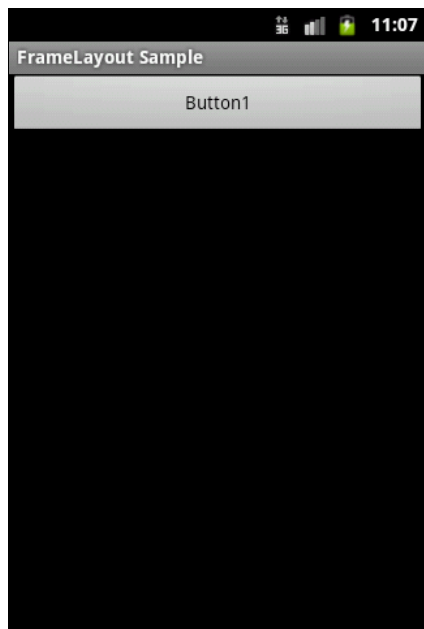


Рис. 5.5. Компоновка `FrameLayout` с одним дочерним элементом — кнопкой

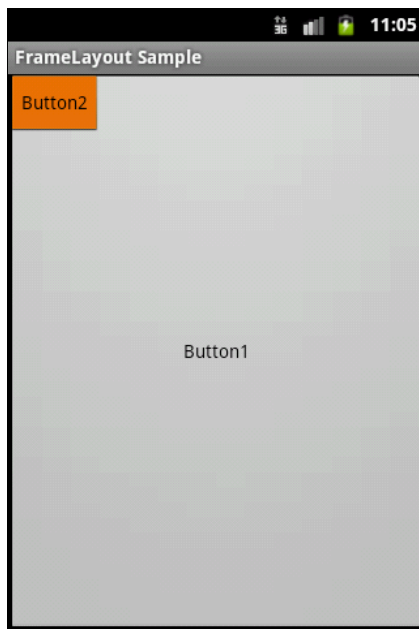


Рис. 5.6. Компоновка `FrameLayout` с двумя дочерними элементами

Компоновка `FrameLayout` применяется довольно редко, т. к. не позволяет создавать сложные окна с множеством элементов. Эту компоновку обычно используют для создания оверлеев. Например, если у вас в окне выводится изображение, занимающее весь экран (это может быть карта, загруженная с сервиса Google Map, или картинка с видеокамеры), можно сверху на изображении расположить элементы управления (в дочернем контейнере, если их несколько), скажем, кнопки для управления камерой и рамку видоискателя, а также выводить индикацию времени съемки и другую полезную информацию.

LinearLayout

Компоновка `LinearLayout` выравнивает все дочерние объекты `View` в одном направлении — вертикально или горизонтально, в зависимости от того, как определен атрибут ориентации `android:orientation`:

```
android:orientation="horizontal"
```

или

```
android:orientation="vertical"
```

Все дочерние элементы помещаются в стек один за другим, так что вертикальный список объектов `View` будет иметь только один дочерний элемент в строке независимо от того, насколько широким он является. Горизонтальное расположение списка будет размещать элементы в одну строку с высотой, равной высоте самого высокого дочернего элемента списка.

Поменяйте код в файле `res/layout/main.xml`: создайте `LinearLayout` с тремя дочерними кнопками, как показано в листинге 5.5.

Листинг 5.5. Файл компоновки `main.xml` для `LinearLayout`

```
<?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="horizontal">

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button1"/>

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button2"/>

    <Button
        android:id="@+id/button3"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Button3"/>

</LinearLayout>
```

ПРИМЕЧАНИЕ

Полный код примера находится в каталоге `Ch05_LinearLayout`.

Обратите внимание, что у первых двух кнопок атрибуту `android:layout_width` присвоено значение `wrap_content`, а у третьей кнопки — `fill_parent`, т. е. последняя кнопка заполнит оставшееся свободное пространство в компоновке.

В результате получится линейное горизонтальное размещение дочерних элементов. Если изменить в корневом элементе значение атрибута `android:layout_height`:

```
android:orientation="vertical",
```

элементы в контейнере расположатся вертикально. Внешний вид экрана для компоновки `LinearLayout` с горизонтальной и вертикальной ориентациями элементов показан на рис. 5.7.

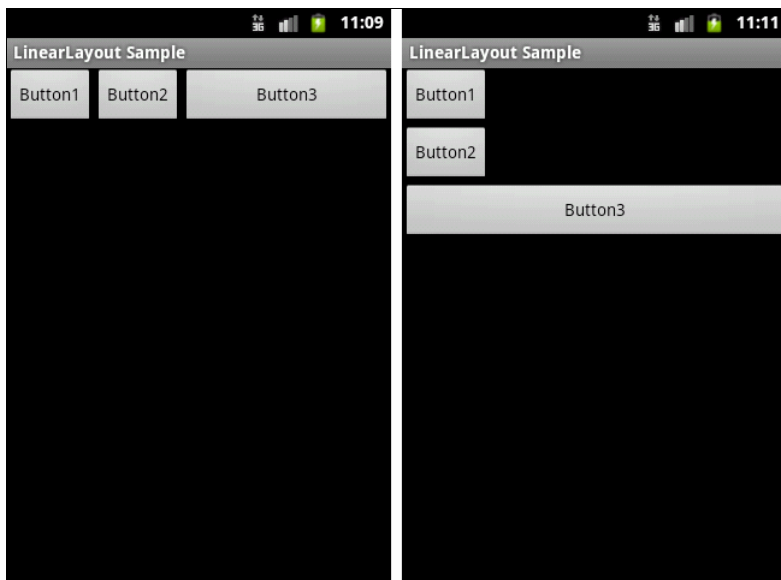


Рис. 5.7. Компоновка `LinearLayout` с горизонтальной и вертикальной ориентациями элементов

Компоновка `LinearLayout` также поддерживает атрибут `android:layout_weight`, который назначает индивидуальный вес для дочернего элемента. Данный атрибут определяет "важность" объекта `View` и позволяет этому элементу расширяться, чтобы заполнить любое оставшееся пространство в родительском объекте `View`. Заданный по умолчанию вес является нулевым.

Например, если есть три текстовых поля, и двум из них объявлен вес со значением 1, в то время как другому не дается никакого веса (0), третье текстовое поле без веса не будет расширяться и займет область, определяемую размером текста, отображаемого этим полем. Другие два расширятся одинаково, чтобы заполнить остаток пространства, не занятого третьим полем. Если третьему полю присвоить вес 2 (вместо 0), это поле будет объявлено как "более важное", чем два других, так что третье поле получит 50% общего пространства, в то время как первые два получают по 25% общего пространства.

Далее в файле `res/layout/main.xml` создайте `LinearLayout` с тремя дочерними элементами `Button`, как показано в листинге 5.6.

Листинг 5.6. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```



```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="B1"
    android:layout_weight="0"/>
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="B2"
    android:layout_weight="1"/>
<Button
    android:id="@+id/button3"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="B3"
    android:layout_weight="2"/>
</LinearLayout>
```

ПРИМЕЧАНИЕ

Полный код примера находится в каталоге Ch05_LinearLayoutWeight.

Внешний вид экрана и влияние атрибута `android:layout_weight` показаны на рис. 5.8.

Обратите внимание, как различные XML-атрибуты определяют поведение элемента. Попробуйте поэкспериментировать с различными значениями `layout_weight` для дочерних элементов, чтобы увидеть, как будет распределяться доступное пространство для элементов.

Компоновки могут быть и вложенными. При проектировании окон с многочисленными элементами управления для заданного расположения элементов часто задаются вложенные компоновки, которые являются контейнерами для элементов управления. Например, для корневой `LinearLayout` с атрибутом `orientation="vertical"` мы можем задать простой дочерний элемент `Button` и еще два контейнера `LinearLayout` с атрибутом `orientation="horizontal"`, которые, в свою очередь, содержат дочерние элементы управления, как показано на рис. 5.9.

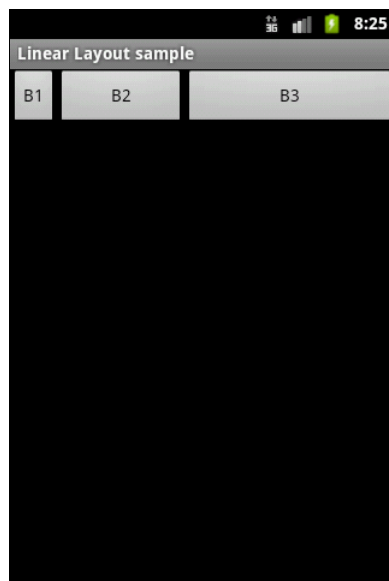


Рис. 5.8. Отображение элементов с различными значениями `android:layout_weight`

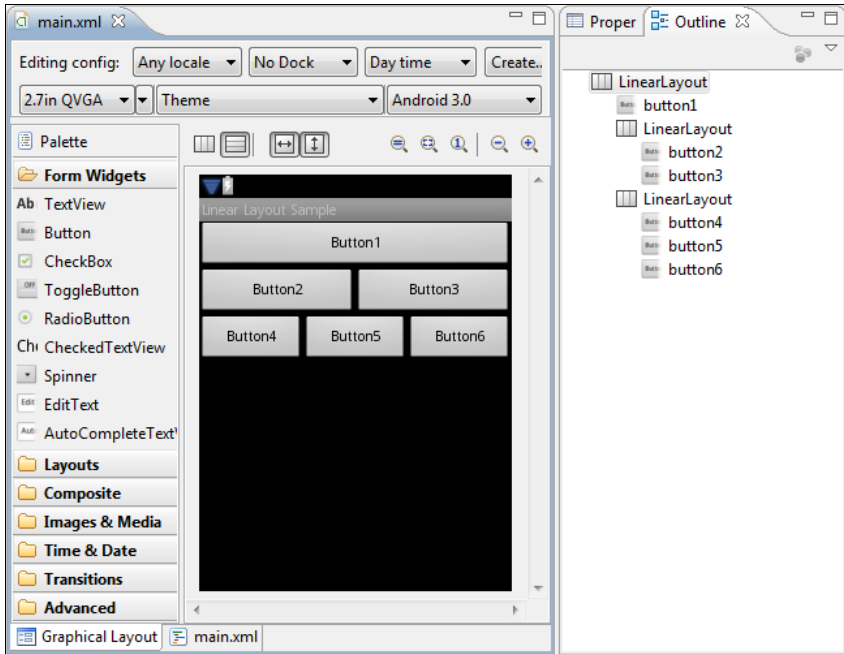


Рис. 5.9. Компоновка LinearLayout с двумя вложенными компоновками

Код файла main.xml в результате должен получиться таким, как в листинге 5.7.

Листинг 5.7. Компоновка LinearLayout с вложенными компоновками

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

    <Button
        android:text="Button1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    <LinearLayout
        android:orientation="horizontal"
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
        <Button
            android:text="Button2"
            android:layout_width="wrap_content"
            android:layout_weight="1"
            android:layout_height="wrap_content"/>
        <Button android:text="Button3"
            android:id="@+id/button3">
```

```
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>

<LinearLayout
    android:orientation="horizontal"
    android:layout_height="wrap_content"
    android:layout_width="match_parent">
    <Button
        android:text="Button4"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button
        android:text="Button5"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>
    <Button
        android:text="Button6"
        android:layout_weight="1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"/>
</LinearLayout>
</LinearLayout>
```

ПРИМЕЧАНИЕ

Полный код примера находится в каталоге Ch05_LinearLayoutTree.

Такое применение вложенных компоновок позволяет строить гибкие и легко перенастраиваемые окна и является самым распространенным способом при создании пользовательского интерфейса для Android-приложений.

TableLayout

Компоновка `TableLayout` позиционирует свои дочерние элементы в строки и столбцы. `TableLayout` не отображает линии обрамления для их строк, столбцов или ячеек. Кроме того, `TableLayout` может иметь строки с разным количеством ячеек. При формировании компоновки таблицы некоторые ячейки при необходимости можно оставлять пустыми.

При создании компоновки для строк используются объекты `TableRow`, которые являются дочерними классами `TableLayout` (каждый `TableRow` определяет единственную строку в таблице). Строка может не иметь ячеек или иметь одну и более ячеек, которые являются контейнерами для других объектов `View` или `ViewGroup`. Ячейка может также быть объектом `ViewGroup` (например, допускается вложить другой `TableLayout` или `LinearLayout` как ячейку).

Для примера с использованием компоновки `TableLayout` можно создать окно, похожее на наборную панель телефона с 12 кнопками. В окне **Layout Editor** создайте `TableLayout` с четырьмя дочерними `TableRow` и двенадцатью кнопками, по три кнопки в каждой строке.

Для каждого элемента `TableRow` на вкладке **Properties** задайте свойства:

- ☐ **Layout height** — `wrap_content`;
- ☐ **Layout width** — `fill_parent`;
- ☐ **Gravity** — `center`.

Свойство `gravity` задает выравнивание дочерних элементов в контейнере, в данном случае — по центру.

Для каждой кнопки на вкладке **Properties** задайте свойства:

- ☐ **Layout height** — `wrap_content`;
- ☐ **Layout width** — `20pt`.

Надписи на кнопках сделайте так, как на телефонной клавиатуре (1, 2, 3, 4, 5, 6, 7, 8, 9, *, 0, #). В результате внешний вид экрана должен получиться в виде телефонной клавиатуры, как на рис. 5.10.

Файл компоновки должен получиться таким, как в листинге 5.8.

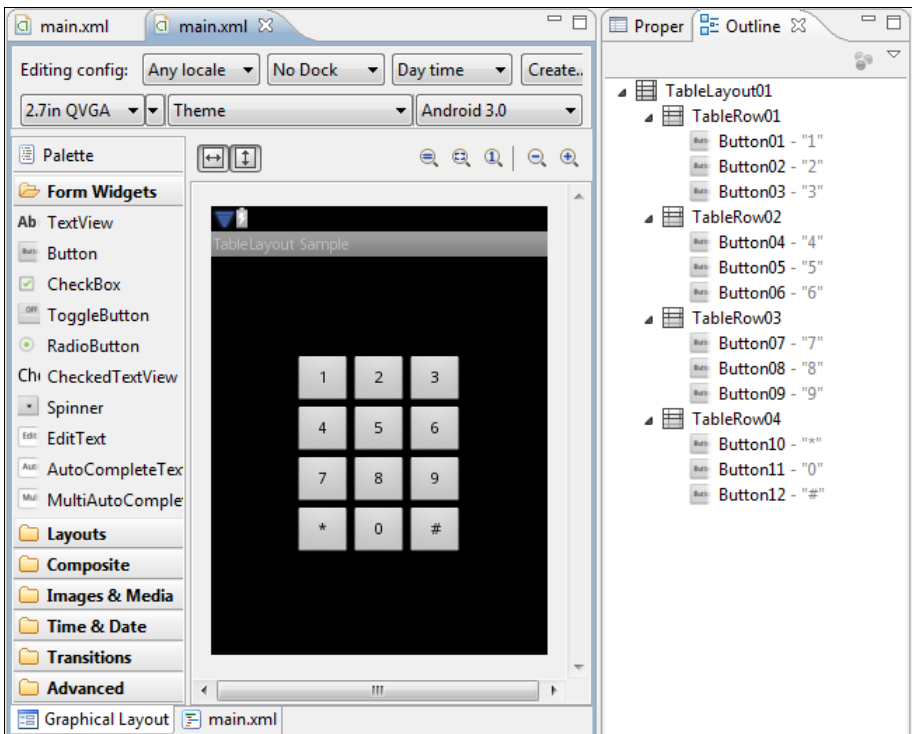


Рис. 5.10. Дерево элементов для компоновки `TableLayout`

Листинг 5.8. Файл компоновки main.xml для TableLayout

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TableLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TableRow
        android:id="@+id/TableRow01"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center">
        <Button
            android:id="@+id/Button01"
            android:layout_height="wrap_content"
            android:text="1"
            android:layout_width="20pt"/>
        <Button
            android:id="@+id/Button02"
            android:layout_height="wrap_content"
            android:text="2"
            android:layout_width="20pt"/>
        <Button
            android:id="@+id/Button03"
            android:layout_height="wrap_content"
            android:text="3"
            android:layout_width="20pt"/>
    </TableRow>
    <TableRow
        android:id="@+id/TableRow02"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:gravity="center">
        <Button
            android:id="@+id/Button04"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="4"/>
        <Button
            android:id="@+id/Button05"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="5"/>
        <Button
            android:id="@+id/Button06"
            android:layout_height="wrap_content"
            android:layout_width="20pt"
            android:text="6"/>
    </TableRow>
```

```

<TableRow
    android:id="@+id/TableRow03"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:gravity="center">
    <Button
        android:id="@+id/Button07"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="7"/>
    <Button
        android:id="@+id/Button08"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="8"/>
    <Button
        android:id="@+id/Button09"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="9"/>
</TableRow>
<TableRow
    android:id="@+id/TableRow04"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:gravity="center">
    <Button
        android:id="@+id/Button10"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="*/>
    <Button
        android:id="@+id/Button11"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="0"/>
    <Button
        android:id="@+id/Button12"
        android:layout_height="wrap_content"
        android:layout_width="20pt"
        android:text="#">
</TableRow>
</TableLayout>

```

ПРИМЕЧАНИЕ

Полный код примера находится в каталоге Ch05_TableLayout.

Компоновка `TableLayout` на практике применяется довольно редко, обычно вместо нее используют сочетание компоновок `LinearLayout`. `TableLayout` удобно использовать, если

расположение элементов представлено в виде "таблицы", как в нашем примере на рис. 5.10.

RelativeLayout

`RelativeLayout` (относительная компоновка) позволяет дочерним объектам определять свою позицию относительно родительского объекта или относительно соседних дочерних элементов (по идентификатору элемента).

В `RelativeLayout` дочерние элементы расположены так, что если первый элемент расположен по центру экрана, другие элементы, выровненные относительно первого элемента, будут выровнены относительно центра экрана. При таком расположении, при объявлении компоновки в XML-файле, элемент, на который будут ссылаться для позиционирования другие объекты, должен быть объявлен раньше, чем другие элементы, которые обращаются к нему по его идентификатору.

Если в программном коде мы не работаем с некоторыми элементами пользовательского интерфейса, создавать идентификаторы для них необязательно, однако определение идентификаторов для объектов важно при создании `RelativeLayout`. В компоновке `RelativeLayout` расположение элемента может определяться относительно другого элемента, на который ссылаются через его уникальный идентификатор:

```
android:layout_toLeftOf="@id/TextView1"
```

Давайте теперь создадим пример окна с корневой компоновкой `RelativeLayout`, в котором будут семь элементов `Button`. В файле компоновки напишите код, как показано в листинге 5.9.

Листинг 5.9. Файл компоновки `main.xml` для `RelativeLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">

    <Button
        android:id="@+id/button_center"
        android:text="Center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_centerInParent="true"/>

    <Button
        android:id="@+id/button_bottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"/>

    <Button
        android:id="@+id/button_top"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>
<Button
    android:id="@+id/button_left"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Left"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"/>
<Button
    android:id="@+id/button_right"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Right"
    android:layout_alignParentRight="true"
    android:layout_centerVertical="true"/>
<Button
    android:id="@+id/button_rel_right"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/button_right"
    android:layout_alignTop="@id/button_right"
    android:text="RelRight"/>
<Button
    android:id="@+id/button_rel_left"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/button_left"
    android:layout_alignTop="@id/button_left"
    android:text="RelLeft"/>

</RelativeLayout>
```

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch05_RelativeLayout.

Кстати, обратите внимание, что атрибуты, которые обращаются к идентификаторам относительных элементов (например, `layout_toLeftOf`), используют синтаксис относительного ресурса `@id/id`. Внешний вид экрана с компоновкой `RelativeLayout` должен получиться, как показано в примере на рис. 5.11.

Тип компоновки `RelativeLayout` применяется довольно редко. Тем более что такое задание расположения элементов зависит от разрешения и ориентации экрана мобильного устройства. Если вы будете использовать `RelativeLayout` в собственных приложениях, всегда проверяйте внешний вид окна для различных разрешений и ориентации экрана, поскольку возможно наложение элементов друг на друга.

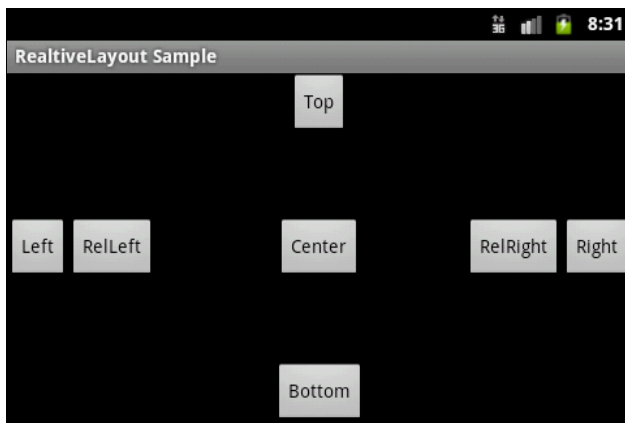


Рис. 5.11. Пример компоновки RelativeLayout

Отладка пользовательского интерфейса с помощью Hierarchy Viewer

В состав Android SDK входит утилита Hierarchy Viewer — полезный инструмент при разработке окон со сложной компоновкой. Он позволяет отлаживать и оптимизировать пользовательский интерфейс приложений. Hierarchy Viewer отображает визуальное представление иерархии элементов создаваемой компоновки и экранную лупу для просмотра пиксельной структуры компоновки экрана мобильного устройства — Pixel Perfect View.

Для запуска Hierarchy Viewer необходимо выполнить следующие действия: подключить мобильное устройство или запустить эмулятор Android, затем ввести в командной строке `hierarchyviewer` (или запустить файл `hierarchyviewer.bat` из каталога `tools/` в Android SDK).

Выберите нужное окно для отладки и нажмите кнопку **Load View Hierarchy** (рис. 5.12). Откроется окно **Layout View**. После этого при необходимости можно загрузить окно **Pixel Perfect View**, нажимая второй значок в левой нижней части окна **Layout View**.

Окно **Layout View** в левой части отображает иерархию компоновки и ее свойства. У окна справа есть три панели:

- ☐ **Tree View** — дерево элементов;
- ☐ **Properties View** — список свойств выбранного элемента;
- ☐ **Wire-frame View** — каркас компоновки.

На рис. 5.13 показано окно **Layout View** с загруженным проектом `RelativeLayoutApp` из листинга 5.9.

Для отображения свойств элемента на панели **Properties View** необходимо выбрать элемент в дереве компоновки на панели **Tree View**. При выборе элемента на панели **Wire-frame View** будут в красном прямоугольнике показаны границы этого элемента. Двойной щелчок на узле дерева компоновки открывает новое окно с рендерингом этого элемента.

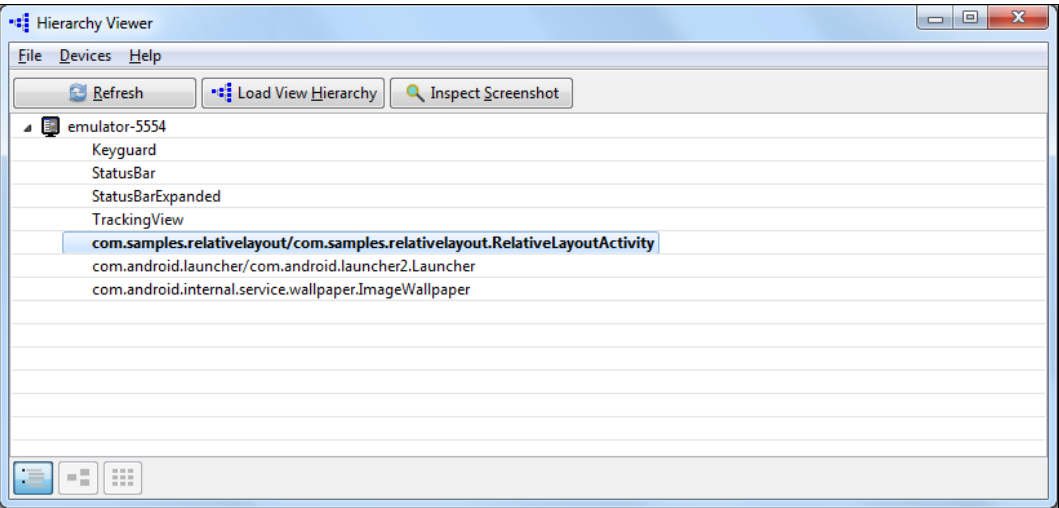


Рис. 5.12. Инструмент Hierarchy Viewer

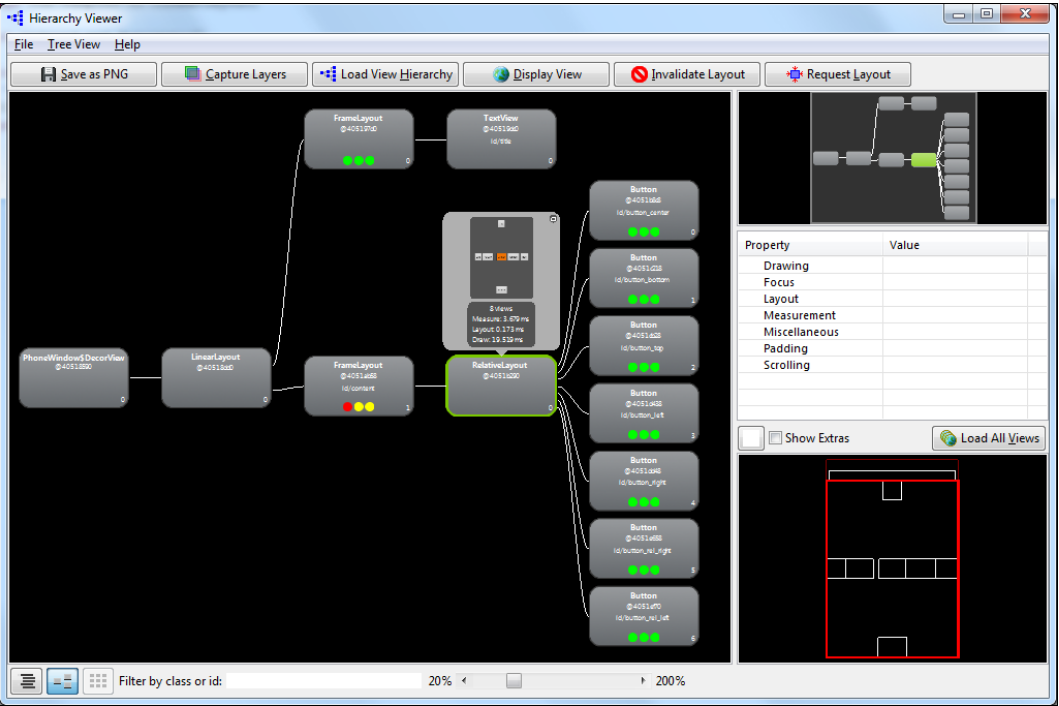


Рис. 5.13. Окно Layout View утилиты Hierarchy Viewer

Окно **Layout View** включает пару других полезных опций отладки создаваемой компоновки — **Invalidate Layout** и **Request Layout**. Эти кнопки на панели инструментов вызывают соответственно методы `invalidate()` и `requestLayout()` для выбранного элемента.

ПРИМЕЧАНИЕ

Если вы меняете выбранный объект, то **Layout View** не будет автоматически его обновлять. Вы должны перезагрузить **Layout View**, нажимая кнопку **Load View Hierarchy**.

Чтобы переключаться между окнами, используются также три кнопки на панели состояния. Если нажать кнопку **Inspect Screenshot** (см. рис. 5.12), откроется окно **Pixel Perfect View**, которое предоставляет экранную лупу для детального просмотра компоновки (рис. 5.14).

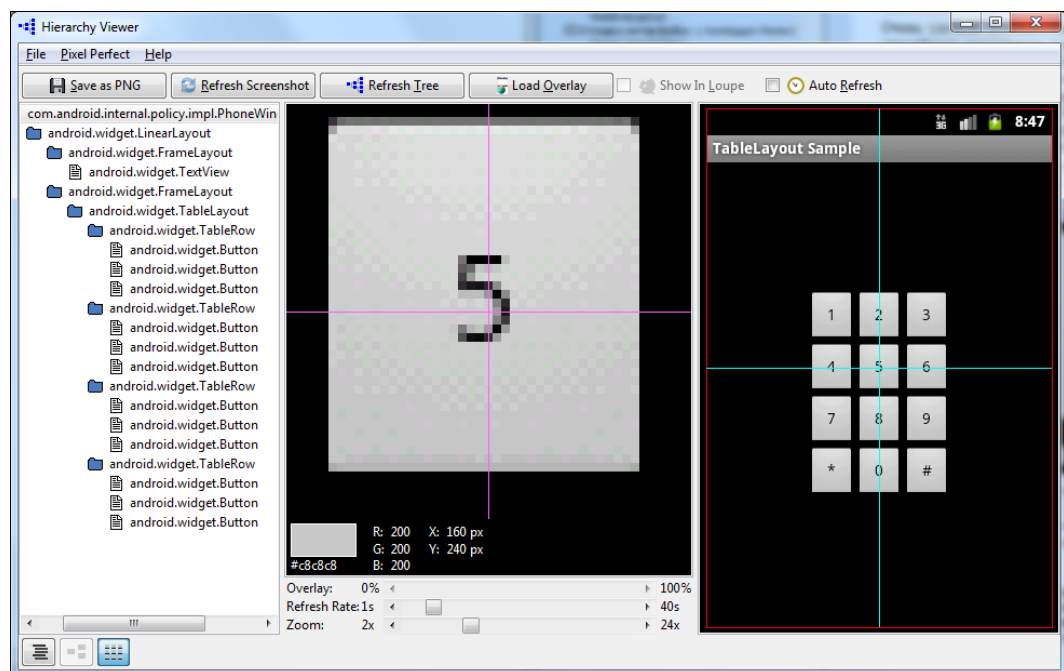


Рис. 5.14. Окно **Pixel Perfect View** утилиты **Hierarchy Viewer**

У окна **Pixel Perfect View** есть три панели:

- ☐ **Explorer View** — показывает иерархию элементов;
- ☐ **Normal View** — нормальное представление окна устройства;
- ☐ **Loupe View** — увеличенное представление пиксельной архитектуры экрана устройства.

Панель схемы компоновки использует множественные прямоугольники для указания границ, отступов (padding) и полей (margin). Фиолетовый или зеленый прямоугольник указывает границы — высоту и ширину — элемента. Внутренний белый или черный прямоугольник указывает границы информационного наполнения, когда в элементе установлены отступы (padding). Черный или белый прямоугольник вне фиолетового или зеленого прямоугольника указывает любые имеющиеся поля (margin).

Очень удобной особенностью проектирования пользовательского интерфейса является способность накладывать дополнительное изображение в режимах **Normal View** и **Loupe View**. Например, у вас могло бы быть изображение макета компоновки окна

приложения, который вы хотели бы применить в разрабатываемом пользовательском интерфейсе. Нажав кнопку **Load Overlay**, можно загрузить изображение макета компоновки, созданной в другом проекте или другим разработчиком. Выбранное изображение прикрепится в углу левой нижней части экрана. Вы можете корректировать непрозрачность оверлея и подстраивать разрабатываемую компоновку для соответствия макету.

ПРИМЕЧАНИЕ

Изображения экрана устройства на панелях **Normal View** и **Loupe View** обновляются через равномерные промежутки времени (5 секунд по умолчанию), но панель **Explorer View** автоматических обновлений не производит. Необходимо вручную обновлять содержимое **Explorer View**, нажимая кнопку **Load View Hierarchy**.

Резюме

В этой главе мы рассмотрели общие принципы компоновки элементов управления в окне Android-приложения. Правильный выбор типа компоновки и ее использование при разработке дизайна окна является важным этапом создания приложения с графическим интерфейсом.

В следующей главе мы переходим к рассмотрению виджетов для отображения и редактирования текста, а также виджетов для отображения графики.



ГЛАВА 6

Базовые виджеты

Виджет — это объект `View`, который служит интерфейсом для взаимодействия с пользователем. Говоря простым языком, виджеты — это элементы управления. Android обеспечивает набор готовых виджетов, таких как кнопки, переключатели и текстовые поля, с помощью которых можно быстро сформировать пользовательский интерфейс приложения.

В этой главе мы изучим простейшие виджеты для отображения и редактирования текста, отображения графики, а также контейнерные виджеты — полосы прокрутки. Эти виджеты являются базовыми для большинства остальных элементов управления, которые мы будем рассматривать в главах этой части.

Текстовые поля

Текстовые поля в Android представлены двумя классами:

- ❑ `TextView`;
- ❑ `EditText`.

Виджет `TextView` предназначен для отображения текста без возможности редактирования его пользователем. Если необходимо редактирование текста, используется виджет `EditText`.

Классы `TextView` и `EditText` имеют множество атрибутов и методов, наследуемых от класса `View`, который был рассмотрен в предыдущей главе. Иерархия классов текстовых полей представлена на рис. 6.1.

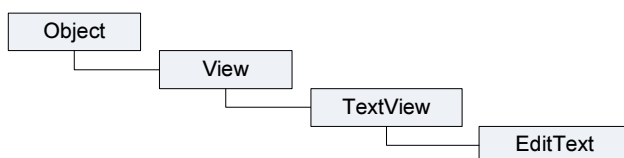


Рис. 6.1. Иерархия классов текстовых полей

TextView

Виджет `TextView` — самый простой и в то же время один из самых используемых в приложениях виджетов. `TextView` служит для представления пользователю описательного текста без возможности его редактирования.

Кроме того, элемент `TextView` используется как элемент для отображения текстовых данных в контейнерных виджетах для отображения списков. От класса `TextView` наследуется множество других виджетов: кнопки, флажки и переключатели — элементы управления, на которых может быть отображен текст. В примерах, приведенных далее в этой части, мы будем активно применять этот элемент для отображения состояния элементов при обработке событий.

Виджет `TextView`, так же как и объект `View`, от которого он наследуется, поддерживает собственное разнообразие XML-атрибутов. Некоторые атрибуты являются определенными только в объекте `TextView`, но эти атрибуты могут также наследоваться любыми объектами, которые расширяют этот класс.

Свойства для элемента `TextView` можно задавать как в файле компоновки, так и в программном коде. Например, для отображения текста в `TextView` в файле компоновки используется атрибут `android:text`, а в программном коде вызывается метод `setText()` этого класса.

В целом, XML-словарь элементов пользовательского интерфейса близок к структуре классов и методов этих элементов, где имя элемента соответствует имени класса, а атрибуты элемента — методам этого класса. Фактически, соответствие является часто настолько точным, что легко предположить без обращения к документации Android, какой XML-атрибут передает метод класса или, наоборот, какой метод класса соответствует конкретному XML-элементу. В табл. 6.1 приведены для примера несколько свойств виджета `TextView`, чтобы вы могли оценить точность соответствия XML-атрибутов и методов класса `TextView`.

Таблица 6.1. Соответствие XML-атрибутов и методов в классах представлений

Имя XML-атрибута	Соответствующий метод в классе Java
<code>android:text</code>	<code>setText()</code>
<code>android:textColor</code>	<code>setTextColor()</code>
<code>android:textSize</code>	<code>setTextSize()</code>
<code>android:textColorHighlight</code>	<code>setHighlightColor()</code>

Однако обратите внимание на последнюю строку в табл. 6.1: не весь XML-словарь идентичен структуре классов.

Некоторые атрибуты виджета `TextView` являются общими по отношению ко всем объектам `View`, потому что они унаследованы от корневого класса `View`. Такими атрибутами являются, например, `id`, `layout_width`, `layout_height`, с которыми вы уже познакомились в главе 4.

Если в программном коде мы работаем с данным элементом пользовательского интерфейса, в файле компоновки обязательно определяют идентификатор, например:

```
android:id="@+id/text1"
```

где символ @ в начале строки указывает, что синтаксический анализатор XML должен проанализировать и развернуть остальную часть строки идентификатора и определить это выражение как ресурс идентификатора. Символ + означает, что это новое имя ресурса, которое должно быть создано и добавлено к нашим ресурсам в файл R.java, который среда Android автоматически генерирует для проекта, как было показано в главе 3 (листинг 3.3).

Требование к уникальности идентификаторов не распространяется на все дерево элементов, но они должны быть уникальны в пределах части дерева (которая нередко может быть и полным деревом, так что лучше создавать полностью уникальные идентификаторы, если это возможно).

Если планируется создание приложения с многоязыковой поддержкой пользовательского интерфейса, вместо непосредственного задания текста в XML-компоновке или в коде программы необходимо создать ссылку на текстовый XML-ресурс:

```
android:text="@string/text_hello",
```

где `text_hello` — имя ресурса.

В коде программы ссылка на XML-ресурс задается методом `setText()`, который принимает ссылку на идентификатор ресурса, определенного в файле R.java (автоматически сгенерированным средой разработки), например:

```
TextView text = (TextView)findViewById(R.id.text);  
// Задаем текст из ресурсов  
text.setText(R.string.text_hello);
```

У элемента `TextView` есть многочисленные методы и XML-атрибуты для работы с текстом. Например, основные XML-атрибуты, отображающие свойства элемента `TextView`:

- ❑ `android:textSize`;
- ❑ `android:textStyle`;
- ❑ `android:textColor`.

Атрибут `android:textSize` задает размер текста. При установке размера текста используются несколько единиц измерения:

- ❑ `px (pixels)` — пикселы;
- ❑ `dp (density-independent pixels)` — независимые от плотности пикселы. Это абстрактная единица измерения, основанная на физической плотности экрана;
- ❑ `sp (scale-independent pixels)` — независимые от масштабирования пикселы;
- ❑ `in (inches)` — дюймы, базируются на физических размерах экрана;
- ❑ `pt (points)` — 1/72 дюйма, базируются на физических размерах экрана;
- ❑ `mm (millimeters)` — миллиметры, также базируются на физических размерах экрана.

Обычно при установке размера текста используются единицы измерения `sp`, которые наиболее корректно отображают шрифты, например:

```
android:textSize="48sp";
```

Атрибут `android:textStyle` представляет стиль текста (нормальный, полужирный, наклонный). Для задания стиля текста используются только следующие константы:

- ☐ `normal`;
- ☐ `bold`;
- ☐ `italic`.

Вот пример установки стиля через атрибуты в файле компоновки:

```
android:textStyle="bold";
```

Атрибут `android:textColor` задает цвет текста. Для задания цвета используются четыре формата в шестнадцатеричной кодировке:

- ☐ `#RGB`;
- ☐ `#ARGB`;
- ☐ `#RRGGBB`;
- ☐ `#AARRGGBB`,

где R, G, B — соответствующий цвет, A — альфа-канал (alpha-channel), который определяет прозрачность. Значение A, установленное в 0, означает прозрачность 100%. Значение по умолчанию без указания значения alpha равно 1, т. е. непрозрачно.

Для всех перечисленных ранее атрибутов в классе `TextView` есть соответствующие методы для чтения или задания соответствующих свойств.

Сейчас мы создадим простое приложение с виджетом `TextView`, в котором рассмотрим различные способы задания его свойств — через атрибуты в файле компоновки и программно, в коде класса, реализующего окно приложения. Для этого создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `TextView`;
- ☐ **Application name** — `TextView Sample`;
- ☐ **Package name** — `com.samples.ui.textview`;
- ☐ **Create Activity** — `TextViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch06_TextView`.

Откройте файл компоновки `main.xml` и создайте компоновку `LinearLayout` и в ней четыре элемента `TextView` с идентификаторами `text1`, `text2`, `text3`, `text4`. Для `text1` задайте текст непосредственно в XML-коде:

```
android:text="Hello, Android!"
```

Для элемента `text2` текст задайте через ссылку на строковый ресурс. Можно также задать различный размер, цвет и стиль форматирования текста для элементов `text3` и `text4`. Полный код файла компоновки показан в листинге 6.1.

Листинг 6.1. Файл компоновки `main.xml`

```
<?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/text1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text from res/layout/main.xml"/>

<TextView
    android:id="@+id/text2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text_hello"
    android:textStyle="bold"/>

<TextView
    android:id="@+id/text3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:textStyle="bold"
    android:textColor="#ABABAB"/>

<TextView
    android:id="@+id/text4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="36sp"
    android:textStyle="italic"/>

</LinearLayout>
```

В файле ресурсов strings.xml добавьте после ресурса app_name новый строковый ресурс "Hello, Android!" (листинг 6.2).

Листинг 6.2. Файл ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">TextView Sample</string>
    <string name="text_hello">Text from res/values/string.xml</string>
</resources>
```

В классе `TextViewActivity` инициализируйте `TextView`-объекты `text3`, `text4` и методом `setText()` задайте для них текст. Полный код класса показан в листинге 6.3.

Листинг 6.3. Файл класса окна приложения `TextViewActivity.java`

```
package com.samples.ui.textview;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
```

```

public class TextViewActivity extends Activity {

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

        // Получаем объекты TextView из ресурсов
        final TextView text3 = (TextView) findViewById(R.id.text3);
        final TextView text4 = (TextView) findViewById(R.id.text4);

        // Устанавливаем текст
        text3.setText("Text from Activity");

        // Загружаем строку текста из ресурсов
        text4.setText(R.string.text_hello);
    }
}

```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Результат должен получиться такой, как на рис. 6.2. Для первого поля текст задается прямо в файле компоновки, для второго — в файле компоновки из ресурса, для третьего — задается в коде, для четвертого поля — читается в коде из файла ресурсов.

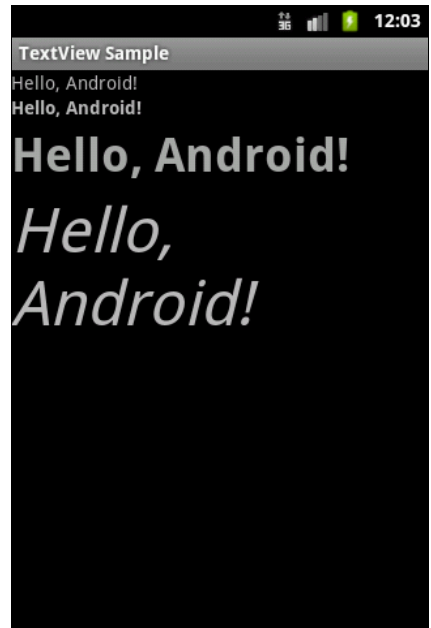


Рис. 6.2. Приложение с виджетами TextView

EditText

Элемент `EditText` — это текстовое поле для пользовательского ввода. `EditText` представляет собой тонкую оболочку над классом `TextView`, которая позволяет редактировать вводимый пользователем текст.

Основной метод класса `EditText` — `getText()`, который возвращает текст, содержащийся в окне элемента `EditText`. Возвращаемое значение имеет тип `Editable`. Класс `Editable` представляет собой интерфейс для текста, информационное наполнение которого мо-

жет изменяться (в противоположность типу `String`, который является неизменяемым, при его изменении просто создается новый экземпляр `String`).

В классе `EditText` есть метод `setHint()` для отображения подсказки. С помощью этого метода можно задать текст подсказки, который увидит пользователь в этом элементе, например "Enter some text...".

В классе также определены методы для выделения текста:

- ❑ `selectAll()` — выделяет весь текст в окне;
- ❑ `setSelection(int start, int stop)` — выделяет участок текста с позиции `start` до позиции `stop`;
- ❑ `setSelection(int index)` — перемещает курсор на позицию `index`.

Большинство методов для работы с текстом и его форматированием унаследованы от базового класса `TextView`. Чаще всего используются методы `setTypeface(null, Typeface)`, `setTextSize(int textSize)`, `setTextColor(int Color)`.

Давайте теперь рассмотрим использование виджета `EditText` в приложении. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `EditTextApp`;
- ❑ **Application name** — `EditText Sample`;
- ❑ **Package name** — `com.samples.ui.scrollview`;
- ❑ **Create Activity** — `EditTextActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch06_EditText`.

В файле компоновки `main.xml` создадим элемент `EditText`, в котором добавим выводимую на экран подсказку: `android:hint="Enter some text..."`, как показано в листинге 6.4.

Листинг 6.4. Файл компоновки `main.xml`

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

    <EditText
        android:id="@+id/text"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:hint="Enter some text...">
    </EditText>
</LinearLayout>
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. При запуске приложения в виджете `EditText` будет выведена подсказка (мы задавали свойство

android:hint в файле компоновки). Когда пользователь начнет вводить текст, на экране автоматически появится экранная клавиатура. Внешний вид программы с элементом TextView показан на рис. 6.3.

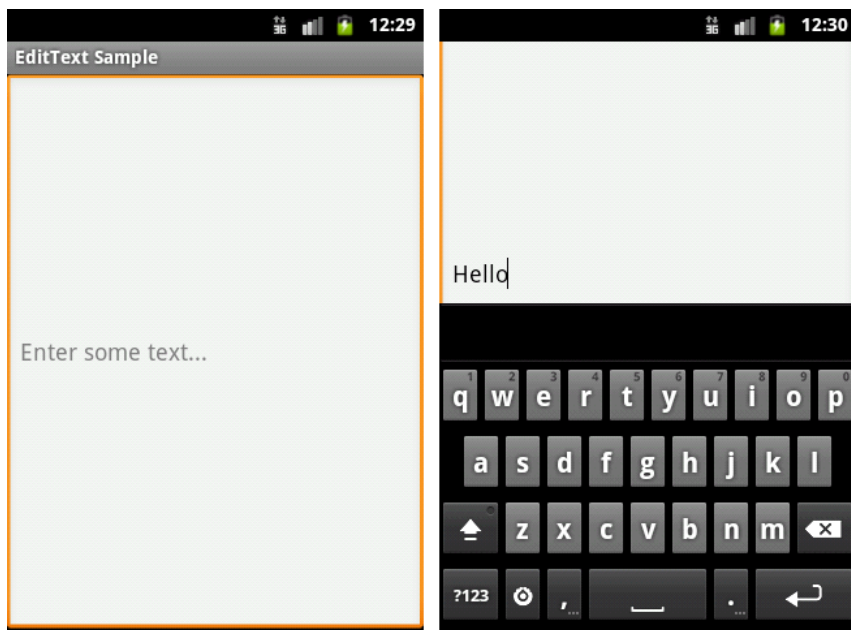


Рис. 6.3. Пример элемента EditText

Тип ввода текста

Возможность автоматического появления экранной клавиатуры заложена для всех виджетов, в которых предусмотрен ввод текста. На закладке **Text Fields** дизайнера компоновки находится множество вариантов текстовых полей с различным режимом ввода текста (рис. 6.4).

Давайте поэкспериментируем с несколькими типами текстовых полей. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — EditTextInputTypes;
- ☐ **Application name** — EditTextInputTypes;
- ☐ **Package name** — com.samples.ui.edittextinputtypes;
- ☐ **Create Activity** — EditTextActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch06_EditTextInputTypes.

В файл компоновки добавьте из группы **Text Fields** три текстовых поля разного типа: **E-mail**, **Number** и **Password**. Код файла компоновки представлен в листинге 6.5.

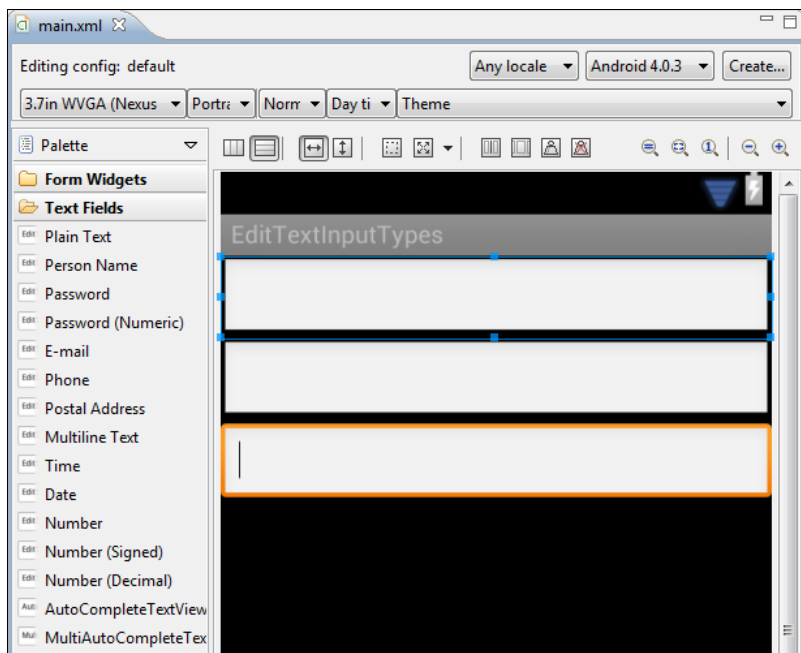


Рис. 6.4. Варианты типа ввода

Листинг 6.5. Файл компоновки main.xml

```
<?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" >
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress"/>
    <EditText
        android:id="@+id/editText3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number" />
    <EditText
        android:id="@+id/editText4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword" />
</LinearLayout>
```

Как вы видите, на самом деле все эти текстовые поля представлены одним виджетом `EditText`, а тип ввода текста определяется одним атрибутом `android:inputType`. Теперь

запустите приложение на выполнение. Если теперь сделать ввод в текстовые поля, в зависимости от типа поля система будет отображать соответствующую экранную клавиатуру. Например, для ввода адреса электронной почты на клавиатуре будет кнопка со значком @ и кнопка перехода **Next**, а на клавиатуре для ввода пароля вместо **Next** стоит кнопка **Done**. Если в текстовое поле разрешено вводить только цифры, будет отображена цифровая клавиатура (рис. 6.5).

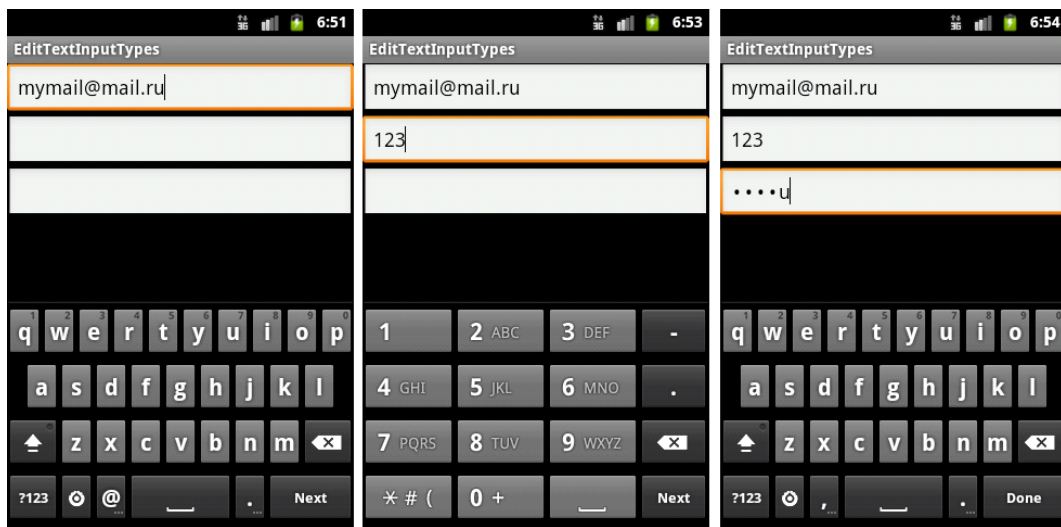


Рис. 6.5. Варианты экранной клавиатуры для текстовых полей с различным типом ввода текста

Как вы видите, Android предоставляет очень гибкую систему для ввода текста, и во многих случаях в программном коде вам нет необходимости определять маски для ввода или делать валидацию для типа введенной информации.

Режимы отображения клавиатуры

Небольшой экран мобильного устройства требует тщательной проработки расположения элементов пользовательского интерфейса. Экранная клавиатура также охватывает много места на экране. Но в то же время Android предоставляет возможности гибкой настройки для отображения экранной клавиатуры и текстовых полей.

Настройка режимов отображения устанавливается в файле манифеста приложения для каждого Activity, входящего в состав приложения. В элементе `<activity>` для этого существует необязательный атрибут (по умолчанию он отсутствует) `android:windowSoftInputMode`.

Этот атрибут определяет два типа вариантов поведения экранной клавиатуры и текстовых полей, находящихся в данном Activity — состояние отображения экранной клавиатуры и поведение текстовых полей при получении или потере фокуса и вводе текста.

Варианты поведения экранной клавиатуры могут быть следующими:

- ☐ `stateUnspecified` — система сама выбирает подходящее поведение экранной клавиатуры. Это настройка по умолчанию;

- ❑ `stateUnchanged` — экранная клавиатура при получении Activity (на котором текстовое поле) фокуса ввода сохраняет свое последнее состояние, видимое или скрытое;
- ❑ `stateHidden` — экранная клавиатура скрыта, когда пользователь открывает Activity. Клавиатура появится при наборе текста. Если пользователь перейдет к следующему Activity, клавиатура скроется. Но при возвращении назад клавиатура останется на экране, если она была видима при закрытии Activity;
- ❑ `stateAlwaysHidden` — экранная клавиатура всегда скрывается, если Activity получает фокус;
- ❑ `stateVisible` — экранная клавиатура видима в обычном режиме (поведение такое же, как в приложении из предыдущего раздела);
- ❑ `stateAlwaysVisible` — экранная клавиатура делается видимой сразу, когда пользователь открывает Activity.

Для определения поведения текстовых полей имеется три значения:

- ❑ `adjustResize` — размеры виджетов, находящихся на Activity, могут изменяться, освобождая место для экранной клавиатуры;
- ❑ `adjustPan` — окно Activity и его виджеты не изменяются при отображении экранной клавиатуры, только автоматически подстраиваются, чтобы текстовое поле, находящееся в фокусе ввода, не было закрыто клавиатурой. Такое поведение используется реже, так как остальные виджеты, находящиеся на этом Activity, например кнопки, становятся недоступными, и надо закрывать клавиатуру для доступа к ним;
- ❑ `adjustUnspecified` — система Android автоматически выберет один из перечисленных ранее режимов в зависимости от компоновки окна Activity и количества дочерних элементов управления на нем. Это настройка по умолчанию для Activity.

Поведение экранной клавиатуры и текстовых полей в атрибуте `android:windowSoftInputMode` задается их комбинацией с использованием знака "|", например, так:

```
<activity
    android:windowSoftInputMode="stateVisible|adjustResize"
    . . .
</activity>
```

Чтобы разобраться, как эти настройки поведения экранной клавиатуры и текстовых полей выглядят на экране, создадим в Eclipse новый проект:

- ❑ **Project name** — `EditTextInputModes`;
- ❑ **Application name** — `EditText Input Mode`;
- ❑ **Package name** — `com.samples.ui.edittextinputmode`;
- ❑ **Create Activity** — `EditTextActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch06_EditTextInputModes`.

В файле манифеста приложения добавим атрибут `android:windowSoftInputMode` и выставим ему значение `stateVisible|adjustPan`, как показано в листинге 6.6.

Листинг 6.6. Файл манифеста приложения AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.ui.edittextinputmode"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="10" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".EditInputModeActivity"
            android:windowSoftInputMode="stateVisible|adjustPan">
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

В файле компоновки главного окна приложения используем три текстовых поля такого же типа, как в предыдущей программе (см. листинг 6.5), но внесем некоторые изменения — с помощью атрибутов `android:layout_height` и `android:layout_weight` (кто забыл, что это такое, посмотрите в предыдущей главе) равномерно распределим эти поля на весь экран. Код файла компоновки представлен в листинге 6.7.

Листинг 6.7. Файл компоновки main.xml

```
<?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" >

    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:inputType="textEmailAddress"
        android:layout_weight="1"
        android:hint="Enter e-mail"/>

    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
        android:inputType="number">
```



```
    android:hint="Enter number"
    android:layout_weight="1" />
<EditText
    android:id="@+id/editText3"
    android:layout_width="match_parent"
    android:layout_height="fill_parent"
    android:inputType="textPassword"
    android:hint="Enter password"
    android:layout_weight="1" />
</LinearLayout>
```

Если запустить приложение и начать вводить информацию в текстовые поля, будет отображена экранная клавиатура и одновременно пропорционально уменьшатся размеры по вертикали всех текстовых полей, как показано на рис. 6.6.

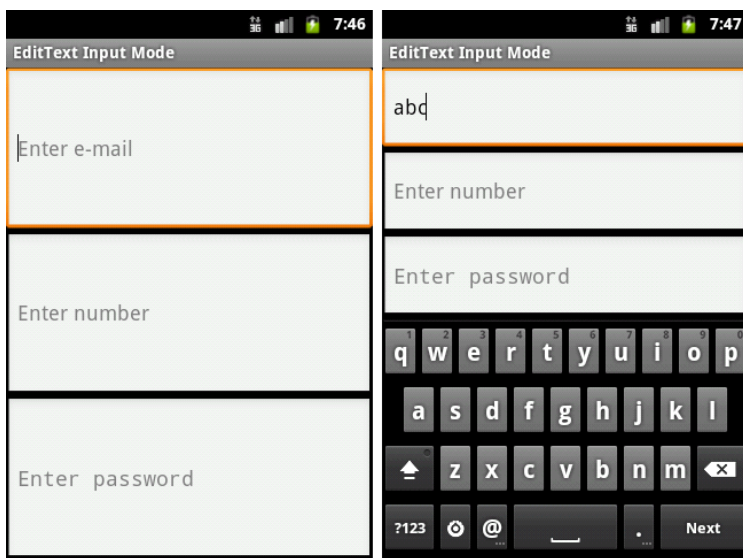


Рис. 6.6. Режим отображения с изменением размеров виджетов

Теперь изменим поведение приложения, поменяв значение в атрибуте `android:windowSoftInputMode` в манифесте приложения:

```
android:windowSoftInputMode="stateVisible|adjustPan"
```

Если перекомпилировать и запустить приложение, то при вводе текста в режиме `adjustPan` экранная клавиатура будет затенять часть окна Activity вместе с виджетами (рис. 6.7).

Есть еще один режим отображения программируемой клавиатуры. Он используется, когда система полагает, что на экране мало места для отображения всех виджетов. В этом случае система автоматически выведет только одно текстовое поле, которое находится в фокусе ввода, экранную клавиатуру и добавит на экран кнопку **Next** для возможности перехода к следующему виджету или возврату к отображению полного окна, если виджеты кончились. Например, в нашем приложении такая ситуация может возникнуть, если его перевернуть в горизонтальное положение (рис. 6.8).

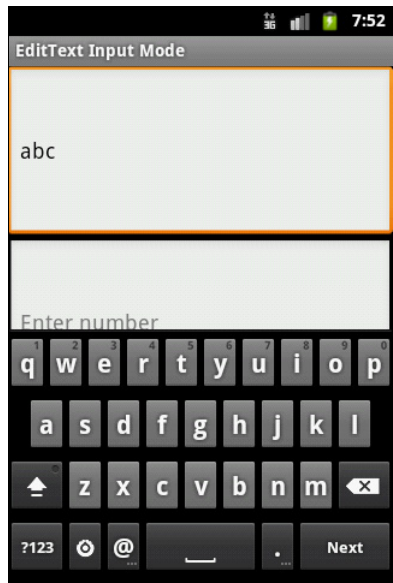


Рис. 6.7. Режим отображения с затемнением находящихся не в фокусе ввода элементов

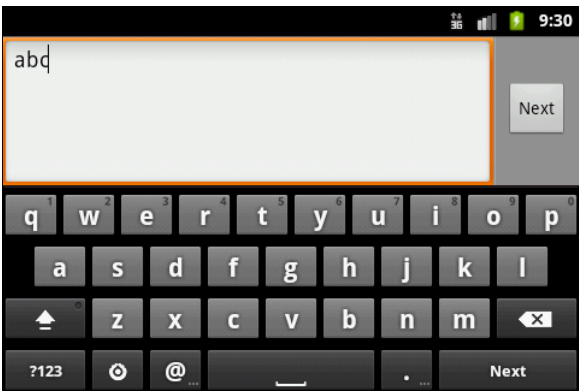


Рис. 6.8. Отображение системой дополнительной кнопки при недостатке места

Опция, определяющая отображение экранной клавиатуры, есть и в настройках мобильного устройства. Возможны случаи, когда использовать всплывающую клавиатуру нет необходимости. Тогда ее можно отключить, открыв настройки мобильного телефона и выбрав **Language & keyboard | Android keyboard | Popup on keypress**, как показано на рис. 6.9 (расположение опций и путь зависит от версии Android, у разных устройств окна настроек могут отличаться).

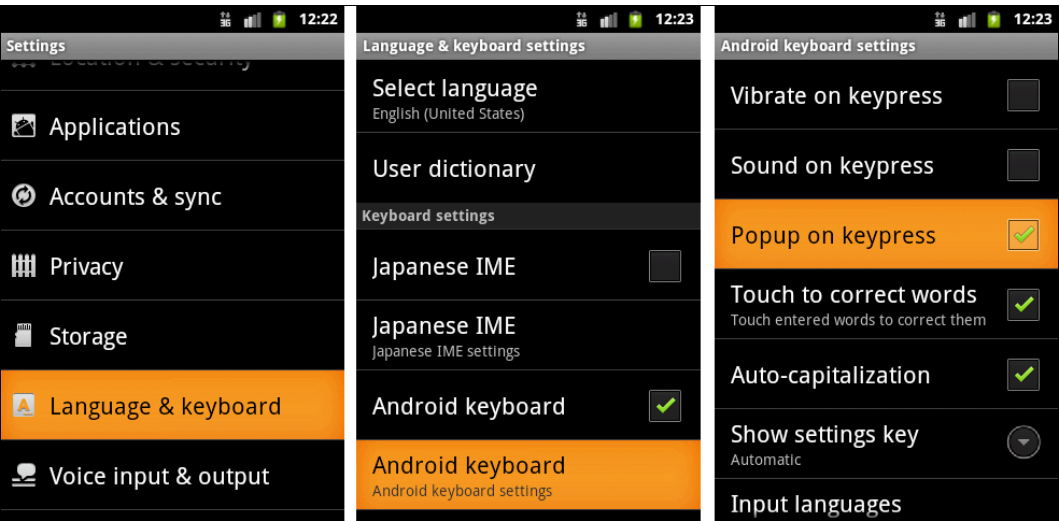


Рис. 6.9. Настройка параметров клавиатуры эмулятора Android

Полосы прокрутки

Класс `TextView` использует свою собственную прокрутку и в принципе не требует добавления отдельных полос прокрутки, но применение видимых полос прокрутки вместе с `TextView` (или любым другим элементом или контейнером, размеры которого больше размера экрана мобильного устройства) улучшает внешний вид и повышает удобство использования приложения.

Полосы прокрутки в Android представляют виджеты `ScrollView` и `HorizontalScrollView`, которые являются контейнерными элементами и наследуются от `ViewGroup`, как показано на рис. 6.10.

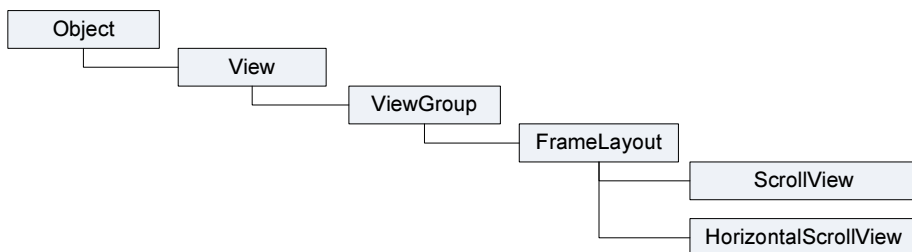


Рис. 6.10. Иерархия классов для `ScrollView` и `HorizontalScrollView`

Элементы `ScrollView` и `HorizontalScrollView` — это контейнеры типа `FrameLayout`, что означает, что в них можно разместить только одно дочернее представление. Этот дочерний элемент может, в свою очередь, быть контейнером со сложной иерархией объектов. В качестве дочернего элемента для полос прокрутки обычно используют `LinearLayout` с вертикальной или горизонтальной ориентацией элементов.

Виджет `ScrollView`, несмотря на свое название, поддерживает только вертикальную прокрутку, поэтому для создания вертикальной и горизонтальной прокрутки необходимо использовать `ScrollView` в сочетании с `HorizontalScrollView`. Обычно `ScrollView` используют в качестве корневого элемента, а `HorizontalScrollView` — дочернего.

Рассмотрим создание полос прокрутки на практике. В среде Eclipse создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ScrollsApp`;
- ☐ **Application name** — `ScrollViews Sample`;
- ☐ **Package name** — `com.samples.ui.scrolls`;
- ☐ **Create Activity** — `ScrollsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch06_ScrollViews`.

Откройте файл компоновки `main.xml` и создайте структуру компоновки, разместив в ней `ScrollView`, `HorizontalScrollView` и `TextView`, как показано в листинге 6.8.

Листинг 6.8. Файл компоновки main.xml

```

<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scroll"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <HorizontalScrollView
        android:id="@+id/scroll_hor"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="#000000"
            android:background="#FFFFFF"
            android:textSize="24px"
            android:shadowDx="0"
            android:shadowDy="0"
            android:shadowRadius="0"
            android:shadowColor="#FFFFFF"
            android:isScrollContainer="true"/>

        </HorizontalScrollView>
    </ScrollView>

```

В классе `ScrollsActivity` в теле метода `onCreate()` создайте ссылку на элемент `TextView`, объявленный в XML-компоновке, и запишите в него методом `setText()` достаточно большой текст, который гарантированно не поместится в видимые размеры экрана устройства. Код класса окна приложения `ScrollsActivity` представлен в листинге 6.9.

Листинг 6.9. Файл класса окна приложения ScrollActivity.java

```

package com.samples.ui.scrolls;

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

public class ScrollActivity extends Activity {

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

        TextView text = (TextView)findViewById(R.id.text);
    }
}

```

```
// Загружаем текст
text.setText("..."); // Введите любой достаточно большой текст
}
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Внешний вид программы с элементом `TextView` и полосами прокрутки показан на рис. 6.11.

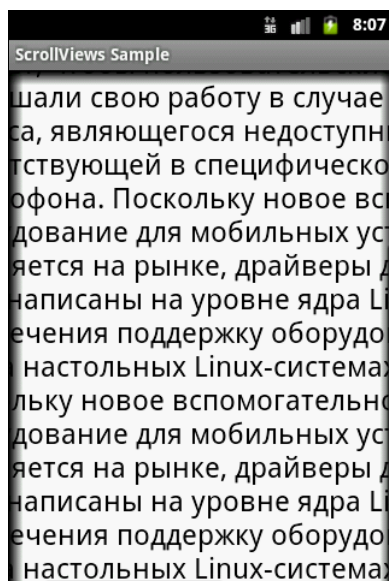


Рис. 6.11. Текстовое поле с горизонтальными и вертикальными полосами прокрутки

Отображение графики — *ImageView*

Для отображения графики предназначен виджет `ImageView`. Как и элемент `TextView`, который является базовым виджетом для текстового наполнения пользовательского интерфейса, `ImageView` является базовым элементом для графического наполнения и может использоваться в контейнерных виджетах для отображения графики.

Класс `ImageView` может загружать изображения из различных источников, таких как ресурсы приложения или внешние файлы с изображениями. В этом классе существует несколько методов для загрузки изображений:

- ☐ `setImageResource(int resId)` — загружает изображение по его идентификатору ресурса;
- ☐ `setImageURI(Uri uri)` — загружает изображение по его URI;
- ☐ `setImageBitmap(Bitmap bitmap)` — загружает растровое изображение.

Для загрузки изображения в XML-файле компоновки используется атрибут `android:src`.

Кроме того, в классе `ImageView` определены методы для установки размеров изображения — `setMaxHeight()`, `setMaxWidth()`, `getMinimumHeight()`, `getMinimumWidth()`, а также его масштабирования — `getScaleType()`, `setScaleType()`.

Для примера приложения с использованием виджета `ImageView` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `ImageViewApp`;
- ❑ **Application name** — `ImageView Sample`;
- ❑ **Package name** — `com.samples.ui.imageview`;
- ❑ **Create Activity** — `ImageViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch06_ImageView`.

В каталог `res/drawable/` проекта поместите два изображения: `android.png` и `androidmarker.png` (их можно найти в FTP-архиве книги, в каталоге `res/drawables/` проекта `Ch06_ImageView`). Откройте файл компоновки и создайте структуру компоновки, разместив в ней два элемента `ImageView` с идентификаторами `image1`, `image2`. Для первого элемента задайте атрибут `android:src="@drawable/android"`. Для второго элемента `ImageView` загрузка ресурса будет происходить в программном коде.

В листинге 6.10 приведен код файла компоновки `main.xml`.

Листинг 6.10. Файл компоновки `main.xml`

```
<?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:gravity="center"
    android:orientation="horizontal">

    <ImageView
        android:id="@+id/image1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/android"
        android:padding="10px"/>

    <ImageView
        android:id="@+id/image2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10px"/>

</LinearLayout>
```

В классе `ImageViewActivity` загрузите программно изображение для второго виджета `ImageView` методом `setImageResource()`, как показано в листинге 6.11.

Листинг 6.11. Файл класса окна приложения `ImageViewActivity.java`

```
package com.samples.ui.imageview;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
```

```
public class ImageViewActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        final ImageView image = (ImageView)findViewById(R.id.image2);  
  
        // Загрузка изображения из ресурса  
        image.setImageResource(R.drawable.androidmarker);  
    }  
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Результат должен получиться такой, как на рис. 6.12.



Рис. 6.12. Пример виджетов `ImageView` с загруженными изображениями

Резюме

В этой главе мы рассмотрели использование основных виджетов для отображения и редактирования текста и загрузки графики. Эти виджеты являются базовыми для большинства других виджетов, используемых для создания пользовательского интерфейса Android-приложения.

В следующей главе мы рассмотрим использование командных элементов управления — кнопок, переключателей, флажков, а также организацию взаимодействия с пользователем с помощью обработки событий в приложении.



ГЛАВА 7

Командные элементы управления и обработка событий

В этой главе мы будем изучать использование командных кнопок, флажков и переключателей и рассмотрим обработку событий, возникающих при взаимодействии пользователя с приложением. Затем мы рассмотрим контейнерные виджеты для группировки переключателей — `RadioGroup` и закладки — виджеты `TabHost` и `TabWidget`.

Обработка событий

После того как вы научились добавлять виджеты в пользовательский интерфейс, вам потребуется организовать взаимодействие виджетов с пользователем. Для этого необходимо определить обработчик события и зарегистрировать его для данного элемента.

Класс `View` содержит коллекцию вложенных интерфейсов, которые называются `On...Listener()`, в каждом из которых объявлен единственный абстрактный метод. Этот метод необходимо переопределить в вашем классе. Его будет вызывать система `Android`, когда с экземпляром `View`, к которому был подсоединен слушатель события, будет взаимодействовать пользователь.

Всего класс `View` содержит шесть вложенных интерфейсов:

- ☐ `OnClickListener`;
- ☐ `OnLongClickListener`;
- ☐ `OnFocusChangeListener`;
- ☐ `OnKeyListener`;
- ☐ `OnTouchListener`;
- ☐ `OnCreateContextMenuListener`.

Например, если требуется, чтобы командная кнопка получила уведомление о нажатии ее пользователем, необходимо в классе окна реализовать интерфейс `OnClickListener` и определить его метод обратного вызова `onClick()`, куда будет помещен код обработки нажатия кнопки, и зарегистрировать слушатель события с помощью метода `setOnClickListener()`:

```
button1.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mText.setText("Click on First Button");
    }
});
```

Существует несколько способов подключения событий, и о них будет рассказано далее в этой главе.

Кнопки и флажки

Кнопки и флажки в Android представлены следующими классами:

- ☐ Button;
- ☐ CheckBox;
- ☐ ToggleButton;
- ☐ RadioButton;
- ☐ ImageButton.

Почти у всех вышеперечисленных виджетов базовым классом является `TextView`, от которого они наследуют многочисленные методы для отображения и форматирования текста. Исключение составляет виджет `ImageButton`, который является наследником класса `ImageView` и представляет собой кнопку с изображением без текста.

Класс `Button` представляет командную кнопку и наследуется от `TextView`. Он является базовым классом для класса `CompoundButton`. От класса `CompoundButton` наследуются остальные кнопки: `CheckBox`, `ToggleButton` и `RadioButton`. Иерархия классов кнопок представлена на рис. 7.1.

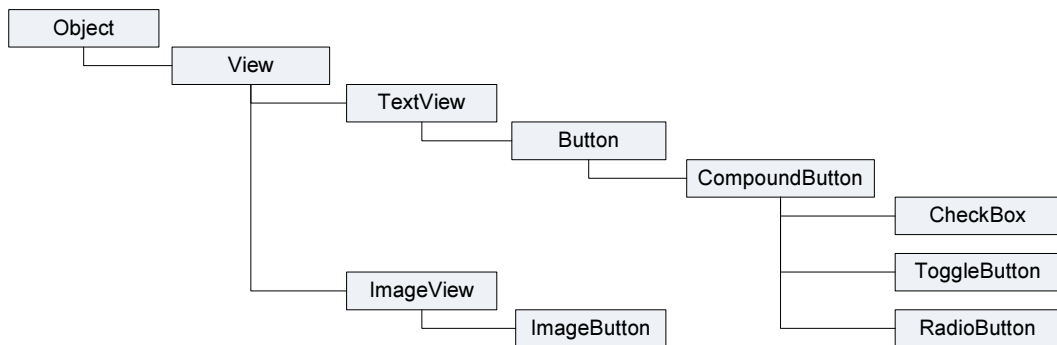


Рис. 7.1. Иерархия классов `Button`, `CheckBox`, `ToggleButton`, `RadioButton` и `ImageButton`

Класс `CompoundButton` представляет базовую функциональность для кнопок с двумя состояниями — `checked` и `unchecked`. При нажатии состояние кнопки изменяется на противоположное. Класс `CompoundButton` содержит вложенный интерфейс `OnCheckedChangeListener` с единственным методом `onCheckedChanged()`.

Button

Класс `Button` (Кнопка) — самый простой из всех элементов управления и при этом самый используемый. Чаще всего кнопка требует написания кода обработки события нажатия `onClick`.

Следующий пример реализует обработчик события `onClick()`. Когда выполняется щелчок на кнопке, появляется сообщение, отображающее имя кнопки. Создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ButtonApp`;
- ☐ **Application name** — `Button Sample`;
- ☐ **Package name** — `com.samples.ui.button`;
- ☐ **Create Activity** — `ButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_Button_v1`.

Откройте файл компоновки и создайте компоновку `LinearLayout` и в ней две кнопки с идентификаторами `button1` и `button2` и надписями **Button 1** и **Button 2** (листинг 7.1).

Листинг 7.1. Файл компоновки `main.xml`

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

    <Button
        android:id="@+id/button1"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Button 1"/>

    <Button
        android:id="@+id/button2"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Button 2" />

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Event:"
        android:textStyle="bold"/>

</LinearLayout>
```

Теперь в классе `ButtonActivity` подключим обработчики событий для кнопок, как показано в листинге 7.2.

Листинг 7.2. Файл класса окна приложения `ButtonActivity.java`

```
package com.samples.ui.button2;

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

```

import android.widget.Button;
import android.widget.TextView;
import android.view.View;

public class ButtonActivity extends Activity {

    private TextView mText;

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

        mText = (TextView) findViewById(R.id.text);

        final Button button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mText.setText("Click on First Button");
            }
        });

        final Button button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mText.setText("Click on Second Button");
            }
        });
    }
}

```

Выполните компиляцию проекта. При нажатии пользователем соответствующей кнопки в надписи под кнопками будет отображаться сообщение о нажатии этой кнопки (рис. 7.2).

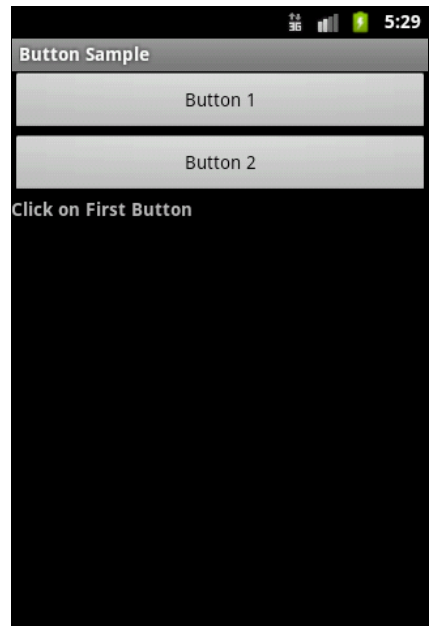


Рис. 7.2. Пример приложения с кнопками и обработчиком события `onClick()`

Существуют и другие варианты подключения обработки событий. В предыдущем примере обработчики событий были реализованы внутри тела метода `onCreate()`. Наличие множества вложенных блоков кода создает трудности восприятия кода, особенно другими программистами, поэтому желательно выносить обработчики событий за пределы метода `onCreate()`. В методе `setOnClickListener()` в качестве параметра передается имя метода обработчика, который мы должны будем реализовать:

```
button.setOnClickListener(button_click);
```

Далее мы описываем реализацию этого метода:

```
public OnClickListener button_click = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // Действия по обработке события
    }
};
```

Внесите в класс `ButtonActivity` изменения, как показано в листинге 7.3, и скомпилируйте проект. Результат не изменился, но код класса стал легче для восприятия.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_Button_v2`.

Листинг 7.3. Подключение обработчиков событий. Второй вариант

```
public class ButtonActivity extends Activity {
    private TextView mText;

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

        mText = (TextView) findViewById(R.id.text);
        final Button button1 = (Button) findViewById(R.id.button1);
        final Button button2 = (Button) findViewById(R.id.button2);

        button1.setOnClickListener(button1_click);
        button2.setOnClickListener(button2_click);
    }

    public OnClickListener button1_click = new OnClickListener() {
        @Override
        public void onClick(View v) {
            mText.setText("Click on First Button");
        }
    };

    public OnClickListener button2_click = new OnClickListener() {
        @Override
```

```

        public void onClick(View v) {
            mText.setText("Click on Second Button");
        }
    };
}

```

Наконец, есть еще способ, более эффективный, чем предыдущие, — реализовать обработку однотипных событий всех элементов в одном методе. Для этого в нашем классе необходимо реализовать интерфейс `View.OnClickListener`:

```

public class EditTextActivity extends Activity
    implements View.OnClickListener

```

Этот интерфейс содержит единственный метод:

```

abstract void onClick(View v),

```

который необходимо определить в нашем классе `ButtonActivity`. Если определен идентификатор элемента (например, в файле компоновки `Activity`), то можно написать обработку событий элементов в операторе `switch`, получив ID элемента методом `getId()`:

```

@Override
public void onClick(View v) {
    switch (v.getId()) {
        // Определяем ID элемента и обрабатываем событие
    }
}

```

Если в методе `onClick()` обрабатывается только одно событие, структура `switch` не нужна.

Теперь используем данный способ обработки событий в нашем приложении, изменив код класса `ButtonActivity`, как показано в листинге 7.4.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_Button_v3`.

Листинг 7.4. Подключение обработчиков событий. Третий вариант

```

public class ButtonActivity extends Activity
    implements View.OnClickListener {
    private TextView mText;

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

        mText = (TextView) findViewById(R.id.text);
        final Button button1 = (Button) findViewById(R.id.button1);
        final Button button2 = (Button) findViewById(R.id.button2);
    }
}

```

```
        button1.setOnClickListener(this);
        button2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button1:
                mText.setText("Click on First Button");
                break;
            case R.id.button2:
                mText.setText("Click on Second Button");
                break;
        }
    }
}
```

Вариант написания обработки одностипных событий, представленный в листинге 7.4, является наиболее предпочтительным в том случае, если у вас несколько одностипных элементов управления. Такой вариант лучше структурирует код и делает его более понятным для восприятия. Кроме того, при обработке меню и диалоговых окон используется такой же стиль написания кода.

Приведенный в листинге 7.4 код можно еще упростить, если вынести объявление обработчика события кнопок в файл компоновки.

Теперь в качестве примера обработки событий от нескольких элементов управления напомним простой текстовый редактор с пятью кнопками для форматирования вводимого текста. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — EditTextApp;
- ☐ **Application name** — Editor Sample;
- ☐ **Package name** — com.samples.ui.editttext;
- ☐ **Create Activity** — AutoCompleteTextViewActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch07_Editor.

Создайте файл компоновки, как в листинге 7.5.

Листинг 7.5. Файл компоновки main.xml

```
<?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">
    <TableRow
        android:layout_width="wrap_content"
```

```

android:layout_height="wrap_content"
android:gravity="center">
<Button
    android:id="@+id/button_r"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="R"/>
<Button
    android:id="@+id/button_b"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="B"/>
<Button
    android:id="@+id/button_i"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="I"/>
<TextView
    android:id="@+id/label"
    android:layout_height="wrap_content"
    android:text="TextSize"
    android:paddingLeft="10px"
    android:layout_width="wrap_content"/>
<Button
    android:id="@+id/button_plus"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="+"/>
<Button
    android:id="@+id/button_minus"
    android:layout_width="50dp"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="-"/>
</TableRow>
<EditText
    android:id="@+id/edit_text"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:text="Hello, Android"/>
</TableLayout>

```

В нашем примере кроме элемента `EditText` будет небольшое меню из пяти кнопок для изменения стиля текста и его размера (листинг 7.6).

Листинг 7.6. Файл класса окна приложения EditTextActivity.java

```
package com.samples.ui.editor;

import android.app.Activity;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.EditText;

public class EditorActivity extends Activity
    implements OnClickListener {

    private float mTextSize = 20;
    private EditText mEdit;

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

        mEdit = (EditText) findViewById(R.id.edit_text);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button_r:
                mEdit.setTypeface(null, Typeface.NORMAL);
                break;
            case R.id.button_b:
                mEdit.setTypeface(null, Typeface.BOLD);
                break;
            case R.id.button_i:
                mEdit.setTypeface(null, Typeface.ITALIC);
                break;
            case R.id.button_plus:
                if (mTextSize <= 72)
                    mTextSize+=2;
                mEdit.setTextSize(mTextSize);
                break;
            case R.id.button_minus:
                if (mTextSize >= 20)
                    mTextSize-=2;
                mEdit.setTextSize(mTextSize);
                break;
        }
    }
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Получившийся простейший текстовый редактор позволяет форматировать вводимый текст, изменяя его стиль и размер (рис. 7.3).

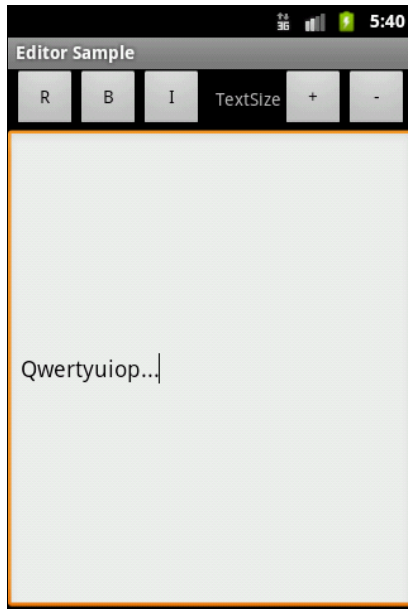


Рис. 7.3. Простой текстовый редактор

RadioButton и RadioGroup

Виджеты `RadioButton` (Переключатели) обычно используются в составе группы — контейнера `RadioGroup`. Контейнер `RadioGroup` наследуется от `ViewGroup` и может использоваться в качестве корневого элемента компоновки окна, если на экране будет располагаться только группа переключателей, или в качестве вложенного в другой контейнер, например `LinearLayout`.

Переключатели дают возможность пользователю выбирать одну из нескольких опций. Когда вы используете множество элементов управления `RadioButton` в одном контейнере, выбранным может быть только один из них. Поэтому если у вас есть три опции, например **Red**, **Green** и **Blue**, и если выбрана опция **Red**, а пользователь щелкает на **Blue**, то опция **Red** автоматически отключается. Основным методом для изменения состояния — `toggle()`, который инвертирует состояние переключателя. Кроме того, от базового класса наследуются и другие методы, например `isChecked()`, который возвращает состояние кнопки, и `setChecked()`, изменяющий состояние кнопки в зависимости от параметра.

Для примера приложения с переключателями создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `RadioButtonApp`;
- ☐ **Application name** — `RadioButton Sample`;
- ☐ **Package name** — `com.samples.ui.radiobutton`;
- ☐ **Create Activity** — `RadioButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_RadioButton`.

Откройте файл компоновки и создайте структуру компоновки с корневым контейнером `RadioGroup`, тремя переключателями и элементом `TextView`. Присвойте кнопкам ID `radio1`, `radio2`, `radio3` и надписи **Mode #1**, **Mode #2**, **Mode #3**, как в листинге 7.7.

Листинг 7.7. Файл компоновки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <RadioButton
        android:id="@+id/radio1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Mode #1"
        android:onClick="onClick"
        android:checked="true"/>

    <RadioButton
        android:id="@+id/radio2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Mode #2"/>

    <RadioButton
        android:id="@+id/radio3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Mode #3"/>

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:textStyle="bold"/>

</RadioGroup>
```

Для определения выбранной опции добавьте обработчик события `onClick()` на кнопках. В классе `RadioButtonActivity` напишите код подобно листингу 7.8.

Листинг 7.8. Файл класса окна приложения RadioButtonActivity.java

```
package com.samples.ui.radiobutton;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
```

```

import android.widget.RadioButton;
import android.widget.TextView;

public class RadioButtonDemo extends Activity
    implements View.OnClickListener {
    private TextView text;

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

        text = (TextView)findViewById(R.id.text);
    }

    public void onClick(View v) {
        // Действия при наступлении события
        // Идентифицируем сработавший переключатель
        // и выводим в текстовое поле надпись на переключателе
        RadioButton rb = (RadioButton)v;
        text.setText("Select: " + rb.getText());
    }
}

```

Выполните компиляцию проекта. Выбранная опция отображается в элементе `TextView`, изменение текста которого происходит в обработчике `radioButton_Click`. Внешний вид программы представлен на рис. 7.4.

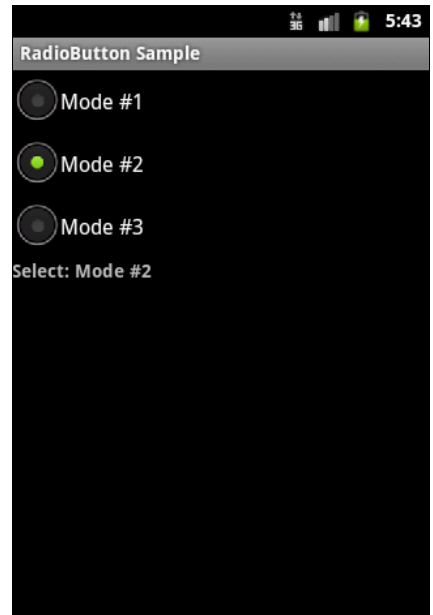


Рис. 7.4. Пример элементов `RadioButton`

CheckBox

Элемент `CheckBox` (Флажок) — это переключатель с двумя состояниями. Для программного отслеживания изменения состояния элемента необходимо реализовать интерфейс `CompoundButton.OnCheckedChangeListener`.

Сейчас мы рассмотрим использование `CheckBox` и обработку событий в приложении. Создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `CheckBox`;
- ☐ **Application name** — `CheckBox Sample`;
- ☐ **Package name** — `com.samples.ui.checkbox`;
- ☐ **Create Activity** — `CheckBox Activity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_CheckBox`.

Откройте файл компоновки и создайте структуру компоновки с одним элементом `CheckBox` подобно листингу 7.9.

Листинг 7.9. Файл компоновки `main.xml`

```
<?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">

    <CheckBox
        android:id="@+id/checkbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CheckBox OFF"/>

</LinearLayout>
```

В классе `CheckBoxActivity` подсоедините слушатель события для флажка и в обработчике события сделайте проверку его состояния (листинг 7.10).

Листинг 7.10. Файл класса окна приложения `CheckBoxActivity.java`

```
package com.samples.ui.checkbox;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;

public class CheckBoxActivity extends Activity
    implements CompoundButton.OnCheckedChangeListener {

    private CheckBox mCheckBox;

    @Override
```

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    mCheckBox = (CheckBox)findViewById(R.id.checkbox);  
  
    // Регистрация слушателя события  
    mCheckBox.setOnCheckedChangeListener(this);  
}  
  
// Обработчик события изменения состояния флажка  
public void onCheckedChanged(  
    CompoundButton buttonView, boolean isChecked) {  
    if (isChecked)  
        mCheckBox.setText("CheckBox ON");  
    else  
        mCheckBox.setText("CheckBox OFF");  
}  
}
```

Выполните компиляцию проекта. Программа обрабатывает событие изменения состояния флажка, отображая текущее состояние в его текстовой метке. Результат должен получиться подобно рис. 7.5.

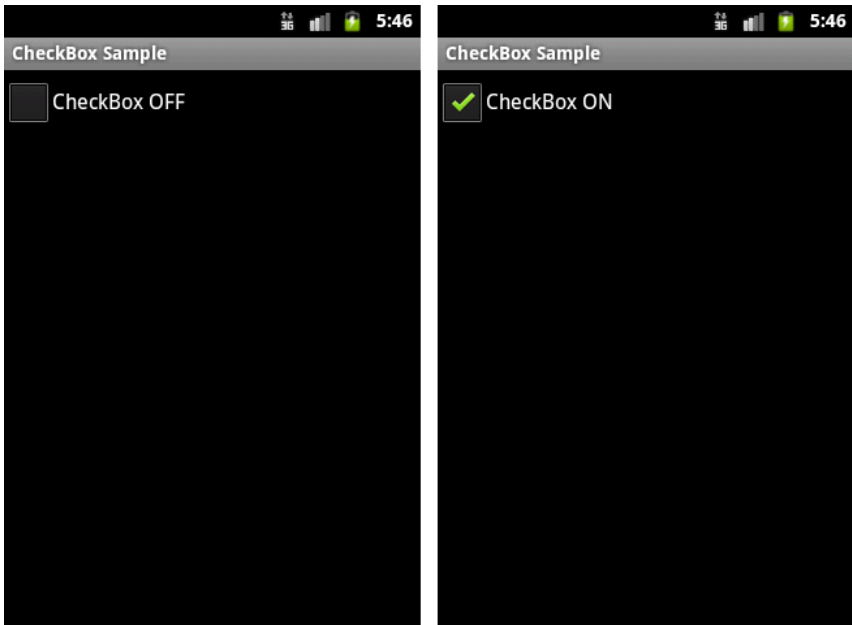


Рис. 7.5. Элемент CheckBox

ToggleButton

Виджет `ToggleButton` — это кнопка с двумя состояниями: "включено" и "выключено". По умолчанию на кнопке определены надписи ON/OFF и LED-индикатор, изменяющий цвет на зеленый при переключении в состояние ON.

Основные свойства `ToggleButton` — `android:textOff` и `android:textOn`, устанавливающие надписи на кнопке в разных состояниях. В программном коде им соответствуют методы `setTextOff()` и `setTextOn()`.

Метод `setChecked(boolean checked)` позволяет программно менять состояние кнопки. Основное событие для кнопки `ToggleButton` такое же, как и для флажка — изменение состояния кнопки `onCheckedChanged()` из интерфейса `CompoundButton.OnCheckedChangeListener`.

В качестве примера создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ToggleButtonApp`;
- ☐ **Application name** — `ToggleButton Sample`;
- ☐ **Package name** — `com.samples.ui.togglebutton`;
- ☐ **Create Activity** — `ToggleButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_ToggleButton`.

Откройте файл компоновки и создайте структуру компоновки, добавив элементы `ToggleButton` и `TextView` для вывода сообщения о состоянии кнопки (листинг 7.11).

Листинг 7.11. Файл компоновки `main.xml`

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

    <ToggleButton
        android:id="@+id/button"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Для создания обработчика события изменения состояния кнопки в нашем классе необходима реализация интерфейса `CompoundButton.OnCheckedChangeListener`.

Этот интерфейс имеет единственный метод `onCheckedChanged()`, который необходимо переопределить в нашем классе. При обработке события для определения состояния используется параметр `isChecked`. Код класса `ToggleButtonActivity` представлен в листинге 7.12.

Листинг 7.12. Файл класса окна приложения `ToggleButtonActivity.java`

```
package com.samples.ui.togglebutton;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CompoundButton;
import android.widget.ToggleButton;
import android.widget.TextView;

public class ToggleButtonActivity extends Activity
    implements CompoundButton.OnCheckedChangeListener {

    private TextView mText;

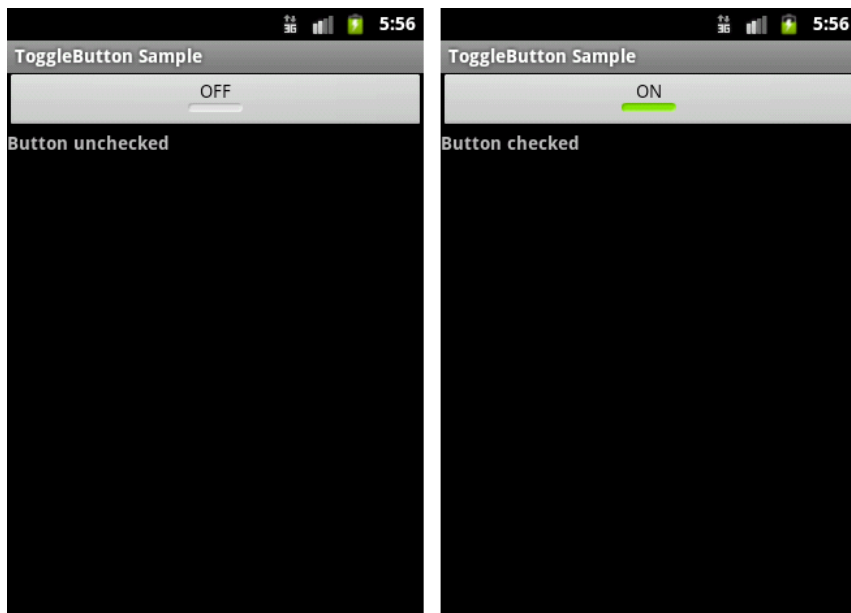
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final ToggleButton mButton = (ToggleButton)findViewById(
            R.id.button);
        mText = (TextView)findViewById(R.id.text);

        mButton.setOnCheckedChangeListener(this);
    }

    @Override
    public void onCheckedChanged(
        CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            mText.setText("Button checked");
        }
        else {
            mText.setText("Button unchecked");
        }
    }
}
```

Выполните компиляцию проекта. При нажатии кнопки пользователем будет меняться надпись на кнопке с OFF на ON и цвет LED-индикатора, как показано на рис. 7.6.

Рис. 7.6. Элемент `ToggleButton`

ImageButton

Виджет `ImageButton` представляет собой кнопку с изображением (вместо текста). По умолчанию `ImageButton` похож на обычный элемент `Button`, со стандартным фоном кнопки, который изменяет цвет во время других состояний кнопки.

Изображение на поверхности кнопки определяется атрибутом `android:src` в элементе `<ImageButton>` или в программном коде методом `setImageResource(int)`.

Рассмотрим простой пример с этой кнопкой. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ImageButtonWidget`;
- ☐ **Application name** — `ImageButton Sample`;
- ☐ **Package name** — `com.samples.ui.imagebutton`;
- ☐ **Create Activity** — `ImageButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_ImageButton`.

Откройте файл компоновки `main.xml` и создайте структуру компоновки с элементом `<ImageButton>` подобно листингу 7.13.

Листинг 7.13. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```

android:layout_width="fill_parent"
android:layout_height="fill_parent">

<ImageButton
    android:id="@+id/ImageButton01"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:onClick="onClick"
    android:src="@drawable/play"/>

</LinearLayout>

```

В папку `res/drawable/` поместите изображения для отображения состояний кнопки (можно взять готовые в каталоге `Resources/Images/` из FTP-архива книги — файлы `play.png` и `pause.png`).

Для класса `ImageButtonActivity` напишите код, как в листинге 7.14.

Листинг 7.14. Файл класса окна приложения `ImageButtonActivity.java`

```

package com.samples.ui.imagebutton;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;

public class ImageButtonActivity extends Activity
    implements View.OnClickListener {

    private ImageButton button;
    private boolean play = true;

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

        button = (ImageButton) findViewById(R.id.ImageButton01);

        // Устанавливаем изображение на кнопке по умолчанию
        button.setImageResource(R.drawable.play);
    }

    public void onClick(View v) {
        // Меняем изображение на кнопке
        if (play) {
            button.setImageResource(R.drawable.pause);
        }
    }
}

```

```
else {  
    button.setImageResource(R.drawable.play);  
}  
  
play = !play;  
}  
}
```

Выполните компиляцию проекта и запустите его на эмуляторе Android. При нажатии кнопки будет загружаться соответствующее изображение для этой кнопки, как показано на рис. 7.7.

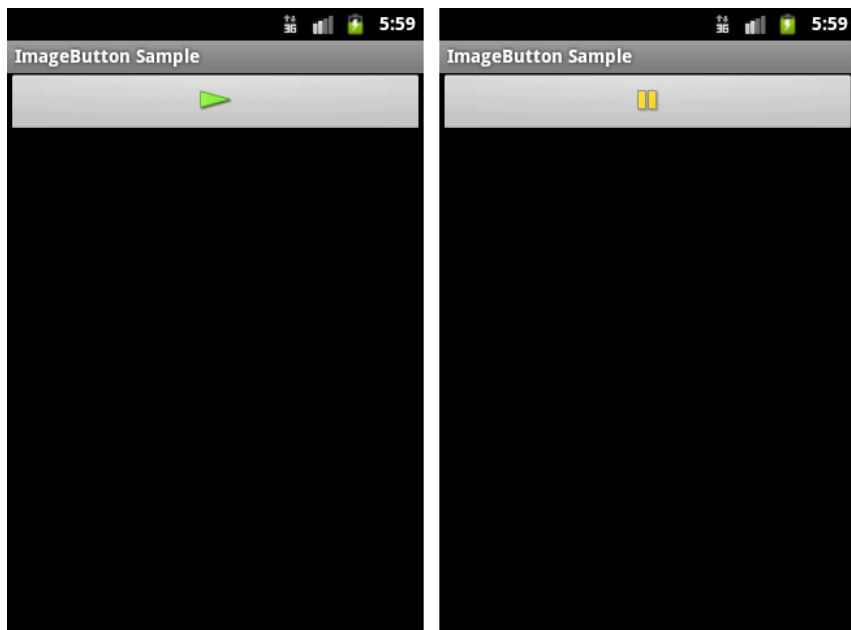


Рис. 7.7. Изменение изображения при нажатии кнопки ImageButton

Закладки

Закладки в Android являются контейнерными виджетами и представлены классами `TabHost` и `TabWidget` (рис. 7.8).

Виджет `TabHost` позволяет группировать связанные элементы управления в серии страниц-вкладок. Элемент `TabHost` является контейнером для коллекции элементов типа `TabWidget`. Когда пользователь выбирает закладку, этот объект посылает сообщение в родительский контейнер `TabHost` для переключения на выбранную закладку.

Контейнерный виджет `TabHost` используется в основном только для добавления закладок и обработки вызовов выбранных закладок. Основные методы для работы с `TabHost`:

- ❑ `setup()` — инициализирует контейнер закладок. Необходимо вызывать перед добавлением закладок, если `TabHost` загружается методом `findViewById()`;

- ❑ `addTab()` — добавляет новую закладку;
- ❑ `setCurrentTab()` — ставит заданную закладку на передний план.

Большинство методов для работы с закладками реализованы в классе `TabWidget`. Чтобы работать с закладками в программном коде, для каждой закладки должны быть определены три свойства:

- ❑ индикатор позиции табуляции — текст, отображаемый на закладке, например "Document 1";
- ❑ информационное наполнение — элемент или контейнер, расположенный на данной закладке;
- ❑ тег для идентификации в программном коде.

Эти свойства необходимо определить для каждой закладки созданием экземпляра вложенного класса `TabSpec` следующим образом:

```
TabHost tabs;
...
// Задаем тег для индентификации в программном коде
TabHost.TabSpec spec = tabs.newTabSpec("tag1");
// Задаем информационное наполнение
spec.setContent(R.id.tabPage1);
// Задаем индикатор
spec.setIndicator("Document 1");
// Добавляем закладку в коллекцию закладок объекта TabHost
tabs.addTab(spec);
```

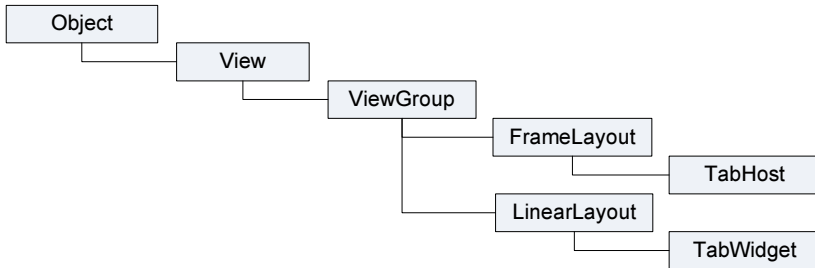


Рис. 7.8. Иерархия классов закладок

Рассмотрим создание закладок на примере. Создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `Tabs`;
- ❑ **Application name** — `TabHost Sample`;
- ❑ **Package name** — `com.samples.ui.tabhost`;
- ❑ **Create Activity** — `TabHostActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch07_Tabs`.

В нашей программе мы создадим документ, состоящий из трех закладок, в каждой из которых будет текстовое поле `EditText`. Откройте файл компоновки `main.xml` и напишите структуру компоновки подобно листингу 7.15.

Листинг 7.15. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TabWidget android:id="@android:id/tabs"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    <FrameLayout
        android:id="@android:id/tabcontent"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:paddingTop="65px">

        <EditText android:id="@+id/tabPage1"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:hint="Enter text for page 1"/>
        <EditText android:id="@+id/tabPage2"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:hint="Enter text for page 2"/>
        <EditText android:id="@+id/tabPage3"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:hint="Enter text for page 3"/>
    </FrameLayout>
</TabHost>
```

В классе `TabHostActivity` реализуем приведенную ранее последовательность инициализации `TabHost` и добавление к нему закладок. Полный код класса приведен в листинге 7.16.

Листинг 7.16. Файл класса окна приложения `TabHostActivity.java`

```
package com.samples.ui.tabhost;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TabHost;
```

```

public class TabHostActivity extends Activity {

    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.main);

        TabHost tabs = (TabHost) findViewById(R.id.tabhost);
        tabs.setup();

        TabHost.TabSpec spec = tabs.newTabSpec("tag1");

        spec.setContent(R.id.tabPage1);
        spec.setIndicator("Document 1");
        tabs.addTab(spec);

        spec = tabs.newTabSpec("tag2");
        spec.setContent(R.id.tabPage2);
        spec.setIndicator("Document 2");
        tabs.addTab(spec);

        spec = tabs.newTabSpec("tag3");
        spec.setContent(R.id.tabPage3);
        spec.setIndicator("Document 3");
        tabs.addTab(spec);

        tabs.setCurrentTab(0);
    }
}

```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Внешний вид программы с виджетом TabHost и тремя закладками показан на рис. 7.9.

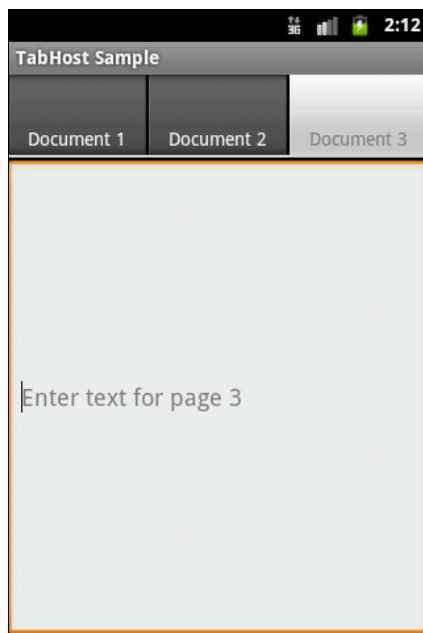


Рис. 7.9. Контейнерный виджет TabHost с тремя закладками

Динамическое создание элементов управления

В некоторых случаях во время выполнения программы нам может потребоваться создание компоновки и элементов управления "на лету", т. е. в коде программы.

В программном коде при создании компоновки используются те же принципы, как и при создании компоновки в XML. Библиотека Android в пакете `android.widget` предоставляет набор классов для создания компоновок аналогичным объявлением в XML, которые мы изучили в предыдущей главе:

- ☐ `LinearLayout`;
- ☐ `FrameLayout`;
- ☐ `TableLayout`;
- ☐ `RelativeLayout`.

Для создания корневой разметки мы создаем экземпляр одного из этих классов, в конструктор которого передается контекст приложения. Контекст приложения — это набор информации о среде, в которой выполняется приложение. В классе окна приложения контекст приложения можно получить вызовом метода `getApplicationContext()`. Например, создать корневую компоновку типа `LinearLayout` можно следующим образом:

```
LinearLayout layout = new LinearLayout(getApplicationContext());
```

В классе, реализующем `Activity`, необязательно даже явно вызывать метод `getApplicationContext()` для получения контекста приложения, можно просто использовать ссылку на данный экземпляр класса `this`, который уже содержит контекст приложения:

```
LinearLayout layout = new LinearLayout(this);
```

Далее необходимо задать параметры компоновки, создав экземпляр класса `LinearLayout.LayoutParams` и передав его в метод `setLayoutParams()` следующим образом:

```
LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(  
    LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);  
layout.setLayoutParams(params);
```

Для задания способа расположения дочерних элементов в классах компоновки (вертикально или горизонтально) есть метод `setOrientation()`, параметром которого служат две целочисленные константы, `HORIZONTAL` и `VERTICAL`:

```
layout.setOrientation(LinearLayout.VERTICAL);
```

Дочерние виджеты, например, кнопки, текстовые поля создаются аналогично корневой компоновке. Например, создать кнопку можно таким образом:

```
Button btn = new Button(this);  
btn.setText("Button1");  
btn.setId(ID_BUTTON1);  
btn.setLayoutParams(new LinearLayout.LayoutParams(  
    LayoutParams.FILL_PARENT, LayoutParams.WRAP_CONTENT));  
btn.setOnClickListener(this);
```

Давайте теперь напишем приложение, в котором будет динамически создаваться окно с кнопками и текстовым полем. Создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — DynamicUI;
- ☐ **Application name** — Dynamic UI;
- ☐ **Package name** — com.samples.ui.dynamicui;
- ☐ **Create Activity** — DynamicUIActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch07_DynamicUI.

В классе `DynamicUIActivity` в теле метода `onCreate()` надо закомментировать вызов метода `setContentView()`, который загружает XML-компоновку, он нам не понадобится, всю компоновку мы создадим в программном коде. Например, сделаем окно с корневой компоновкой `LinearLayout` с вертикальной ориентацией дочерних виджетов, двух кнопок и текстового поля. К кнопкам подключим обработчики событий, при нажатии кнопки в текстовое поле будет выводиться информация о нажатой кнопке. Код класса `DynamicUIActivity` представлен в листинге 7.17.

Листинг 7.17. Файл класса окна приложения `DynamicUIActivity.java`

```
package com.samples.ui.dynamicui;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class DynamicUIActivity extends Activity
    implements View.OnClickListener {

    private static final int ID_BUTTON1 = 101;
    private static final int ID_BUTTON2 = 102;
    private static final int ID_TEXT = 103;

    private TextView text;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Комментируем метод, он нам не нужен
        //setContentView(R.layout.main);

        // Создаем корневой Layout
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
```



```
layout.setLayoutParams(new LinearLayout.LayoutParams (
    LayoutParams.FILL_PARENT,
    LayoutParams.FILL_PARENT));

// Создаем 1-ю кнопку
Button button1 = new Button(this);
button1.setText("Button1");
button1.setId(ID_BUTTON1);
button1.setLayoutParams(new LinearLayout.LayoutParams (
    LayoutParams.FILL_PARENT,
    LayoutParams.WRAP_CONTENT));
button1.setOnClickListener(this);

// Создаем 2-ю кнопку
Button button2 = new Button(this);
button2.setText("Button2");
button2.setId(ID_BUTTON2);
button2.setLayoutParams(new LinearLayout.LayoutParams (
    LayoutParams.FILL_PARENT,
    LayoutParams.WRAP_CONTENT));
button2.setOnClickListener(this);

// Создаем текстовое поле
text = new TextView(this);
text.setId(ID_TEXT);
text.setText("Event:");
text.setLayoutParams(new LinearLayout.LayoutParams (
    LayoutParams.WRAP_CONTENT,
    LayoutParams.WRAP_CONTENT));

layout.addView(button1);
layout.addView(button2);
layout.addView(text);

// Добавляем созданный контент на Activity
this.addContentView(layout, new LinearLayout.LayoutParams (
    LayoutParams.FILL_PARENT,
    LayoutParams.FILL_PARENT));
}

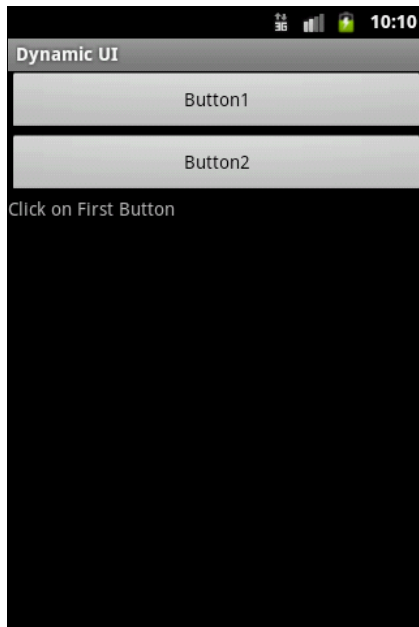
// Обработчик событий нажатия кнопок
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case ID_BUTTON1:
            text.setText("Click on First Button");
            break;
        case ID_BUTTON2:
            text.setText("Click on Second Button");
    }
}
```

```
        break;  
    }  
}  
}
```

Внешний вид окна приложения и работа с ним ничем не отличается от окна со статической компоновкой (рис. 7.10).

Безусловно, динамическое создание пользовательского интерфейса очень удобно применять в приложениях, требующих гибкого изменения внешнего вида окон приложения, но надо учитывать, что создание пользовательского интерфейса в коде приложения занимает более длительное время, чем при задании компоновки окна в статических файлах XML. Поэтому использовать создание пользовательского интерфейса "на лету" лучше, когда в этом действительно есть необходимость.

Рис. 7.10. Окно с динамически созданными элементами управления



Резюме

В этой главе мы рассмотрели использование командных виджетов (кнопок, флажков и переключателей) и обработку событий для организации взаимодействия пользователя с Android-приложением. Также мы изучили создание закладок на основе элементов `TabHost` и `TabWidget`.

В следующей главе мы рассмотрим использование виджета `ProgressBar` для отображения задач, требующих длительного времени выполнения, а также особенности порождения потоков в приложениях с графическим интерфейсом.



ГЛАВА 8

Отображение длительно выполняющихся задач

В этой главе мы рассмотрим индикаторы, слайдеры и компоненты отображения времени. Некоторые операции требуют для своего выполнения длительного времени, от нескольких секунд до минут и более. Для отображения степени завершенности данных операций обычно используются индикаторы. Примерами подобных задач могут быть загрузка файла из сетевого ресурса, перенос мультимедийных файлов на карту памяти или поиск доступной точки подключения к Wi-Fi. Такие операции требуется выполнять в фоновом потоке, поэтому в данной главе мы также рассмотрим создание фоновых потоков в приложении.

Создание фоновых потоков

Выполнение длительной задачи лучше производить в отдельном потоке. Android предоставляет класс `Handler` для порождения фоновых потоков и их безопасного взаимодействия с пользовательским интерфейсом.

Самое удобное средство порождения фонового потока — создать экземпляр `Handler` в классе `Activity`. Фоновый поток может взаимодействовать с объектом `Handler`, который, в свою очередь, будет обновлять графический интерфейс в основном потоке `Activity` (например, шкалу `ProgressBar`) фрагментов.

Чтобы послать сообщение в объект `Handler`, сначала необходимо вызвать метод `obtainMessage()`, чтобы извлечь объект `Message` из глобального пула сообщений:

```
Handler h;
...
// Получаем сообщение
Message msg = mHandler.obtainMessage();
// Вставляем сообщение в очередь сообщений объекта Handler
h.sendMessage(msg);
```

Для вставки сообщения в очередь сообщений объекта `Handler` существует несколько методов:

- ☐ `sendMessage()` — помещает сообщение в очередь немедленно (в конец очереди);
- ☐ `sendMessageAtFrontOfQueue()` — помещает сообщение в очередь немедленно и, кроме того, помещает это сообщение впереди очереди (по умолчанию оно ставится в конец очереди), таким образом ваше сообщение берет приоритет над всеми другими;

- ❑ `sendMessageAtTime()` — помещает сообщение в очередь в установленное время в миллисекундах;
- ❑ `sendMessageDelayed()` — помещает сообщение в очередь после задержки, выраженной в миллисекундах.

Чтобы обрабатывать эти сообщения, для объекта `Handler` необходимо реализовать метод обратного вызова `handleMessage()`, который будет вызываться каждым сообщением из очереди сообщения.

```
Handler h;
...
h = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        // Код для обработки
        ...
    }
};
```

Виджет *ProgressBar*

Для отображения степени завершенности длительных задач в приложении применяется элемент управления `ProgressBar`. Основные методы, используемые для работы с объектом `ProgressBar`:

- ❑ `setProgress()` — устанавливает заданное значение прогресса;
- ❑ `getProgress()` — возвращает текущее значение прогресса;
- ❑ `incrementProgressBy()` — устанавливает величину дискретизации приращения значения прогресса;
- ❑ `setMax()` — устанавливает максимальное значение величины прогресса.

Для примера создадим приложение с `ProgressBar`, который будет отображать ход выполнения длительной задачи (это будет простой цикл с приостановкой потока на 0,1 секунды в каждой итерации цикла) и обновлять степень завершения этой задачи через объект `Handler` в классе `Activity`. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `ProgressBarApp`;
- ❑ **Application name** — `ProgressBar Sample`;
- ❑ **Package name** — `com.samples.ui.progressbar`;
- ❑ **Create Activity** — `ProgressBarActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch08_ProgressBar`.

В файле компоновки создайте контейнер `LinearLayout`, добавьте в него `ProgressBar` и две кнопки, **Start** и **Stop**, как показано в листинге 8.1.

Листинг 8.1. Файл компоновки main.xml

```

<?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:gravity="center"
    android:orientation="vertical">
    <TextView
        android:id="@+id/text"
        android:text="Progress: 0%"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_margin="5px"
        android:textStyle="bold"/>
    <ProgressBar
        android:id="@+id/progress"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <LinearLayout
        android:id="@+id/LinearLayout01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_margin="5px">

        <Button
            android:id="@+id/button_start"
            android:layout_height="wrap_content"
            android:text="Start"
            android:layout_weight="1"
            android:onClick="onClick"
            android:layout_width="fill_parent"/>

        <Button
            android:id="@+id/button_stop"
            android:layout_height="wrap_content"
            android:text="Stop"
            android:layout_weight="1"
            android:onClick="onClick"
            android:layout_width="fill_parent"/>

    </LinearLayout>
</LinearLayout>

```

В классе `ProgressBarActivity` создадим отдельный поток, работающий достаточно длительное время, состояние которого будем отображать в `ProgressBar`. Код класса `ProgressBarActivity` представлен в листинге 8.2.

Листинг 8.2. Файл класса окна приложения ProgressBarActivity.java

```
package com.samples.ui.progressbar;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.TextView;

public class ProgressBarActivity extends Activity
    implements View.OnClickListener {
    private ProgressBar progress;
    private TextView text;
    private boolean isRunning = false;

    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            progress.incrementProgressBy(1);
            text.setText("Progress: " + progress.getProgress() + "%");
        }
    };

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

        progress = (ProgressBar)findViewById(R.id.progress);
        text = (TextView)findViewById(R.id.text);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button_start:
                onStart();
                break;
            case R.id.button_stop:
                onStop();
                break;
        }
    }

    public void onStart() {
        super.onStart();
        progress.setProgress(0);
    }
}
```

```
// Создаем новый поток
Thread background = new Thread(new Runnable() {
    public void run() {
        while (isRunning) {
            try {
                Thread.sleep(100);
            }
            catch (InterruptedException e) {
                Log.e("ERROR", "Thread Interrupted");
            }
            handler.sendMessage(handler.obtainMessage());
        }
    }
});
isRunning = true;
background.start();
}

public void onStop() {
    super.onStop();
    isRunning = false;
}
}
```

Запустите проект на выполнение. Внешний вид программы представлен на рис. 8.1.

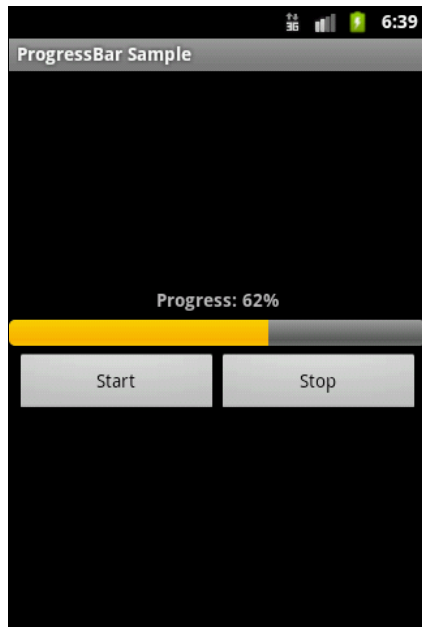


Рис. 8.1. Виджет `ProgressBar` в действии

Расширения класса *ProgressBar*

Кроме виджета `ProgressBar`, библиотека виджетов содержит набор виджетов, являющихся наследниками класса `ProgressBar` и довольно часто используемых для создания графического пользовательского интерфейса, — это индикаторы и слайдеры, представленные классами `RatingBar` и `SeekBar`.

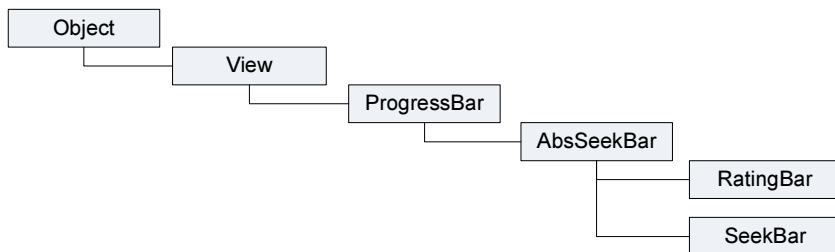


Рис. 8.2. Иерархия классов для ProgressBar, RatingBar и SeekBar

Иерархия классов для этих виджетов показана на рис. 8.2.

Классы RatingBar (отображает рейтинг в виде звездочек) и SeekBar (слайдер) являются близкими родственниками ProgressBar. Это специализированные расширения классов ProgressBar и AbsSeekBar. Класс AbsSeekBar предоставляет базовую функциональность для интерактивного взаимодействия пользователя с элементами RatingBar и SeekBar.

SeekBar

Виджет SeekBar — это слайдер (ползунок). Пользователь может коснуться пальцем или использовать клавиши курсора и переместить движок влево или вправо, чтобы установить нужное положение. Этот виджет пригодится нам в следующих главах, мы будем его использовать в приложениях, поэтому есть смысл рассмотреть его подробнее.

Для программного отслеживания перемещения ползунка SeekBar необходимо реализовать вложенный интерфейс SeekBar.OnSeekBarChangeListener. Этот интерфейс объявляет три метода, которые необходимо переопределить в классе Activity:

- ☐ onProgressChanged() — уведомление о том, что положение ползунка изменилось;
- ☐ onStartTrackingTouch() — уведомление о том, что пользователь начал перемещение ползунка;
- ☐ onStopTrackingTouch() — уведомление о том, что пользователь закончил перемещение ползунка.

Основное событие виджета SeekBar, которое используют для программного отслеживания перемещения ползунка, — SeekBar.OnSeekBarChangeListener.

Для примера приложения с SeekBar создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — SeekBar;
- ☐ **Application name** — SeekBar Sample;
- ☐ **Package name** — com.samples.ui.seekbar;
- ☐ **Create Activity** — SeekBarActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch08_SeekBar.

В файле компоновки создайте контейнер `LinearLayout`, добавьте в него элемент `SeekBar` и два текстовых поля для отображения значения, установленного слайдером (листинг 8.3).

Листинг 8.3. Файл компоновки main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:padding="10px">

    <SeekBar
        android:id="@+id/seek_bar"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="10px">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Value: "
            android:textStyle="bold"/>
        <TextView
            android:id="@+id/text_value"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="0"
            android:textStyle="bold"/>
    </LinearLayout>
</LinearLayout>

```

В классе `SeekBarActivity` реализуем обработку события остановки ползунка, значение которого будет выводиться в текстовое поле. Полный код класса приведен в листинге 8.4.

Листинг 8.4. Файл класса окна приложения SeekBarActivity.java

```

package com.samples.ui.seekbar;

import android.app.Activity;
import android.os.Bundle;
import android.widget.SeekBar;
import android.widget.TextView;

public class SeekBarActivity extends Activity
    implements SeekBar.OnSeekBarChangeListener {

    TextView mTextValue;

```

```

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

    final SeekBar seekBar = (SeekBar)findViewById(R.id.seek_bar);
    seekBar.setOnSeekBarChangeListener(this);

    mTextView = (TextView)findViewById(R.id.text_value);
    mTextView.setText("0");
}

// Обработка события остановки ползунка
@Override
public void onStopTrackingTouch(SeekBar seekBar) {
    mTextView.setText(String.valueOf(seekBar.getProgress()));
}

// Остальные методы, реализующие интерфейс OnSeekBarChangeListener,
// в этой программе используются только как заглушки
@Override
public void onProgressChanged(SeekBar seekBar, int progress,
    boolean fromUser) {
    // TODO Auto-generated method stub
    // TODO Сгенерированная автоматически заглушка для метода
}
@Override
public void onStartTrackingTouch(SeekBar seekBar) {
    // TODO Auto-generated method stub
    // TODO Сгенерированная автоматически заглушка для метода
}
}

```

Запустите проект на выполнение. Внешний вид программы представлен на рис. 8.3. При перемещении ползунка в текстовом поле будет отображаться его текущее значение.

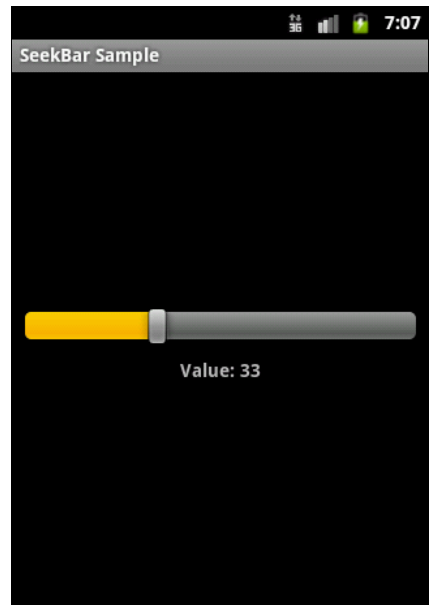


Рис. 8.3. Приложение с виджетом SeekBar

RatingBar

Виджет `RatingBar` — расширение классов `AbsSeekBar` и `ProgressBar`, который показывает значение рейтинга в виде звездочек. Пользователь может касанием пальца или с помощью клавиш курсора устанавливать оценку (рейтинг), используя заданное заранее максимальное количество звездочек в элементе `RatingBar`. Элемент `RatingBar` также может отображать рейтинг в режиме без взаимодействия с пользователем "только для чтения".

Основные методы, используемые при работе с `RatingBar`:

- ☐ `setNumStars(int)` — устанавливает количество звездочек;
- ☐ `getRating()` — возвращает значение рейтинга;
- ☐ `isIndicator()` — устанавливает `RatingBar` в режим "только для чтения";
- ☐ `setRating(float)` — устанавливает значение рейтинга;
- ☐ `setStepSize(float)` — устанавливает значение приращения рейтинга.

Кроме того, `RatingBar` имеет вложенный интерфейс `OnRatingBarChangeListener` для реализации отслеживания изменения рейтинга в приложении.

Для примера приложения с `RatingBar` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `RatingBar`;
- ☐ **Application name** — `RatingBar Sample`;
- ☐ **Package name** — `com.samples.ui.ratingbar`;
- ☐ **Create Activity** — `RatingBarActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch08_RatingBar`.

В файле компоновки создайте контейнер `LinearLayout`, добавьте в него `RatingBar`, две кнопки, **Up** и **Down**, и текстовые поля для отображения рейтинга в числовом виде (листинг 8.5).

Листинг 8.5. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <RatingBar
        android:id="@+id/rating"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <LinearLayout
        android:layout_width="wrap_content"
```

```

android:layout_height="wrap_content"
android:padding="20px">
<Button
    android:id="@+id/button_up"
    android:layout_height="wrap_content"
    android:text="Up"
    android:onClick="onClick"
    android:layout_width="60px"/>
<Button
    android:id="@+id/button_down"
    android:layout_height="wrap_content"
    android:text="Down"
    android:onClick="onClick"
    android:layout_width="60px"/>
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Value: "
    android:padding="10px"
    android:textStyle="bold"/>
<TextView
    android:id="@+id/text_value"
    android:layout_height="wrap_content"
    android:layout_width="40px"
    android:textStyle="bold"/>

```

```
</LinearLayout>
```

```
</LinearLayout>
```

В классе `RatingBarActivity` реализованы методы обратного вызова для кнопок **Up** и **Down**, устанавливающие значения рейтинга без контакта пользователя с элементом `RatingBar`, и обработчик события `OnRatingBarChangeListener` для отображения изменения численного значения рейтинга (листинг 8.6).

Листинг 8.6. Файл класса окна приложения `RatingBarActivity.java`

```

package com.samples.ui.ratingbar;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.RatingBar;
import android.widget.TextView;

public class RatingBarActivity extends Activity
    implements View.OnClickListener, RatingBar.OnRatingBarChangeListener {

    private static final int NUM_STARS = 5;

```

```
private float step = 0.5f;
private float rat = 1.0f;

private RatingBar ratBar;
private TextView text;

@Override
public void onCreate(Bundle icle) {
    super.onCreate(icle);
    setContentView(R.layout.main);

    ratBar = (RatingBar)findViewById(R.id.rating);
    ratBar.setNumStars(NUM_STARS);
    ratBar.setRating(rat);
    ratBar.setStepSize(0.5f);
    ratBar.setOnRatingBarChangeListener(this);

    text = (TextView)findViewById(R.id.text_value);
    text.setText(String.valueOf(rat));
}

// Обработчик события щелчка на кнопках
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_up:
            rat += step;
            if (rat > NUM_STARS) {
                rat = NUM_STARS;
            }
            ratBar.setRating(rat);
            break;
        case R.id.button_down:
            rat -= step;
            if (rat < 0) {
                rat = 0;
            }
            ratBar.setRating(rat);
            break;
    }
}

// Обработчик события изменения рейтинга
@Override
public void onRatingChanged(RatingBar ratingBar, float rating,
    boolean fromUser) {
    text.setText(String.valueOf(ratingBar.getRating()));
    rat = ratingBar.getRating();
}
}
```

Запустите проект на выполнение. Внешний вид программы представлен на рис. 8.4. Приложение позволяет пользователю взаимодействовать с RatingBar двумя способами: используя сенсорный режим, устанавливая значение рейтинга в RatingBar касанием звездочек пальцем, и через кнопки Up и Down.

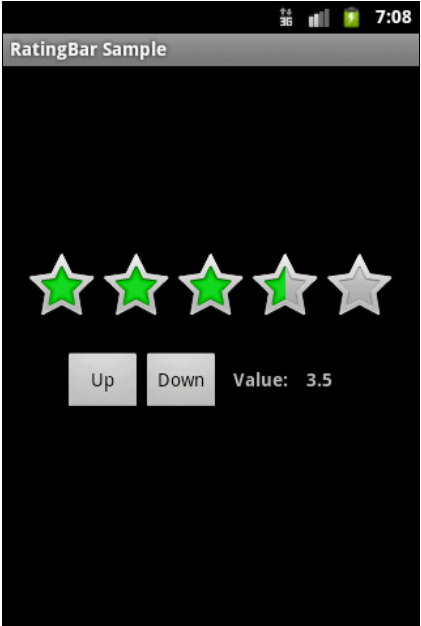


Рис. 8.4. Приложение с виджетом RatingBar

Использование системных таймеров и отображение системного времени

Виджеты для отображения времени представлены тремя классами:

- ❑ AnalogClock;
- ❑ DigitalClock;
- ❑ Chronometer.

Для вывода системного времени используются виджеты DigitalClock и AnalogClock. Они чрезвычайно удобны, поскольку автоматически синхронизируются с системным временем. Иерархия классов для AnalogClock, DigitalClock и Chronometer представлена на рис. 8.5.

Виджеты AnalogClock и DigitalClock очень простые и служат только для отображения системного времени пользователю (рис. 8.6).

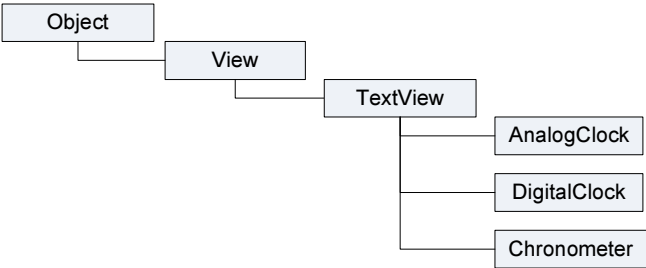
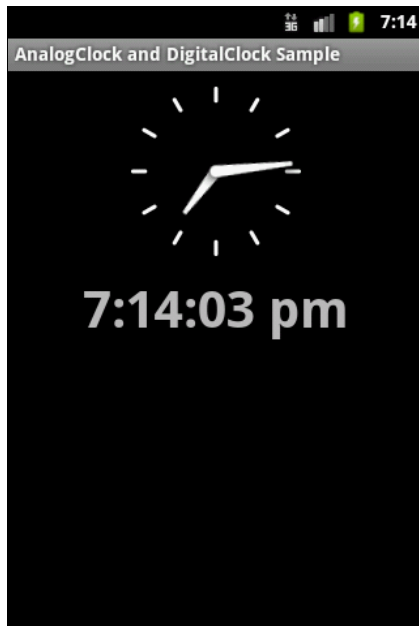


Рис. 8.5. Иерархия классов для AnalogClock, DigitalClock и Chronometer

Рис. 8.6. Виджеты для отображения времени



Виджет `Chronometer` является более интересным элементом управления и представляет собой управляемый таймер. Он позволяет пользователю запускать и останавливать отсчет времени, задавать время запуска таймера.

Основные методы для работы с виджетом `Chronometer`:

- ❑ `start()` — запускает отсчет времени;
- ❑ `stop()` — останавливает отсчет времени;
- ❑ `setFormat()` — задает формат отображения времени. По умолчанию текущее значение таймера отображается в формате "MM:SS" или "H:MM:SS".

Класс `Chronometer` имеет вложенный интерфейс `OnChronometerTickListener`, который содержит два метода:

- ❑ `getOnChronometerTickListener();`
- ❑ `setOnChronometerTickListener().`

Эти методы предназначены для реализации отслеживания изменения значения таймера в приложении.

Для примера приложения, использующего виджет `Chronometer`, создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `ChronometerApp`;
- ❑ **Application name** — `Chronometer Sample`;
- ❑ **Package name** — `com.samples.ui.chronometer`;
- ❑ **Create Activity** — `ChronometerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch08_Chronometer`.

В файле компоновки создайте корневой контейнер `LinearLayout`, в котором поместите элемент `Chronometer` и три кнопки для управления виджетом — **Start**, **Stop** и **Reset** (листинг 8.7).

Листинг 8.7. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```

android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:gravity="center">
<Chronometer
    android:id="@+id/chronometer"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="36px"
    android:gravity="center"/>
<LinearLayout
    android:id="@+id/LinearLayout01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <Button
        android:id="@+id/button_start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Start"/>
    <Button
        android:id="@+id/button_stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Stop"/>
    <Button
        android:id="@+id/button_reset"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Reset"/>
    </LinearLayout>
</LinearLayout>

```

В классе `ChronometerActivity` напишите обработчики событий кнопок для запуска, остановки и сброса таймера, как показано в листинге 8.8.

Листинг 8.8. Файл класса окна приложения `ChronometerActivity.java`

```

package com.samples.ui.chronometer;

import android.app.Activity;
import android.os.Bundle;
import android.os.SystemClock;
import android.view.View;
import android.widget.Chronometer;

public class ChronometerActivity extends Activity
    implements View.OnClickListener {

```



```
private Chronometer chronometer;

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

    chronometer = (Chronometer)findViewById(R.id.chronometer);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button_start:
            chronometer.start();
            break;
        case R.id.button_stop:
            chronometer.stop();
            break;
        case R.id.button_reset:
            chronometer.setBase(SystemClock.elapsedRealtime());
            break;
    }
}
}
```

Выполните компиляцию проекта и запустите его в эмуляторе Android. Внешний вид экрана приложения с виджетом `Chronometer` и кнопками для его управления представлен на рис. 8.7.

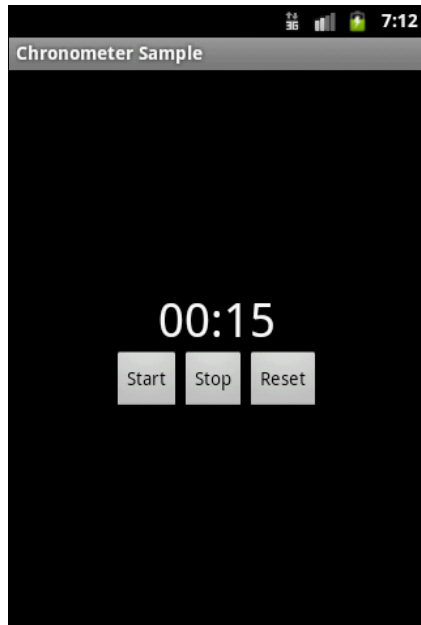


Рис. 8.7. Приложение с виджетом `Chronometer`

Резюме

В этой главе мы изучили использование виджетов для отображения хода выполнения длительных задач. Мы также разобрали создание фоновых потоков в приложениях с графическим интерфейсом пользователя, использование таймеров и виджетов для отображения времени.

На этом мы заканчиваем рассмотрение виджетов и переходим к изучению пользовательских уведомлений, которые применяются для оповещения пользователя о событиях, возникающих в приложении или системе. К виджетам мы еще вернемся, когда будем изучать использование и загрузку ресурсов, работу с базами данных и создание графики и анимации в приложении, для отображения которых в Android имеется набор специализированных элементов управления.



ГЛАВА 9

Уведомления

При работе пользователя с приложением могут возникать различные ситуации, о которых необходимо уведомить пользователя.

Для информирования пользователя о наступившем событии существует два типа уведомлений:

- ❑ **Toast Notification** — всплывающие уведомления, для кратких сообщений, не требующих реакции пользователя;
- ❑ **Status Bar Notification** — уведомления в строке состояния, которые требуют обязательной реакции пользователя для их закрытия (например, приход SMS или MMS).

В этой главе мы изучим создание всплывающих уведомлений. Уведомления в строке состояния мы будем изучать несколько позже, в *главе 27*, в которой мы рассмотрим их создание и взаимодействие со службами, т. к. для создания уведомления в строке состояния требуется взаимодействие с системной службой Notification Service.

Всплывающие уведомления

Всплывающее уведомление является сообщением, которое появляется на поверхности окна приложения. Всплывающее уведомление заполняет необходимое ему количество пространства, требуемого для сообщения, и текущий Activity приложения остается видимым и интерактивным для пользователя. Само уведомление в течение нескольких секунд плавно закрывается и не принимает события взаимодействия. Всплывающее уведомление также может быть создано службой, работающей в фоновом режиме.

Всплывающее уведомление обычно применяется для коротких текстовых сообщений. Для создания всплывающего уведомления сначала необходимо одним из методов `Toast.makeText()` инициализировать объект `Toast`, затем вызовом метода `show()` отобразить сообщение на экране, как показано в следующем примере:

```
Context context = getApplicationContext();  
Toast toast = Toast.makeText(context,  
    "This is Toast Notification", Toast.LENGTH_SHORT);  
toast.show();
```

Метод `makeText()` принимает три параметра:

- ❑ контекст приложения;
- ❑ текстовое сообщение;

- ❑ продолжительность времени показа уведомления, которое определяется двумя константами:
 - `LENGTH_SHORT` — показывает текстовое уведомление на короткий промежуток времени (является значением по умолчанию);
 - `LENGTH_LONG` — показывает текстовое уведомление в течение длительного периода времени.

Продолжительность времени показа уведомления можно также задавать, выставив конкретное значение в миллисекундах.

По умолчанию стандартное всплывающее уведомление появляется в нижней части экрана. Изменить место появления уведомления можно с помощью метода `setGravity(int, int, int)`. Этот метод принимает три параметра:

- ❑ стандартная константа для размещения объекта в пределах потенциально большего контейнера, определенная в классе `Gravity` (например, `GRAVITY.CENTER`, `GRAVITY.TOP` и др.);
- ❑ смещение по оси *X*;
- ❑ смещение по оси *Y*.

Например, если уведомление должно появляться в центральной части экрана, необходимо добавить следующий код:

```
toast.setGravity(Gravity.CENTER, 0, 0);
```

Если требуется сместить уведомление вправо, необходимо увеличить значение второго параметра. Чтобы сместить уведомление вниз — увеличить значение последнего параметра.

Для примера приложения со всплывающим уведомлением создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `ToastNotification`;
- ❑ **Application name** — `Toast Notification Sample`;
- ❑ **Package name** — `com.samples.ui.toastnotification`;
- ❑ **Create Activity** — `ToastNotificationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится на прилагаемом к книге электронном архиве в каталоге `Ch09_ToastNotification`.

Откройте файл компоновки и создайте структуру с элементом `Button` для вызова уведомления подобно листингу 9.1.

Листинг 9.1. Файл компоновки `main.xml`

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

```
<Button
    android:id="@+id/button"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="Call a Toast Notification"/>
```

```
</LinearLayout>
```

В классе `ToastActivity`, реализующем `Activity`, напишите код, как в листинге 9.2.

Листинг 9.2. Файл класса окна приложения `ToastActivity.java`

```
package com.samples.ui.toastnotification;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.Gravity;
import android.view.View;
import android.widget.Toast;

public class ToastActivity extends Activity
    implements View.OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
    }

    public void onClick(View view) {
        // Получаем контекст приложения
        Context context = getApplicationContext();

        // Создаем уведомление
        Toast toast = Toast.makeText(context,
            "This is Toast Notification", Toast.LENGTH_SHORT);

        // Устанавливаем уведомление в центр экрана
        toast.setGravity(Gravity.CENTER, 0, 0);

        // Отображаем уведомление на экране
        toast.show();
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова должно появиться на несколько секунд окно уведомления с текстовым сообщением (рис. 9.1).

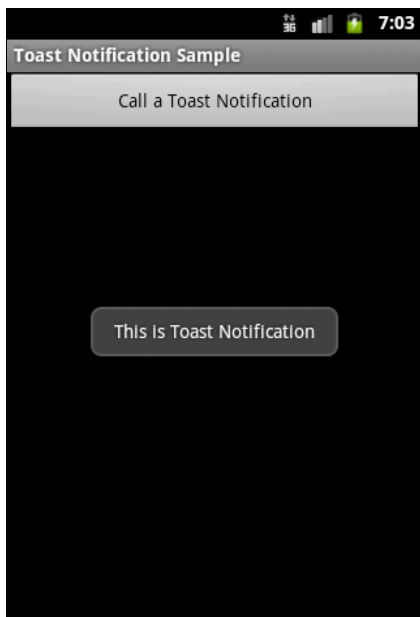


Рис. 9.1. Пример вызова всплывающего уведомления

Создание уведомлений с заданной компоновкой

Если простого текстового сообщения недостаточно для уведомления пользователя приложения, можно создать собственный дизайн компоновки уведомления.

Для получения компоновки из XML-файла и работы с ней в программе используется класс `LayoutInflater` и его методы `getLayoutInflater()` или `getSystemService()`, которые возвращают объект `LayoutInflater`. Затем вызовом метода `inflate()` получают корневой объект `View` этой компоновки. Например, для файла компоновки уведомления с именем `custom_layout.xml` и его корневого представления с идентификатором `android:id="@+id/toast_layout"` код будет таким:

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

Параметры, передаваемые в метод `inflate()`:

- ❑ идентификатор ресурса схемы размещения (в примере — `custom_layout.xml`);
- ❑ идентификатор ресурса корневого представления (в примере — `toast_layout`).

После получения корневого представления из него можно получить все дочерние представления уже известным методом `findViewById()` и определить информационное наполнение для этих элементов.

Затем создается объект `Toast` и устанавливаются нужные свойства, такие, например, как `Gravity` и продолжительность времени показа уведомления.

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

После этого вызывается метод `setView()`, которому передается компоновка уведомления, и метод `show()`, чтобы отобразить уведомление с собственной компоновкой:

```
toast.setView(layout);  
toast.show();
```

Для примера приложения с вызовом собственного уведомления создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `CustomToastNotification`;
- ☐ **Application name** — `Custom Toast Sample`;
- ☐ **Package name** — `com.samples.ui.customtoast`;
- ☐ **Create Activity** — `CustomToastActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch09_CustomToastNotification`.

Файл компоновки экрана используйте из предыдущего примера (см. листинг 9.1). Для создания новой компоновки уведомления щелкните правой кнопкой мыши на папке с проектом и в контекстном меню выберите **Android Tools | New Resource File**. Появится окно **New Android XML File**. В поле **File** введите имя нового файла компоновки — `custom_layout.xml`, в группе переключателей **What type of resource would you like to create?** установите значение **Layout**, поля **Folder** и **Select the root element for the XML file** оставьте без изменений (рис. 9.2).

Нажмите кнопку **Finish**. В созданном файле компоновки корневому элементу `LinearLayout` присвойте идентификатор `toast_layout`. Определите два дочерних элемента, `ImageView` и `TextView`, как показано в листинге 9.3.

Листинг 9.3. Файл компоновки `custom_layout.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/toast_layout"  
    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>
```

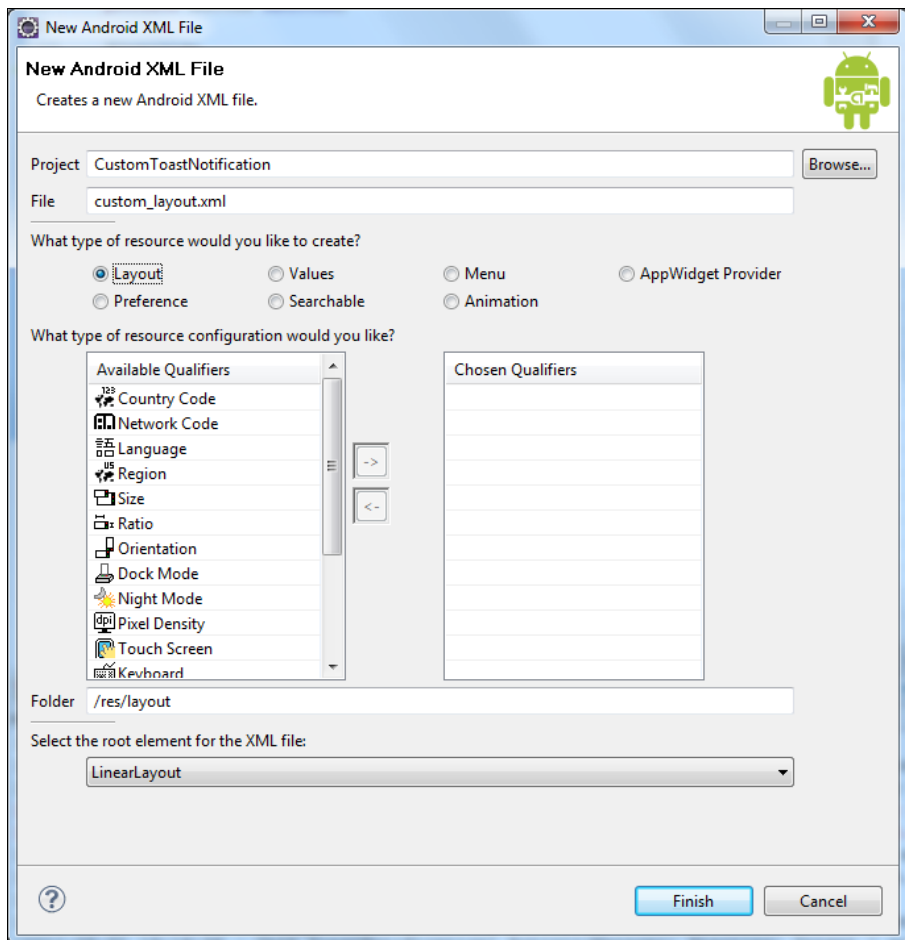


Рис. 9.2. Окно создания нового XML-файла

Полный код класса `CustomToastActivity` для вызова и работы с уведомлением представлен в листинге 9.4.

Листинг 9.4. Файл класса окна приложения `CustomToastActivity.java`

```
package com.samples.ui.customtoast;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
```



```
public class CustomToastNotificationActivity extends Activity
    implements View.OnClickListener {

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

    public void onClick(View view) {
        LayoutInflater inflater = getLayoutInflater();
        View layout = inflater.inflate(R.layout.custom_layout,
            (ViewGroup)findViewById(R.id.toast_layout));

        final ImageView image =
            (ImageView)layout.findViewById(R.id.image);
        image.setImageResource(R.drawable.android3d);

        final 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();
    }
}
```

Запустите проект на выполнение. При нажатии кнопки вызова должно появиться на несколько секунд окно уведомления с текстовым сообщением и значком (рис. 9.3).

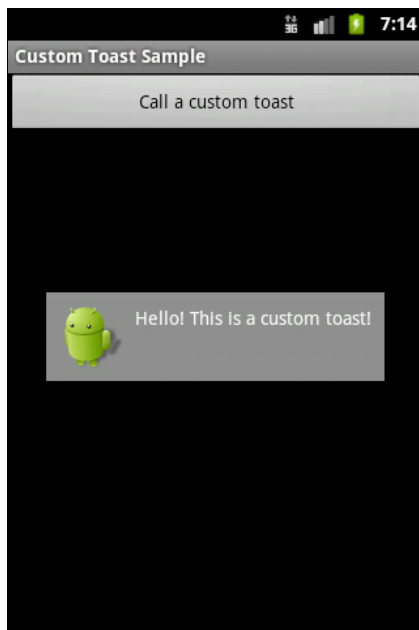


Рис. 9.3. Пример создания собственного всплывающего уведомления

Резюме

В этой главе мы рассмотрели создание уведомлений. Уведомления широко используются в приложениях для информирования пользователя о наступивших событиях в системе или в приложении. В случае если дизайн окна стандартного всплывающего сообщения вам не подходит, вы всегда можете разработать собственное, более информативное окно сообщения для своего приложения.

В следующей главе мы изучим создание и вызов диалоговых окон для взаимодействия пользователя с приложением.



ГЛАВА 10

Диалоговые окна

Диалог — это маленькое окно, которое появляется перед текущим Activity. Основной Activity при этом теряет фокус, и диалоговое окно принимает на себя все пользовательское взаимодействие. Диалоги обычно используются для уведомлений и коротких действий, которые непосредственно касаются приложения, а также для индикации прогресса выполнения длительных задач.

Типы диалогов

Система Android поддерживает следующие типы диалоговых окон:

- ☐ AlertDialog — диалог с кнопками, списком, флажками или переключателями;
- ☐ ProgressDialog — диалог с индикатором прогресса;
- ☐ DatePickerDialog — диалог выбора даты;
- ☐ TimePickerDialog — диалог выбора времени.

Иерархия классов диалоговых окон представлена на рис. 10.1.

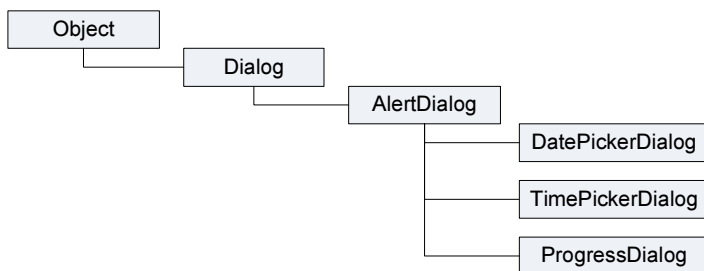


Рис. 10.1. Иерархия классов диалоговых окон

Класс `Dialog` является базовым для всех классов диалоговых окон. Поскольку `ProgressDialog`, `TimePickerDialog` и `DatePickerDialog` — расширения класса `AlertDialog`, они также могут иметь командные кнопки для открытия и закрытия диалогового окна.

Создание диалоговых окон

Диалоговое окно всегда создается и отображается как часть находящегося в фокусе Activity. Диалоги обычно создают внутри метода обратного вызова `onCreateDialog()`, который необходимо реализовать в коде Activity. При этом система Android автоматически управляет состоянием диалога (или нескольких диалогов) и прикрепляет их к текущему Activity, фактически делая его владельцем каждого диалога.

ОБРАТИТЕ ВНИМАНИЕ

Можно создавать диалог без `onCreateDialog()`, например в обработчике нажатия кнопки вызова диалога, но тогда он не будет присоединен к текущему Activity. Чтобы прикрепить его к Activity, необходимо вызвать метод `setOwnerActivity()`, передав ему в качестве параметра текущий Activity.

Для отображения диалогового окна необходимо вызвать метод `showDialog()` и передать ему в качестве параметра идентификатор диалога (константу, которую надо объявить в коде программы), который вы хотите отобразить. Например:

```
private static final int IDD_EXIT = 0;
...
showDialog(IDD_EXIT);
```

Когда диалог вызывается впервые, система Android вызывает `onCreateDialog()` из открытого Activity. В `onCreateDialog()` передается тот же самый идентификатор, который передавался в `showDialog()`. После того как вы создали диалог, вы возвращаете объект `Dialog` в конце метода.

Если в Activity должны вызываться несколько различных диалоговых окон, сначала необходимо определить целочисленный идентификатор для каждого диалога, например:

```
private static final int IDD_ALERT = 0;
private static final int IDD_EXIT = 1;
```

Эти идентификаторы потом можно использовать в вызове метода `showDialog()` и в обработчике события `onCreateDialog()` в операторе `switch`:

```
protected Dialog onCreateDialog(int id) {
    Dialog dialog;
    switch(id) {
        case IDD_ALERT:
            ...
            break;
        case IDD_EXIT:
            ...
            break;
        default:
            dialog = null;
    }
    return dialog;
}
```

Внутри оператора `switch` описывается сама процедура создания диалоговых окон, которая различна для каждого типа диалоговых окон и будет описана в следующих разделах этой главы.

Перед отображением диалогового окна Android вызывает дополнительный метод обратного вызова `onPrepareDialog(int, Dialog)`. Если требуется перед каждым вызовом диалогового окна изменять его свойства (например, текстовое сообщение или количество кнопок), это можно реализовать внутри этого метода. В этот метод передают идентификатор диалога и сам объект `Dialog`, который был создан в методе обратного вызова `onCreateDialog()`.

AlertDialog

Диалог `AlertDialog` — расширение класса `Dialog`. Он используется при построении большинства диалоговых окон. В этих диалогах доступна для использования любая функциональность из нижеперечисленных:

- ☐ заголовок;
- ☐ текстовое сообщение;
- ☐ одна, две или три кнопки;
- ☐ список;
- ☐ флажки;
- ☐ переключатели.

AlertDialog с кнопками

Для создания `AlertDialog` с кнопками используется группа методов `set...Button()` класса `AlertDialog.Builder`:

- ☐ `setPositiveButton();`
- ☐ `setNegativeButton();`
- ☐ `setNeutralButton().`

Для создания диалогового окна сначала надо создать объект класса `Builder`, передав в качестве параметра контекст приложения:

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

Затем, используя методы класса `Builder`, задать для создаваемого диалога необходимые свойства, например текстовое сообщение в окне методом `setMessage()`:

```
builder.setMessage("Are you sure you want to exit?");
```

После задания свойств диалога определяют командные кнопки диалога и обработку событий на них. В `AlertDialog` можно добавить только по одной кнопке каждого типа: `Positive`, `Negative` и `Neutral`, т. е. максимально возможное количество кнопок в диалоге — три.

Для каждой кнопки используется один из методов `set...Button()`, которые принимают в качестве параметров надпись для кнопки, и интерфейс `DialogInterface.OnClickListener`, который определяет действие, когда пользователь нажимает кнопку. Например, для создания диалога с кнопками **Yes** и **No** код будет выглядеть приблизительно так:

```
builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Закрываем текущий Activity
        AlertDialogButtonActivity.this.finish();
    }
});
builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int id) {
        // Закрываем диалог и возвращаемся к текущему Activity
        dialog.cancel();
    }
});
```

Чтобы пользователь не мог закрыть диалог клавишей <Back> на клавиатуре мобильного устройства, вызывается метод `setCancelable()`:

```
builder.setCancelable(false);
```

И наконец, отображаем диалог:

```
AlertDialog alert = builder.create();
```

Теперь попробуем создать приложение, в котором будет вызываться `AlertDialog`. Для этого создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `AlertDialogButtonApp`;
- ☐ **Application name** — `AlertDialog with Buttons`;
- ☐ **Package name** — `com.samples.ui.alertdialogbutton`;
- ☐ **Create Activity** — `AlertDialogButtonActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch10_AlertDialogButtons`.

Откройте файл компоновки и создайте структуру компоновки с единственной кнопкой для вызова диалога подобно листингу 10.1.

Листинг 10.1. Файл компоновки main.xml

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

    <Button
        android:id="@+id/button"
        android:text="Call AlertDialog with Buttons"
        android:layout_width="fill_parent"
        android:onClick="onClick"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

В классе `AlertDialogButtonActivity` реализуйте последовательность создания диалога, приведенную ранее. Полный код класса представлен в листинге 10.2.

Листинг 10.2. Файл класса окна приложения `AlertDialogButtonActivity.java`

```
package com.samples.ui.alertdialogbutton;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;

public class AlertDialogButtonActivity extends Activity
    implements View.OnClickListener {
    // Идентификатор диалогового окна
    private final int IDD_EXIT = 0;

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

    @Override
    public void onClick(View v) {
        // Вызываем диалог
        showDialog(IDD_EXIT);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
            case IDD_EXIT:
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setMessage("Are you sure you want to exit?");

                // Создаем кнопку "Yes" и обработчик события
                builder.setPositiveButton("Yes",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            AlertDialogButtonActivity.this.finish();
                        }
                    }
                );

                // Создаем кнопку "No" и обработчик события
                builder.setNegativeButton("No",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
```

```

        dialog.cancel();
    }

    });
    builder.setCancelable(false);
    return builder.create();
default:
    return null;
}
}
}

```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно диалога `AlertDialog` с кнопками **Yes** и **No**. При закрытии диалога кнопкой **Yes** приложение закончит работу. Внешний вид приложения показан на рис. 10.2.

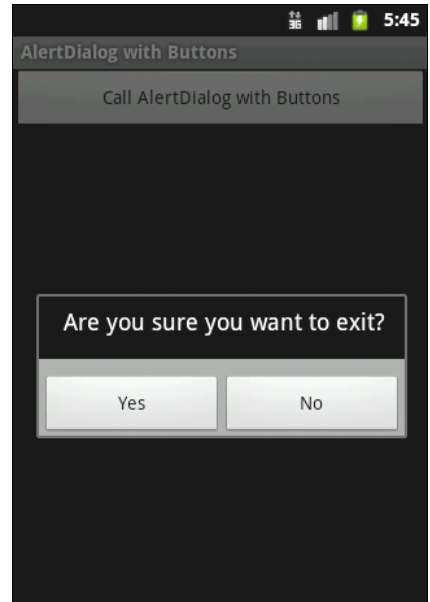


Рис. 10.2. Диалог `AlertDialog` с кнопками **Yes** и **No**

Добавление в *AlertDialog* значка и заголовка

Диалоговое окно при желании можно усовершенствовать и сделать более привлекательным, добавив значок и заголовок для диалогового окна. Для добавления заголовка используется метод `setTitle()` со строковым параметром, представляющим надпись на заголовке. Для добавления значка применяется метод `setIcon()`, в который надо передать идентификатор графического ресурса.

Добавьте в наше приложение (см. листинг 10.2) вызов этих методов, используя для заголовка название приложения, а для значка — значок приложения, как представлено в листинге 10.3.

Листинг 10.3. Дополнение в методе `onCreateDialog()` класса `AlertDialogButtonActivity`

```

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?");
builder.setTitle(R.string.app_name);
builder.setIcon(R.drawable.icon);

```


Внешний вид диалоговых окон с заголовком и заголовком со значком представлен на рис. 10.3. К диалогу будет добавлена дополнительная область для заголовка, размер которой установит система Android, при этом поле будет несколько больше при наличии значка.

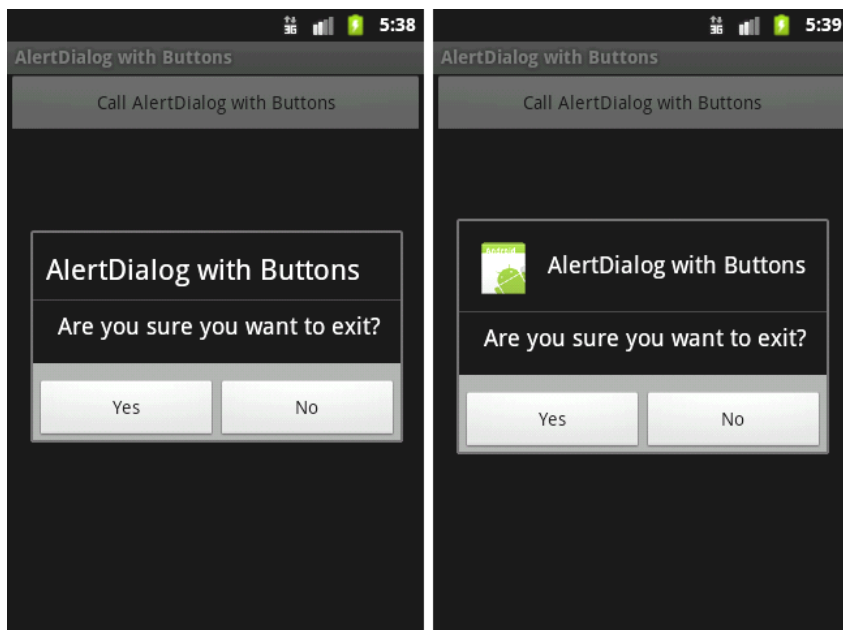


Рис. 10.3. Диалоговые окна с заголовком и значком

Такое гибкое построение диалоговых окон в Android позволяет при небольшом разрешении экрана создавать окно оптимального размера для экономии места на экране мобильного устройства.

AlertDialog со списком

Чтобы создавать AlertDialog со списком выбираемых пунктов, необходимо использовать метод `setItems()`, параметрами которого является массив данных для отображения в списке диалога и интерфейс `DialogInterface.OnClickListener`, который определяет действие, когда пользователь выбирает элемент списка, например:

```
CharSequence[] colors = {"Red", "Green", "Blue"};
...
builder.setItems(colors, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        // Обрабатываем событие выбора элемента списка
        Toast.makeText(getApplicationContext(),
            "Color: " + mColors[item], Toast.LENGTH_SHORT).show();
    }
});
```

Теперь реализуем вызов `AlertDialog` со списком в учебном приложении. Для этого создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `AlertDialogList`;
- ❑ **Application name** — `AlertDialog with List`;
- ❑ **Package name** — `com.samples.ui.alertdialoglist`;
- ❑ **Create Activity** — `AlertDialogListActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch10_AlertDialogList`.

Файл компоновки `main.xml` сделайте аналогично листингу 10.1. В классе `AlertDialogListActivity` реализуйте последовательность создания диалога со списком, приведенную ранее. Полный код класса представлен в листинге 10.4.

Листинг 10.4. Файл класса окна приложения `AlertDialogListActivity.java`

```
package com.samples.ui.alertdialoglist;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class AlertDialogListActivity extends Activity
    implements View.OnClickListener {
    // Идентификатор диалогового окна
    private final int IDD_COLOR = 0;
    // Массив данных для отображения в списке
    private final CharSequence[] colors = {"Red", "Green", "Blue"};

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

    @Override
    public void onClick(View v) {
        showDialog(IDD_COLOR);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
```

```
switch (id) {
case IDD_COLOR:
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Pick a color");

    builder.setItems(colors,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int item) {
                Toast.makeText(getApplicationContext(), "Color: " +
                    colors[item], Toast.LENGTH_SHORT).show();
                dialog.cancel();
            }
        });

    builder.setCancelable(false);
    return builder.create();
default:
    return null;
}
}
```

Запустите проект на выполнение. Если нажать кнопку вызова диалога `AlertDialog`, должно появиться окно со списком из трех пунктов для выбора цвета (рис. 10.4). При выборе одного из пунктов меню появится всплывающее уведомление, показывающее выбранный цвет.

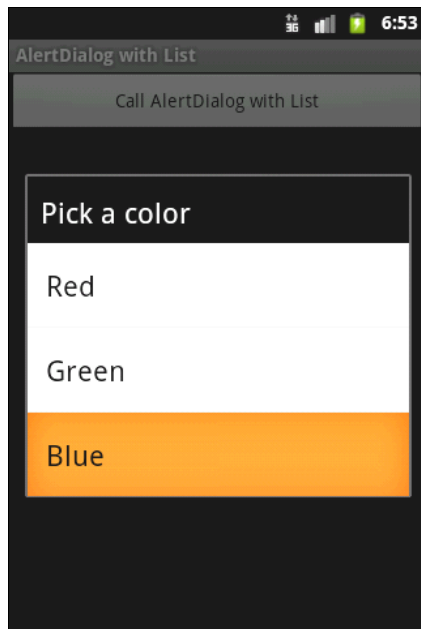


Рис. 10.4. Диалог `AlertDialog` со списком

AlertDialog с переключателями

Для создания диалогового окна с переключателями применяется метод `setSingleChoiceItems()`. Если диалоговое окно создается внутри `onCreateDialog()`, система Android управляет состоянием списка с переключателями. Пока главный Activity,

из которого был вызван диалог, остается незакрытым, диалоговое окно при последующих вызовах запоминает ранее выбранные пункты.

Создание `AlertDialog` с переключателями похоже на создание диалога со списком, только вместо метода `setItems()` вызывается метод `setSingleChoiceItems()`:

```
CharSequence[] colors = {"Red", "Green", "Blue"};
...
builder.setSingleChoiceItems(colors, 0,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int item) {
            Toast.makeText(getApplicationContext(), "Color: " + mColors[item],
                Toast.LENGTH_SHORT).show();
        }
    });
```

Первый параметр в методе `setSingleChoiceItems()` — массив значений для переключателей, второй параметр — целочисленное значение индекса переключателя, который будет включен по умолчанию при вызове диалога. Если требуется по умолчанию установить все переключатели в выключенное состояние, необходимо установить значение второго параметра в `-1`.

ОБРАТИТЕ ВНИМАНИЕ

При нажатии переключателя диалог закрываться не будет. В отличие от диалога со списком, который закрывается после выбора элемента списка, диалог с переключателями требует добавления командных кнопок для управления диалогом. Диалог закрывается только командными кнопками.

Реализуем теперь приложение с вызовом диалогового окна с переключателями в среде Eclipse. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `AlertDialogRadioButtons`;
- ☐ **Application name** — `AlertDialog with RadioButtons`;
- ☐ **Package name** — `com.samples.ui.alertdialogradiobuttons`;
- ☐ **Create Activity** — `AlertDialogRadioButtonsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch10_AlertDialogRadioButtons`.

Файл компоновки `main.xml` сделайте аналогично листингу 10.1. В классе `AlertDialogRadioButtonsActivity` реализуйте последовательность создания диалога с переключателями, как в показанном ранее примере. Полный код класса представлен в листинге 10.5.

Листинг 10.5. Файл класса окна приложения `AlertDialogRadioButtonsActivity.java`

```
package com.samples.ui.alertdialogradiobuttons;

import android.app.Activity;
import android.app.AlertDialog;
```

```
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class AlertDialogRadioButtonsActivity extends Activity
    implements View.OnClickListener {

    private final static int IDD_COLOR = 0;
    private final CharSequence[] colors = {"Red", "Green", "Blue"};
    private int result;

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

    @Override
    public void onClick(View v) {
        showDialog(IDD_COLOR);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
            case IDD_COLOR:
                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setTitle("Pick a color");
                builder.setSingleChoiceItems(colors, 0,
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int item) {
                            result = item;
                        }
                    });
                builder.setPositiveButton("OK",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            Toast.makeText(getApplicationContext(),
                                "Color: " + colors[result], Toast.LENGTH_SHORT).show();
                        }
                    });
                builder.setNegativeButton("Cancel",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            dialog.cancel();
                            Toast.makeText(getApplicationContext(),
                                "Dialog cancel", Toast.LENGTH_SHORT).show();
                        }
                    });
        }
    }
}
```

```

        builder.setCancelable(false);
        return builder.create();
    default:
        return null;
    }
}
}

```

Запустите проект на выполнение. При нажатии кнопки вызова диалога `AlertDialog` должно появиться окно с тремя переключателями для выбора цвета. При выборе одной из кнопок появится всплывающее уведомление, показывающее выбранный цвет (рис. 10.5).

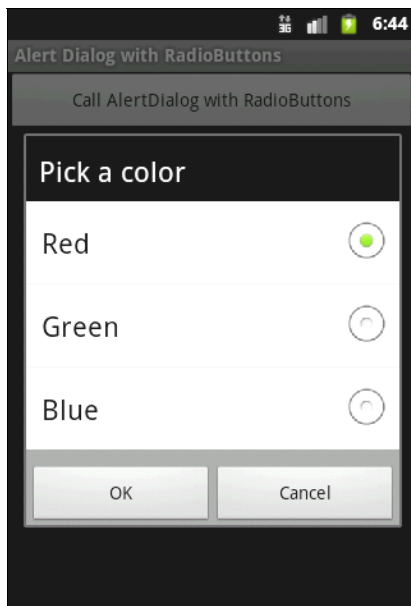


Рис. 10.5. Диалог `AlertDialog` с переключателями

***AlertDialog* с флажками**

При создании диалогового окна с переключателями применяется метод `setSingleChoiceItems()`. Если диалоговое окно создается внутри `onCreateDialog()`, система Android управляет состоянием списка. Пока текущий `Activity` активен, диалоговое окно при последующих вызовах запоминает ранее выбранные пункты.

Создание диалогового окна с флажками очень похоже на создание диалога с переключателями, только вместо метода `setSingleChoiceItems()` вызывается метод `setMultiChoiceItems()`:

```

CharSequence[] colors = {"Red", "Green", "Blue"};
final boolean[] mCheckedItems = {true, false, true};
...
builder.setMultiChoiceItems(colors, checkedItems,
    new DialogInterface.OnMultiChoiceClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which,
            boolean isChecked) {
            mCheckedItems[which] = isChecked;
        }
    });
}

```

Первый параметр в методе `setMultiChoiceItems()` — массив значений для списка с флажками, второй параметр — булевый массив состояний флажков списка по умолчанию при вызове диалога.

ОБРАТИТЕ ВНИМАНИЕ

Так же, как и для диалога с переключателями, для диалога с флажками добавляют командные кнопки для его закрытия.

Для примера приложения с вызовом диалогового окна с флажками создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `AlertDialogCheckBoxesApp`;
- ☐ **Application name** — `AlertDialog with CheckBoxes`;
- ☐ **Package name** — `com.samples.ui.alertdialogcheckboxes`;
- ☐ **Create Activity** — `AlertDialogCheckBoxesActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch10_AlertDialogCheckBoxes`.

Файл компоновки `main.xml` сделайте аналогично листингу 10.1. В классе `AlertDialogCheckBoxesActivity` реализуйте последовательность создания диалога с переключателями, приведенную ранее. Полный код класса, реализующего Activity, представлен в листинге 10.6.

Листинг 10.6. Файл класса окна приложения `AlertDialogCheckBoxesActivity.java`

```
package com.samples.ui.alertdialogcheckboxes;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class AlertDialogCheckBoxesActivity extends Activity
    implements View.OnClickListener {
    private final static int IDD_COLOR = 0;
    // Массив значений для списка
    private final CharSequence[] colors = {"Red", "Green", "Blue"};
    // Булевый массив состояний флажков списка по умолчанию
    private final boolean[] checkedItems = {true, false, true};

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

```

@Override
public void onClick(View v) {
    showDialog(IDD_COLOR);
}

@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case IDD_COLOR:
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Pick a color");

            builder.setMultiChoiceItems(colors, checkedItems,
                new DialogInterface.OnMultiChoiceClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which,
                        boolean isChecked) {
                        checkedItems[which] = isChecked;
                    }
                });

            builder.setPositiveButton("OK",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        StringBuilder state = new StringBuilder();
                        for (int i = 0; i < colors.length; i++) {
                            state.append("Color: " + colors[i] + ", state: ");
                            if (checkedItems[i])
                                state.append("checked\n");
                            else
                                state.append("unchecked\n");
                        }
                        Toast.makeText(getApplicationContext(),
                            state.toString(), Toast.LENGTH_LONG).show();
                    }
                });

            builder.setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        dialog.cancel();
                        Toast.makeText(getApplicationContext(), "Dialog cancel",
                            Toast.LENGTH_LONG).show();
                    }
                });

            builder.setCancelable(false);
            return builder.create();

```



```
default:
    return null;
}
}
```

Запустите проект на выполнение. При нажатии кнопки вызова диалога `AlertDialog` должно появиться окно со списком из трех флажков для выбора цвета (рис. 10.6). При закрытии диалогового окна кнопкой **ОК** появится всплывающее уведомление, показывающее состояние флажков выбора цвета.

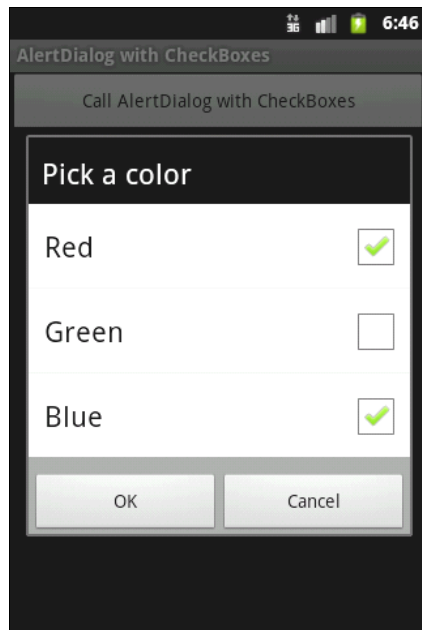


Рис. 10.6. Диалог `AlertDialog` с флажками

ProgressDialog

`ProgressDialog` — подкласс `AlertDialog`, который представляет собой диалоговое окно с индикатором прогресса. В диалог также можно добавить управляющие кнопки, например для отмены выполняемой задачи.

Для создания диалога с индикатором прогресса необходимо инициализировать объект `ProgressDialog` вызовом конструктора класса `ProgressDialog(Context)`, передав в качестве параметра контекст текущего `Activity`:

```
ProgressDialog progressDialog = new ProgressDialog(Activity_Name.this);
```

Далее надо установить требуемые свойства диалогового окна, например:

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

Обычно выполнение длительных задач происходит в другом потоке, т. е. в нашем приложении необходимо создать второй поток и сообщать о прогрессе его выполнения назад, в основной поток `Activity` через объект `Handler` (создание потока и объект `Handler` мы использовали ранее в главе 8):

```

Handler handler = new Handler() {
    public void handleMessage(Message msg) {
        ...
        progressDialog.setProgress(total);
        ...
    }
};

```

В качестве примера приложения, в котором будет вызываться диалоговое окно с индикатором прогресса, создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — ProgressDialogApp;
- ☐ **Application name** — ProgressDialog Sample;
- ☐ **Package name** — com.samples.ui.progressdialog;
- ☐ **Create Activity** — ProgressDialogActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch10_ProgressDialog.

Структура файла компоновки будет аналогична листингу 10.1. В классе ProgressDialogActivity реализуйте последовательность создания и задания свойств для диалога, приведенную ранее. Код для запуска и работы второго потока реализован в классе SecondThread, который создается и запускается в методе обратного вызова onCreateDialog() класса Activity.

Полный код класса ProgressDialogActivity, реализующего Activity для приложения, представлен в листинге 10.7. Код класса SecondThread для работы с потоком — в листинге 10.8.

Листинг 10.7. Файл класса окна приложения ProgressDialogActivity.java

```

package com.samples.ui.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.widget.Toast;

public class ProgressDialogActivity extends Activity
    implements View.OnClickListener {
    private static final int IDD_PROGRESS = 0;

    private ProgressThread thread;
    private ProgressDialog dialog;

```

```

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

@Override
public void onClick(View v) {
    showDialog(IDD_PROGRESS);
}

protected Dialog onCreateDialog(int id) {
    switch(id) {
        case IDD_PROGRESS:
            dialog = new ProgressDialog(ProgressDialogActivity.this);
            dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            dialog.setMessage("Loading...");
            thread = new ProgressThread(handler);
            thread.start();
            return dialog;
        default:
            return null;
    }
}

// Обработчик, который получает сообщения от потока
// и обновляет индикатор прогресса
final Handler handler = new Handler() {
    public void handleMessage(Message msg) {
        int total = msg.getData().getInt("total");
        dialog.setProgress(total);

        if (total >= 100){
            dismissDialog(IDD_PROGRESS);
            thread.setState(ProgressThread.STATE_DONE);
            Toast.makeText(getApplicationContext(), "Task is finished",
                Toast.LENGTH_SHORT).show();
        }
    }
};
}

```

Листинг 10.8. Файл класса окна приложения SecondThread.java

```

package com.samples.ui.progressdialog;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

```

```

public class ProgressThread extends Thread {
    // Константы, отображающие состояние индикатора,
    // используются в вызывающем Activity
    public final static int STATE_DONE = 0;
    public final static int STATE_RUNNING = 1;

    private Handler handler;
    private int stateRun;
    private int total;

    ProgressThread(Handler h) {
        handler = h;
    }

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

    public void setState(int state) {
        stateRun = state;
    }
}

```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться диалоговое окно с индикатором прогресса и надписями, отображающими степень завершения задачи (рис. 10.7).

Метод `setProgressStyle()` устанавливает стиль диалога. Стили задаются константами, определенными в классе `ProgressDialog`:

- ☐ `STYLE_HORIZONTAL`;
- ☐ `STYLE_SPINNER`.

Попробуйте поменять стиль шкалы индикатора прогресса в приложении. Измените в листинге 10.6 строку с вызовом метода `setProgressStyle()`:

```
progressDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
```

Перекомпилируйте проект и снова запустите его на выполнение. При нажатии кнопки вызова диалога `ProgressDialog` должно появиться окно, но уже не с линейным, а с круговым индикатором прогресса (рис. 10.8).

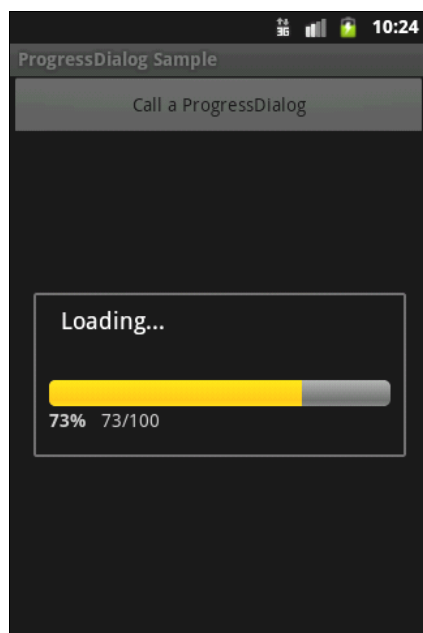


Рис. 10.7. Диалог `ProgressDialog` со стилем `STYLE_HORIZONTAL`

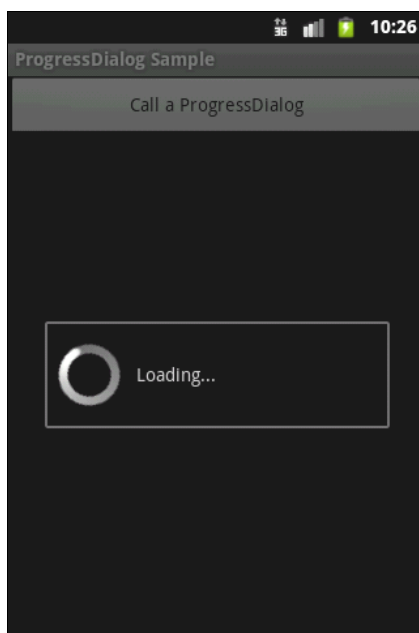


Рис. 10.8. Диалог `ProgressDialog` со стилем `STYLE_SPINNER`

DatePickerDialog

`DatePickerDialog` предназначен для выбора даты пользователем. Обычно перед созданием диалога выбора даты надо получить текущий год, месяц и день из системы. Это можно сделать, создав экземпляр класса `Calendar` и записав дату через метод `get()` класса `Calendar` в переменные, созданные в нашем классе:

```
Calendar c = Calendar.getInstance();
mYear = c.get(Calendar.YEAR);
mMonth = c.get(Calendar.MONTH);
mDay = c.get(Calendar.DAY_OF_MONTH);
```

Затем в `onCreateDialog()` надо создать объект `DatePickerDialog` вызовом конструктора класса:

```
DatePickerDialog dialog = new DatePickerDialog(
    this, mDateSetListener, mYear, mMonth, mDay);
```

Первый параметр, передаваемый в конструктор, — это контекст текущего `Activity`, второй — имя метода обратного вызова для обработки события установки даты, остальные параметры содержат дату, которую отобразит запускаемый диалог.

Имя метода обратного вызова для обработки события установки даты — это реализация вложенного интерфейса `DatePickerDialog.OnDateSetListener`, которая, например, может модифицировать передаваемые переменные, хранящие дату:

```
private DatePickerDialog.OnDateSetListener mDateSetListener =
    new DatePickerDialog.OnDateSetListener() {
        public void onDateSet(DatePicker view, int year,
            int monthOfYear, int dayOfMonth) {
            mYear = year;
            mMonth = monthOfYear;
            mDay = dayOfMonth;
        }
    };
```

Вызов диалога производится обычным способом — через метод `showDialog(int dialogId)`.

В качестве примера приложения с вызовом диалогового окна установки даты создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `DatePickerDialogApp`;
- ☐ **Application name** — `DatePickerDialog Sample`;
- ☐ **Package name** — `com.samples.ui.datepickerdialog`;
- ☐ **Create Activity** — `DatePickerDialogActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch10_DatePickerDialog`.

Откройте файл компоновки и создайте структуру компоновки с текстовым полем и кнопкой вызова подобно листингу 10.9.

Листинг 10.9. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical" android:gravity="center">

    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="28px"
        android:textStyle="bold"
        android:text="" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:onClick="onClick"  
        android:text="Change the date"/>
```

```
</LinearLayout>
```

Полный код класса Activity, реализующий работу с DatePickerDialog, приведен в листинге 10.10.

Листинг 10.10. Файл класса окна приложения DatePickerDialogActivity.java

```
package com.samples.ui.datepickerdialog;  
  
import java.util.Calendar;  
  
import android.app.Activity;  
import android.app.DatePickerDialog;  
import android.app.Dialog;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.DatePicker;  
import android.widget.TextView;  
  
public class DatePickerDialogActivity extends Activity  
    implements View.OnClickListener {  
    private TextView text;  
  
    private int years;  
    private int months;  
    private int days;  
  
    static final int IDD_DATE = 0;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        text = (TextView)findViewById(R.id.text);  
  
        final Calendar c = Calendar.getInstance();  
        years = c.get(Calendar.YEAR);  
        months = c.get(Calendar.MONTH);  
        days = c.get(Calendar.DAY_OF_MONTH);  
  
        updateDisplay();  
    }  
  
    @Override  
    public void onClick(View v) {  
        showDialog(IDD_DATE);  
    }  
}
```

```

@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
        case IDD_DATE:
            return new DatePickerDialog(this,
                                     mDateSetListener,
                                     years, months, days);
    }
    return null;
}

private void updateDisplay() {
    text.setText(
        new StringBuilder()
            .append(months + 1).append("-")
            .append(days).append("-")
            .append(years).append(" ")
    );
}

private DatePickerDialog.OnDateSetListener mDateSetListener =
    new DatePickerDialog.OnDateSetListener() {

        public void onDateSet(DatePicker view, int year,
                               int monthOfYear, int dayOfMonth) {
            years = year;
            months = monthOfYear;
            days = dayOfMonth;
            updateDisplay();
        }
    };
}

```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно диалога `DatePickerDialog`, в котором пользователь может установить дату. При закрытии диалога возвращаемый результат запишется в текстовое поле в верхней части экрана приложения.

Внешний вид приложения с вызванным диалогом установки даты показан на рис. 10.9.

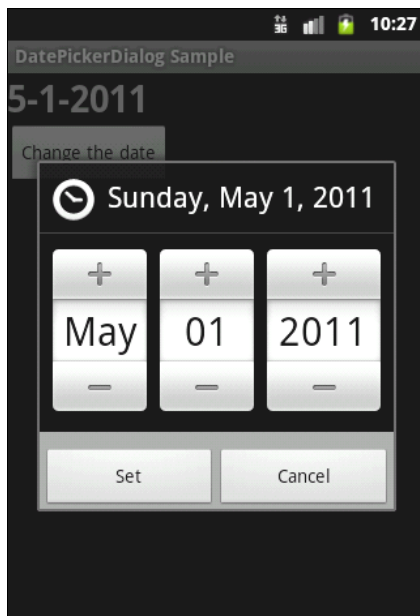


Рис. 10.9. Диалог `DatePickerDialog`

TimePickerDialog

`TimePickerDialog` предназначен для выбора даты пользователем. В целом процедура создания диалога выбора времени аналогична созданию `DatePickerDialog` с небольшими отличиями в деталях.

Если необходимо прочитать текущее системное время, также используется класс `Calendar`, например:

```
Calendar c = Calendar.getInstance();
mHour = c.get(Calendar.HOUR_OF_DAY);
mMinute = c.get(Calendar.MINUTE);
```

Затем в `onCreateDialog()` надо создать объект `TimePickerDialog` вызовом конструктора класса:

```
TimePickerDialog dialog = new TimePickerDialog(
    this, timeSetListener, mHour, mMinute, false);
```

В последнем параметре конструктора указывается формат отображения времени 12-часовой (AM/PM) — `false` или 24-часовой — `true`.

Имя метода обратного вызова для обработки события установки даты — это реализация вложенного интерфейса `TimePickerDialog.OnTimeSetListener`, например:

```
private TimePickerDialog.OnTimeSetListener mTimeSetListener =
    new TimePickerDialog.OnTimeSetListener() {
        public void onTimeSet(TimePicker view,
            int hourOfDay, int minute) {
            mHour = hourOfDay;
            mMinute = minute;
        }
    };
```

В качестве примера приложения с вызовом диалога установки времени создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `TimePickerDialogActivityApp`;
- ☐ **Application name** — `TimePickerDialog`;
- ☐ **Package name** — `com.samples.ui.timepickerdialog`;
- ☐ **Create Activity** — `TimePickerDialogActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch10_TimePickerDialog`.

Структура файла компоновки `main.xml` аналогична листингу 10.9 для `DatePickerDialog`. Код класса `Activity` `TimePickerDialogActivity` приведен в листинге 10.11.

Листинг 10.11. Файл класса окна приложения `TimePickerDialogActivity.java`

```
package com.samples.ui.timepickerdialog;

import java.util.Calendar;
```

```
import android.app.Activity;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.TimePicker;

public class TimePickerDialogActivity extends Activity
    implements View.OnClickListener {

    private TextView text;

    private int hours;
    private int min;

    private static final int IDD_TIME = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        text = (TextView)findViewById(R.id.text);

        final Calendar calendar = Calendar.getInstance();
        hours = calendar.get(Calendar.HOUR_OF_DAY);
        min = calendar.get(Calendar.MINUTE);

        updateDisplay();
    }

    @Override
    public void onClick(View v) {
        showDialog(IDD_TIME);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
            case IDD_TIME:
                return new TimePickerDialog(
                    this, mTimeSetListener, hours, min, false);
            }
        return null;
    }

    private void updateDisplay() {
        text.setText(
```

```

        new StringBuilder()
            .append(pad(hours)).append(":")
            .append(pad(min));
    }

    private TimePickerDialog.OnTimeSetListener mTimeSetListener =
        new TimePickerDialog.OnTimeSetListener() {
            public void onTimeSet(
                TimePicker view, int hourOfDay, int minute) {
                hours = hourOfDay;
                min = minute;
                updateDisplay();
            }
        };

    private static String pad(int c) {
        if (c >= 10) {
            return String.valueOf(c);
        }
        else {
            return "0" + String.valueOf(c);
        }
    }
}

```

Запустите проект на выполнение. При нажатии кнопки вызова диалога должно появиться окно `TimePickerDialog`, в котором пользователь может установить время. При закрытии диалога возвращаемый результат запишется в текстовое поле в верхней части экрана приложения (рис. 10.10).

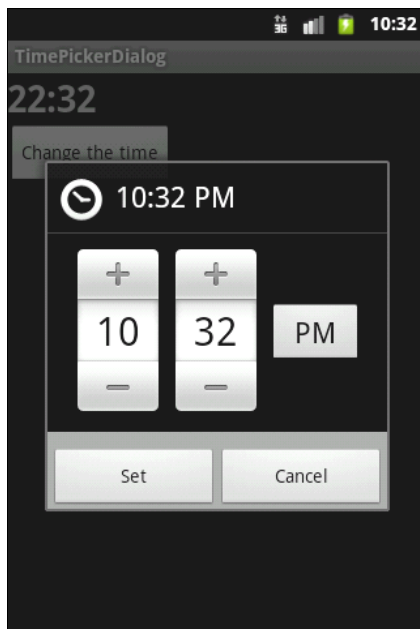


Рис. 10.10. Диалог `TimePickerDialog`

Создание собственных диалогов

Если есть необходимость в создании оригинального дизайна для диалога, можно создать собственную компоновку для диалогового окна. После создания компоновки необходимо передать корневой объект компоновки в код создания диалога.

Чтобы получить корневой объект компоновки, используется класс `LayoutInflater`. Этот класс нужен для преобразования XML-компоновки в соответствующие объекты `View` в коде программы.

Сначала необходимо инициализировать объект `LayoutInflater` вызовом метода `getLayoutInflater()`, затем получить корневое представление методом `inflate(int, ViewGroup)`, где первый параметр — идентификатор ресурса схемы размещения, второй — идентификатор корневого представления компоновки:

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

Получив корневое представление, можно методом `findViewById()` инициализировать все дочерние представления в компоновке и задать для них информационное наполнение. Например, если в компоновке определены виджеты `TextView` и `ImageView`, код может выглядеть так:

```
TextView text = (TextView) layout.findViewById(R.id.text);  
text.setText("Are you sure you want to exit?");  
  
ImageView image = (ImageView) layout.findViewById(R.id.image);  
image.setImageResource(R.drawable.android3d);
```

Далее создается объект `AlertDialog.Builder` и методом `setView()` для него устанавливается полученная ранее компоновка:

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

Остальная инициализация свойств диалога и работа с ним в коде программы аналогична работе со стандартным `AlertDialog`.

В качестве примера приложения с собственным диалоговым окном создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `CustomDialogApp`;
- ☐ **Application name** — `Custom Dialog Sample`;
- ☐ **Package name** — `com.samples.ui.customdialog`;
- ☐ **Create Activity** — `CustomDialogActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch10_CustomDialog`.

Структура файла компоновки `main.xml` для главного окна приложения будет аналогична листингу 10.1. Создайте также отдельный файл компоновки `custom_layout.xml` для диалогового окна, как показано в листинге 10.12.

Листинг 10.12. Файл компоновки custom_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toast_layout"
    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>
```

В классе `CustomDialogActivity` реализуйте создание и вызов диалогового окна с собственной компоновкой, как было показано ранее в этом разделе. Полный код класса `CustomDialogActivity` показан в листинге 10.13.

Листинг 10.13. Файл класса окна приложения CustomDialogActivity.java

```
package com.samples.ui.customdialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

public class CustomDialogActivity extends Activity
    implements View.OnClickListener {

    private final static int IDD_CUSTOM = 0;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

        setContentView(R.layout.main);
    }

    @Override
    public void onClick(View v) {
        showDialog(IDD_CUSTOM);
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        switch (id) {
            case IDD_CUSTOM:
                LayoutInflater inflater = getLayoutInflater();
                View layout = inflater.inflate(
                    R.layout.custom_layout,
                    (ViewGroup) findViewById(R.id.toast_layout));

                TextView text = (TextView) layout.findViewById(R.id.text);
                text.setText("Are you sure you want to exit?");
                ImageView image = (ImageView) layout.findViewById(R.id.image);
                image.setImageResource(R.drawable.android3d);

                AlertDialog.Builder builder = new AlertDialog.Builder(this);
                builder.setView(layout);
                builder.setMessage("This is a custom dialog!");

                builder.setPositiveButton("Yes",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            CustomDialogActivity.this.finish();
                        }
                    });

                builder.setNegativeButton("No",
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int id) {
                            dialog.cancel();
                        }
                    });

                builder.setCancelable(false);
                return builder.create();
            default:
                return null;
        }
    }
}

```

Запустите проект на выполнение. При нажатии кнопки вызова диалога `AlertDialog` должно появиться окно с кнопками **Yes** и **No** и с созданным нами нестандартным информационным наполнением. При закрытии диалога кнопкой **Yes** приложение закончит работу (рис. 10.11).

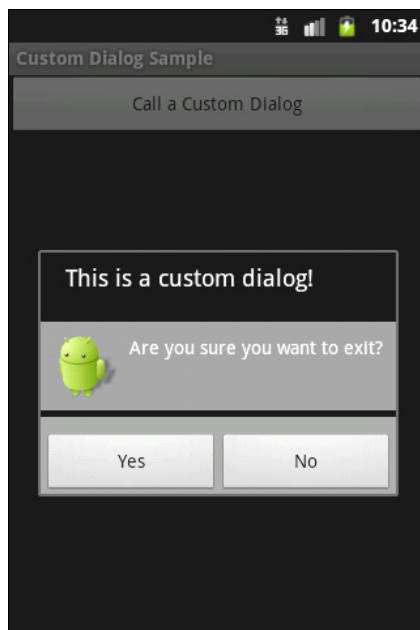


Рис. 10.11. Диалог с собственным дизайном

Резюме

В этой главе мы рассмотрели использование диалоговых окон в приложениях. Как вы убедились, система Android предоставляет в распоряжение разработчика множество вариантов диалогов самого разного типа, которые вы можете легко создавать и использовать при разработке собственных приложений.

В следующей главе мы рассмотрим создание меню, типы меню и варианты их использования в Android-приложениях.



ГЛАВА 11

Меню

Меню — важная часть любого приложения. Система Android предлагает достаточно простой интерфейс для создания стандартизированных прикладных меню и использования их в приложениях с разнообразной функциональностью.

Android предлагает три базовых типа прикладных меню.

- ❑ *Меню выбора опций* (Options Menu) — набор пунктов меню, прикрепляемый к Activity. Меню появляется внизу экрана при нажатии клавиши <MENU> на мобильном устройстве. Для меню выбора опций дополнительно существует еще две разновидности меню:
 - *меню со значками* (Icon Menu) — расширение меню выбора опций, добавляющее значки к тексту в пункты меню. Меню может содержать максимум шесть пунктов. Этот тип меню — единственный, который поддерживает значки;
 - *расширенное меню* (Expanded Menu) — вертикальный выпадающий список пунктов меню. Расширенное меню появляется при наличии более шести пунктов меню. При этом в меню выбора опций появляется дополнительный пункт **More**. Расширенное меню добавляется автоматически системой Android. При нажатии пункта **More** показывается расширенное меню со списком пунктов, которые не поместились в основной части меню выбора опций.
- ❑ *Контекстное меню* (Context Menu) — всплывающий список пунктов меню, который появляется при касании сенсорного экрана в течение двух и более секунд (событие long-press).
- ❑ *Подменю* (Submenu) — всплывающий список пунктов меню, который привязан к конкретному пункту в меню выбора опций или в контекстном меню. Пункты подменю не поддерживают вложенные подменю.

Меню выбора опций

Меню выбора опций — наиболее распространенный тип меню в приложениях. Меню, как уже говорилось ранее, открывается при нажатии на мобильном устройстве клавиши <MENU>.

Когда это меню открывается впервые, система Android вызывает метод `onOptionsItemSelected()`, передавая в качестве параметра объект `Menu`. Этот метод необходимо реализовать в классе `Activity`, где происходит вызов меню, и создать информаци-

онное наполнение для объекта `Menu`. Меню можно определить в XML-файле или использовать метод `add()` для последовательного присоединения каждого пункта меню, например:

```
// Сначала определяем идентификаторы для создаваемых пунктов меню
private static final int IDM_OPEN = 101;
private static final int IDM_SAVE = 102;
...
public boolean onCreateOptionsMenu(Menu menu) {
    // Добавляем пункты меню
    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
}
```

Метод `add()`, используемый в этом примере, принимает четыре параметра:

- ❑ идентификатор группы — позволяет связывать данный пункт меню с группой других пунктов этого меню (о группах меню будет рассказано далее в этой главе);
- ❑ идентификатор пункта для обработчика события выбора пункта меню (определяется в коде заранее);
- ❑ порядок расположения пункта в меню — позволяет определять позицию пункта в меню. По умолчанию (значение `Menu.NONE` или 0) пункты меню будут отображены в соответствии с последовательностью добавления в код;
- ❑ заголовок — текст пункта меню (он может также быть строковым ресурсом, если необходимо создавать локализованные приложения).

Этот метод возвращает объект `MenuItem`, который можно использовать для установки дополнительных свойств, например значка, "горячих" клавиш и других параметров настройки для этого пункта меню.

Метод `onCreateOptionsMenu()` вызывается системой только один раз — при создании меню. Если требуется обновлять меню каждый раз при его вызове из программы, необходимо определить в программе метод обратного вызова `onPrepareOptionsMenu()`.

При выборе пункта меню пользователем будет вызван метод `onOptionsItemSelected()`, который необходимо определить в классе, реализующем `Activity`. Этот метод обратного вызова передает в программу объект `MenuItem` — пункт меню, который был выбран пользователем. Идентифицировать выбранный пункт меню можно методом `getItemId()`, который возвращает целое число, являющееся идентификатором пункта меню, который был назначен ему в методе `add()` при создании меню в `onCreateOptionsMenu()`. После идентификации пункта меню можно написать код, реализующий обработку события выбора меню.

Обработчик события выбора пункта меню будет выглядеть примерно так:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case IDM_OPEN:
            ...
            return true;
    }
}
```

```
case IDM_SAVE:
    ...
    return true;
}
return false;
}
```

Для меню также возможно добавить "горячие" клавиши (или сочетания клавиш) для быстрого доступа, используя символы клавиатуры. Для добавления "горячих" клавиш в меню существует несколько методов:

- ❑ `setAlphabeticShortcut(char)` — добавляет символ;
- ❑ `setNumericShortcut(int)` — добавляет число;
- ❑ `setShortcut(char, int)` — добавляет комбинацию символа и числа.

Например:

```
setAlphabeticShortcut('q');
```

Теперь при открытии меню (или при удерживании клавиши <MENU>) нажатие клавиши <q> выберет этот пункт меню. Эта быстрая клавиша (или сочетание клавиш) будет показана как подсказка, отображающаяся ниже имени пункта меню.

Для примера приложения с меню создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `OptionsMenuApp`;
- ❑ **Application name** — `OptionsMenu Sample`;
- ❑ **Package name** — `com.samples.ui.optionsmenu`;
- ❑ **Create Activity** — `OptionsMenuActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch11_OptionsMenu`.

Откройте файл компоновки и создайте структуру подобно листингу 11.1.

Листинг 11.1. Файл компоновки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Press MENU button..."
        android:gravity="center"
        android:textStyle="bold"/>

</LinearLayout>
```

В классе `OptionsMenuActivity` реализуйте процедуру создания и обработки пользовательского взаимодействия с меню, приведенную ранее. Полный код класса `OptionsMenuActivity` приведен в листинге 11.2.

Листинг 11.2. Файл класса окна приложения `OptionsMenuActivity.java`

```
package com.samples.ui.optionsmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class OptionsMenuActivity extends Activity {
    // Идентификаторы для пунктов меню
    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EDIT = 103;
    public static final int IDM_HELP = 104;
    public static final int IDM_EXIT = 105;

    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.main);
    }

    // Создание меню
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Добавляем пункты меню
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setAlphabeticShortcut('o');
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setAlphabeticShortcut('s');
        menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit")
            .setAlphabeticShortcut('e');
        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help")
            .setAlphabeticShortcut('h');
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setAlphabeticShortcut('x');

        return(super.onCreateOptionsMenu(menu));
    }

    // Обработка события выбора пункта меню
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
```

```
switch (item.getItemId()) {
    case IDM_OPEN:
        message = "Open item selected";
        item.setChecked(true);
        break;
    case IDM_SAVE:
        message = "Save item selected";
        break;
    case IDM_HELP:
        message = "Help item selected";
        break;
    case IDM_EDIT:
        message = "Edit item selected";
        break;
    case IDM_EXIT:
        message = "Exit item selected";
        break;
    default:
        return false;
}

// Выводим уведомление о выбранном пункте меню
Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();

return true;
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора на экране должно появиться меню из пяти пунктов: **Open**, **Save**, **Edit**, **Help**, **Exit** (рис. 11.1). При выборе одного из пунктов меню будет появляться соответствующее всплывающее сообщение.

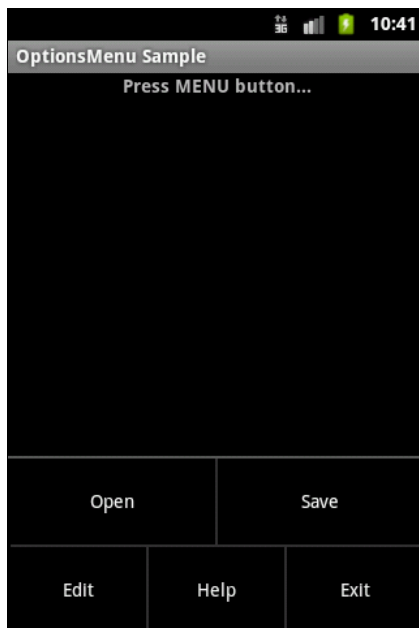


Рис. 11.1. Вызов меню в Activity

Меню со значками

В меню можно также отображать значки для каждого пункта. Значки добавляются методом `setIcon()` при создании меню. Например:

```
menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
    .setIcon(R.drawable.ic_menu_open);
menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
    .setIcon(R.drawable.ic_menu_save);
```

В качестве примера приложения, имеющего меню со значками, создайте в Eclipse новый проект и в диалоге **Create New Project** заполните поля:

- ☐ **Project name** — `OptionsIconMenu`;
- ☐ **Application name** — `OptionsMenu with icons`;
- ☐ **Package name** — `com.samples.ui.optionsiconmenu`;
- ☐ **Create Activity** — `OptionsIconMenuActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch11_OptionsIconMenu`.

Файл компоновки для меню со значками оставьте таким же, как в предыдущем примере (см. листинг 11.1). Примеры значков можно взять из прилагаемого к книге архива в каталоге `Resources/Menu_Icons/` — это файлы `ic_menu_open.png`, `ic_menu_save.png`, `ic_menu_edit.png`, `ic_menu_help.png` и `ic_menu_exit.png`.

Класс `OptionsIconMenuActivity`, реализующий меню со значками, представлен в листинге 11.3.

Листинг 11.3. Файл класса окна приложения `OptionsIconMenuActivity.java`

```
package com.samples.ui.optionsiconmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class OptionsIconMenuActivity extends Activity {
    // Идентификаторы для пунктов меню
    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EDIT = 103;
    public static final int IDM_HELP = 104;
    public static final int IDM_EXIT = 105;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setIcon(R.drawable.ic_menu_open);
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setIcon(R.drawable.ic_menu_save);
        menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit")
            .setIcon(R.drawable.ic_menu_edit);
        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help")
            .setIcon(R.drawable.ic_menu_help);
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setIcon(R.drawable.ic_menu_exit);

        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case IDM_OPEN:
                message = "Open item selected";
                break;
            case IDM_SAVE:
                message = "Save item selected";
                break;
            case IDM_HELP:
                message = "Help item selected";
                break;
            case IDM_EDIT:
                message = "Edit item selected";
                break;
            case IDM_EXIT:
                message = "Exit item selected";
                break;
            default:
                return false;
        }

        // Выводим уведомление о выбранном пункте меню
        Toast toast = Toast.makeText(
            this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();

        return true;
    }
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора должно появиться меню из пяти пунктов, такое же, как в предыдущем примере, но теперь уже со значками для каждого пункта меню (рис. 11.2).

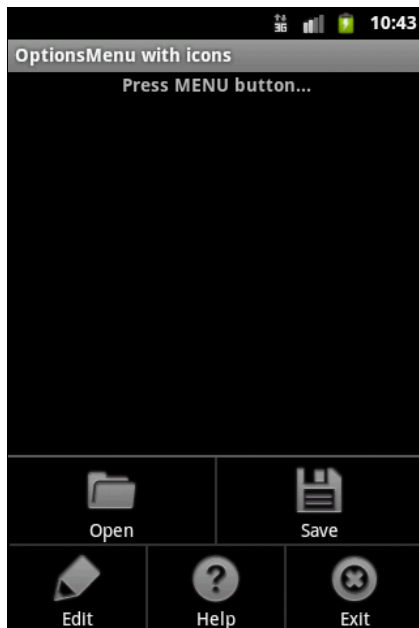


Рис. 11.2. Меню со значками

Расширенное меню

Расширенное меню, как уже было сказано, появляется при количестве пунктов меню больше шести. Это меню добавляется автоматически системой Android при отображении меню на экране.

Для примера приложения с расширенным меню создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — OptionsExpandedMenu;
- ☐ **Application name** — Options Expandable Menu Sample;
- ☐ **Package name** — com.samples.ui.optionsexpandedmenu;
- ☐ **Create Activity** — OptionsExpandedMenuActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch11_OptionsExpandedMenu.

Файл компоновки для расширенного меню оставьте таким же, как в предыдущем примере (см. листинг 11.1). Значки для пунктов меню можно взять в каталоге Resources/Menu_Icons/ прилагаемого к книге архива.

Для класса, реализующего Activity, возьмите код из предыдущего примера и добавьте в него три дополнительных идентификатора для пунктов меню:

```
private static final int IDM_FIND_REPLACE = 106;
private static final int IDM_FIND_NEXT = 107;
private static final int IDM_FIND_PREV = 108;
```


В методе `onCreateOptionsMenu()` добавьте новые пункты меню (теперь в меню будет восемь пунктов):

```
public boolean onCreateOptionsMenu(Menu menu) {  
    ...  
    menu.add(Menu.NONE, IDM_FIND_REPLACE, Menu.NONE, "Find/Replace");  
    menu.add(Menu.NONE, IDM_FIND_NEXT, Menu.NONE, "Find Next");  
    menu.add(Menu.NONE, IDM_FIND_PREV, Menu.NONE, "Find Previous");  
  
    return(super.onCreateOptionsMenu(menu));  
}
```

ОБРАТИТЕ ВНИМАНИЕ

Значки для дополнительных пунктов меню не добавляются, поскольку они все равно не будут отображаться в пунктах расширенного меню.

Добавьте также в метод `onOptionsItemSelected()` обработку событий выбора новых пунктов меню. Полный код класса `OptionsExpandedMenuActivity` приведен в листинге 11.4.

Листинг 11.4. Файл класса окна приложения `OptionsExpandedMenuActivity.java`

```
package com.samples.ui.optionsexpandedmenu;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.view.Gravity;  
import android.view.Menu;  
import android.view.MenuItem;  
import android.widget.Toast;  
  
public class OptionsExpandedMenuActivity extends Activity {  
    // Идентификаторы пунктов основного меню  
    public static final int IDM_OPEN = 101;  
    public static final int IDM_SAVE = 102;  
    public static final int IDM_EDIT = 103;  
    public static final int IDM_HELP = 104;  
    public static final int IDM_EXIT = 105;  
  
    // Идентификаторы пунктов расширенного меню  
    public static final int IDM_FIND_REPLACE = 106;  
    public static final int IDM_FIND_NEXT = 107;  
    public static final int IDM_FIND_PREV = 108;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
}
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Пункты основного меню
    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
        .setIcon(R.drawable.ic_menu_open);
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
        .setIcon(R.drawable.ic_menu_save);
    menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit")
        .setIcon(R.drawable.ic_menu_edit);
    menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help")
        .setIcon(R.drawable.ic_menu_help);
    menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
        .setIcon(R.drawable.ic_menu_exit);

    // Пункты расширенного меню. Значки не добавляем,
    // т. к. в расширенном меню они не отображаются
    menu.add(Menu.NONE, IDM_FIND_REPLACE, Menu.NONE, "Find/Replace");
    menu.add(Menu.NONE, IDM_FIND_NEXT, Menu.NONE, "Find Next");
    menu.add(Menu.NONE, IDM_FIND_PREV, Menu.NONE, "Find Previous");

    return(super.onCreateOptionsMenu(menu));
}
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    CharSequence message;
    switch (item.getItemId()) {
        case IDM_OPEN:
            message = "Open item selected";
            break;
        case IDM_SAVE:
            message = "Save item selected";
            break;
        case IDM_HELP:
            message = "Help item selected";
            break;
        case IDM_EDIT:
            message = "Edit item selected";
            break;
        case IDM_EXIT:
            message = "Exit item selected";
            break;
        case IDM_FIND_REPLACE:
            message = "Find/Replace item selected";
            break;
        case IDM_FIND_NEXT:
            message = "Find Next item selected";
            break;
        case IDM_FIND_PREV:
            message = "Find Previous item selected";
            break;
    }
}
```

```
        default:
            return false;
    }

    // Выводим уведомление о выбранном пункте меню
    Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.CENTER, 0, 0);
    toast.show();

    return true;
}
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора должно появиться меню из пяти пунктов и дополнительным пунктом **More** для расширенного меню. При выборе пункта **More** появится расширенное меню с дополнительными пунктами (рис. 11.3).

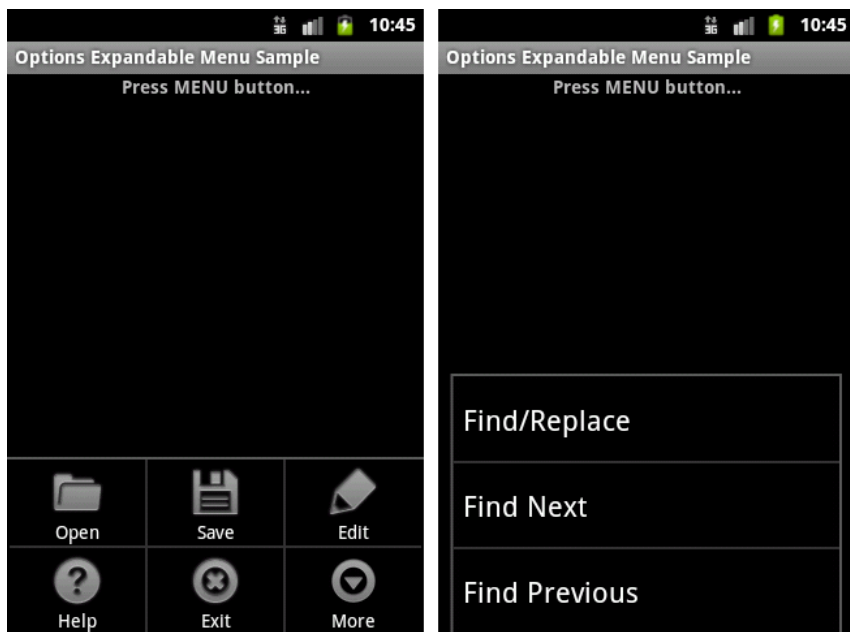


Рис. 11.3. Расширенное меню

Контекстное меню

Контекстное меню в Android напоминает контекстное меню в настольных системах, появляющееся при нажатии правой кнопки мыши. Меню вызывается при нажатии на объект в течение двух секунд (событие `long-press`).

ОБРАТИТЕ ВНИМАНИЕ

Пункты контекстного меню не поддерживают значки или быстрые клавиши (сочетания клавиш).

Для создания контекстного меню необходимо реализовать в классе `Activity` метод обратного вызова меню `onCreateContextMenu()`. В методе `onCreateContextMenu()` можно добавить пункты меню, используя один из методов `add()` и метод обратного вызова `onContextItemSelected()`.

Код для создания контекстного меню может выглядеть следующим образом:

```
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);

    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
    ...
}
```

При выборе пользователем пункта меню будет вызван метод `onContextItemSelected()`, который необходимо определить в классе, реализующем `Activity`. Этот метод передает в программу объект `MenuItem` — пункт меню, который был выбран пользователем. Для обработки события используются те же процедуры идентификации выбранного пункта меню, что и в предыдущих примерах меню.

```
public boolean onContextItemSelected(MenuItem item) {
    CharSequence message;
    switch (item.getItemId()) {
        case IDM_OPEN:
            ...
            break;
        case IDM_SAVE:
            ...
            break;
        ...
        default:
            return super.onContextItemSelected(item);
    }
}
```

Для примера приложения с контекстным меню создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ContextMenuApp`;
- ☐ **Application name** — `Context Menu Sample`;
- ☐ **Package name** — `com.samples.ui.contextmenu`;
- ☐ **Create Activity** — `ContextMenuActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch11_ContextMenu`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 11.5.

Листинг 11.5. Файл компоновки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

    android:id="@+id/Root"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Long-press for call a ContextMenu"/>

</LinearLayout>
```

В классе `ContextMenuActivity` напишите код, как в листинге 11.6.

Листинг 11.6. Файл класса окна приложения ContextMenuActivity.java

```
package com.samples.ui.contextmenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
import android.widget.LinearLayout;
import android.widget.Toast;

public class ContextMenuActivity extends Activity {
    // Идентификаторы пунктов меню
    public static final int IDM_OPEN = 101;
    public static final int IDM_SAVE = 102;
    public static final int IDM_EDIT = 103;
    public static final int IDM_HELP = 104;
    public static final int IDM_EXIT = 105;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final LinearLayout edit = (LinearLayout)findViewById(R.id.root);

        registerForContextMenu(edit);
    }
}
```

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);

    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
    menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, "Edit");
    menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help");
    menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit");
}

// Обработчик события выбора пункта меню
@Override
public boolean onContextItemSelected(Menu.Item item) {
    CharSequence message;
    switch (item.getItemId()) {
        case IDM_OPEN:
            message = "Open item selected";
            break;
        case IDM_SAVE:
            message = "Save item selected";
            break;
        case IDM_HELP:
            message = "Help item selected";
            break;
        case IDM_EDIT:
            message = "Edit item selected";
            break;
        case IDM_EXIT:
            message = "Exit item selected";
            break;
        default:
            return super.onContextItemSelected(item);
    }

    // Выводим уведомление о выбранном пункте меню
    Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.CENTER, 0, 0);
    toast.show();

    return true;
}
}

```

Запустите проект на выполнение. После запуска программы в эмуляторе при нажатии правой кнопки мыши и удержании ее в течение 1—2 секунд должно появиться контекстное меню из пяти пунктов. Внешний вид приложения с контекстным меню показан на рис. 11.4.

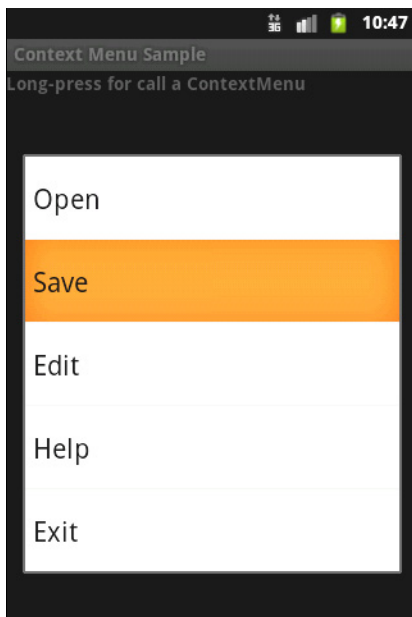


Рис. 11.4. Пример контекстного меню

Подменю

Подменю можно добавить в любое меню, кроме другого подменю. Они очень полезны, когда приложение имеет много функций, которые должны быть организованы в разделе подобно пунктам в главном меню приложений для настольных систем (**File**, **Edit**, **View** и т. д.).

Подменю создается в методе обратного вызова `onCreateOptionsMenu()`, определяемого в классе, реализующем `Activity`. Подменю добавляется для уже существующего пункта меню с помощью метода `addSubMenu()`, который возвращает объект `SubMenu`.

В объект `SubMenu` можно добавить дополнительные пункты к этому меню, используя метод `add()`. Например:

```
public boolean onCreateOptionsMenu(Menu menu) {
    SubMenu subMenuFile = menu.addSubMenu("File");
    subMenuFile.add(Menu.NONE, IDM_NEW, Menu.NONE, "New");
    subMenuFile.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
    subMenuFile.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
    ...
}
```

Для примера приложения с подменю создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — SubMenuApp;
- ☐ **Application name** — SubMenu Sample;
- ☐ **Package name** — com.samples.ui.submenu;
- ☐ **Create Activity** — SubMenuActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch11_SubMenu.

Файл компоновки для приложения будет аналогичен компоновке, приведенной в листинге 11.1. В примере будет меню из трех пунктов: **File**, **Edit** и **Help**. Для первых двух пунктов определим подменю:

❑ **File** — **New, Open, Save**;

❑ **Edit** — **Cut, Copy, Paste**.

Полный код класса SubMenuActivity представлен в листинге 11.7.

Листинг 11.7. Файл класса окна приложения SubMenuActivity.java

```
package com.samples.ui.submenu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.widget.Toast;

public class SubMenuActivity extends Activity {
    // Идентификаторы пунктов меню
    public static final int IDM_HELP = 101;

    public static final int IDM_NEW = 201;
    public static final int IDM_OPEN = 202;
    public static final int IDM_SAVE = 203;

    public static final int IDM_CUT = 301;
    public static final int IDM_COPY = 302;
    public static final int IDM_PASTE = 303;

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        SubMenu subMenuFile = menu.addSubMenu("File");
        subMenuFile.add(Menu.NONE, IDM_NEW, Menu.NONE, "New");
        subMenuFile.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
        subMenuFile.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");

        SubMenu subMenuEdit = menu.addSubMenu("Edit");
        subMenuEdit.add(Menu.NONE, IDM_CUT, Menu.NONE, "Cut");
```



```
subMenuEdit.add(Menu.NONE, IDM_COPY, Menu.NONE, "Copy");
subMenuEdit.add(Menu.NONE, IDM_PASTE, Menu.NONE, "Paste");

menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help");

return super.onCreateOptionsMenu(menu);
}

// Обработчик события выбора пункта меню
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    CharSequence message;
    switch (item.getItemId()) {
        case IDM_NEW:
            message = "New item selected";
            break;
        case IDM_OPEN:
            message = "Open item selected";
            break;
        case IDM_SAVE:
            message = "Save item selected";
            break;
        case IDM_CUT:
            message = "Cut item selected";
            break;
        case IDM_COPY:
            message = "Copy item selected";
            break;
        case IDM_PASTE:
            message = "Paste item selected";
            break;
        case IDM_HELP:
            message = "Help item selected";
            break;
        default:
            return false;
    }

    // Выводим уведомление о выбранном пункте меню
    Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.CENTER, 0, 0);
    toast.show();

    return true;
}
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора должно появиться меню из трех пунктов. При выборе пункта **File** появится его подменю (рис. 11.5). Пункт меню **Edit** также имеет собственное подменю.

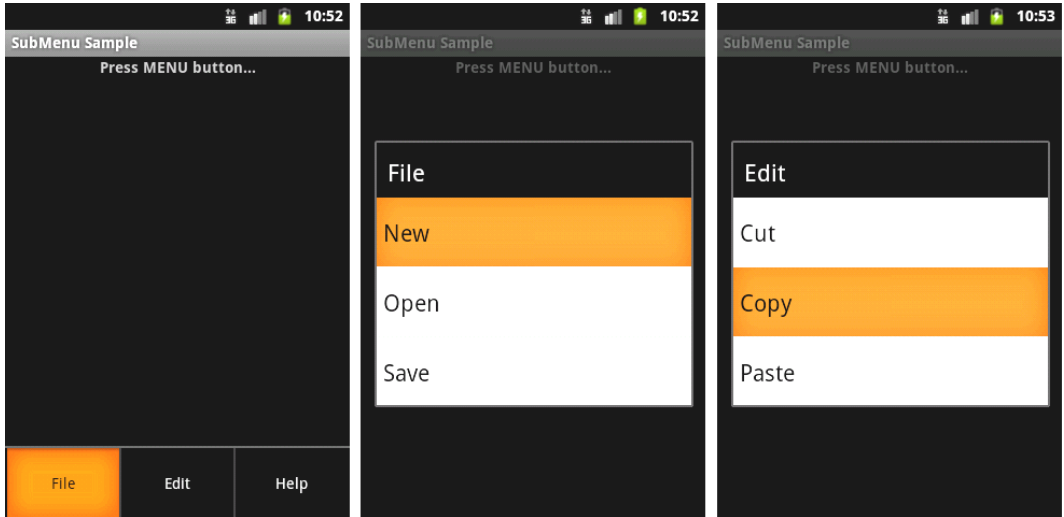


Рис. 11.5. Меню и его подменю в приложении

Добавление флажков и переключателей в меню

Для расширения функциональности в пункты меню можно добавить флажки или переключатели. Например, чтобы добавить флажок для отдельного элемента меню, необходимо использовать метод `setCheckable()`:

```
MenuItem item = menu.add(0, IDM_FORMAT_BOLD, 0, "Bold")
item.setCheckable(true);
```

Если есть необходимость добавить несколько пунктов меню с флажками или переключателями, целесообразно объединять их в группы меню.

Группа меню, так же как и пункты меню, определяется идентификатором (целым числом). Пункт меню можно добавить к группе, используя один из вариантов метода `add()`, передав ему в качестве первого параметра идентификатор группы меню. Например, пусть в коде объявлены идентификаторы для группы меню **Color** и элементов меню для установки цвета:

```
public static final int IDM_COLOR_GROUP = 200;
public static final int IDM_COLOR_RED = 201;
public static final int IDM_COLOR_GREEN = 202;
public static final int IDM_COLOR_BLUE = 203;
```

Тогда для создания группы меню с флажками необходимо назначить тот же самый идентификатор группы на каждый пункт меню и вызвать метод `setGroupCheckable()` для всей группы. В этом случае нет необходимости вызывать метод `setCheckable()` для каждого пункта меню:

```
SubMenu subMenuFile = menu.addSubMenu("Color");

subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_RED, Menu.NONE, "Red");
```

```
subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_GREEN, Menu.NONE, "Green");
subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_BLUE, Menu.NONE, "Blue");

subMenuFile.setGroupCheckable(IDM_COLOR_GROUP, true, false);
```

В метод `setGroupCheckable()` передаются три параметра:

- ❑ первый — идентификатор группы меню;
- ❑ второй — признак того, что в группе разрешены (`true`) переключатели или флажки;
- ❑ третий — устанавливает единственный (`true`) или множественный (`false`) выбор пунктов меню. Этот параметр фактически определяет внешний вид меню — с переключателями или с флажками.

Для управления состоянием флажков и переключателей необходимо написать дополнительный код в обработчике события выбора пункта меню. Например, изменение состояния флажка будет выглядеть так:

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case IDM_COLOR_RED:
            // Инвертируем состояние флажка
            item.setChecked(!item.isChecked());
            ...
        break;
```

Для примера использования групп меню в приложении создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `CheckableSubMenuApp`;
- ❑ **Application name** — `CheckableSubMenu Sample`;
- ❑ **Package name** — `com.samples.ui.checkablesubmenu`;
- ❑ **Create Activity** — `CheckableSubMenuActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch11_CheckableSubMenu`.

Файл компоновки `main.xml` используйте из листинга 11.1. В приложении создайте меню из трех пунктов: **Color**, **Font Style**, **Help**. Пункты **Color** и **Font Style** являются группами меню. Для этих групп определим подменю:

- ❑ **Color** — **Red, Green, Blue**;
- ❑ **Font Style** — **Regular, Bold, Italic**.

Полный код класса `CheckableSubMenuActivity` представлен в листинге 11.8.

Листинг 11.8. Файл класса окна приложения `CheckableSubMenuActivity.java`

```
package com.samples.ui.checkablesubmenu;

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

```

import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.widget.Toast;
public class CheckableSubMenuActivity extends Activity {
    // Идентификаторы пунктов меню
    public static final int IDM_HELP = 101;
    // Идентификаторы группы меню Color
    // и его подменю с флажками
    public static final int IDM_COLOR_GROUP = 200;
    public static final int IDM_COLOR_RED = 201;
    public static final int IDM_COLOR_GREEN = 202;
    public static final int IDM_COLOR_BLUE = 203;
    // Идентификаторы группы меню Font
    // и его подменю с переключателями
    public static final int IDM_FONT_GROUP = 300;
    public static final int IDM_REGULAR = 301;
    public static final int IDM_BOLD = 302;
    public static final int IDM_ITALIC = 303;

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

    // Создание меню
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        SubMenu subMenuFile = menu.addSubMenu("Color");
        subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_RED, Menu.NONE, "Red");
        subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_GREEN, Menu.NONE,
            "Green");
        subMenuFile.add(IDM_COLOR_GROUP, IDM_COLOR_BLUE, Menu.NONE, "Blue");
        subMenuFile.setGroupCheckable(IDM_COLOR_GROUP, true, false);

        SubMenu subMenuEdit = menu.addSubMenu("Font Style");
        subMenuEdit.add(IDM_FONT_GROUP, IDM_REGULAR, Menu.NONE, "Regular")
            .setChecked(true);
        subMenuEdit.add(IDM_FONT_GROUP, IDM_BOLD, Menu.NONE, "Bold");
        subMenuEdit.add(IDM_FONT_GROUP, IDM_ITALIC, Menu.NONE, "Italic");
        subMenuEdit.setGroupCheckable(IDM_FONT_GROUP, true, true);

        menu.add(Menu.NONE, IDM_HELP, Menu.NONE, "Help");

        return super.onCreateOptionsMenu(menu);
    }
}

```

```
// Обработчик события выбора пункта меню
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    CharSequence message;
    switch (item.getItemId()) {
        case IDM_COLOR_RED:
            item.setChecked(!item.isChecked());
            message = "Red item selected";
            break;
        case IDM_COLOR_GREEN:
            item.setChecked(!item.isChecked());
            message = "Green item selected";
            break;
        case IDM_COLOR_BLUE:
            item.setChecked(!item.isChecked());
            message = "Blue item selected";
            break;
        case IDM_REGULAR:
            item.setChecked(true);
            message = "Regular item selected";
            break;
        case IDM_BOLD:
            item.setChecked(true);
            message = "Bold item selected";
            break;
        case IDM_ITALIC:
            item.setChecked(true);
            message = "Italic item selected";
            break;
        case IDM_HELP:
            message = "Help item selected";
            break;
        default:
            return false;
    }

    // Выводим уведомление о выбранном пункте меню
    Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.CENTER, 0, 0);
    toast.show();

    return true;
}
}
```

Запустите проект на выполнение. При нажатии клавиши <MENU> клавиатуры эмулятора на экране должно появиться меню из трех пунктов. При выборе пункта **Color** появится его подменю выбора цвета с флажками, а для **Font Style** — подменю с переключо-

чателями. Состояние флажков и переключателей обрабатывается в коде программы и сохраняется при повторных вызовах меню, если программа не была закрыта.

Внешний вид приложения показан на рис. 11.6.

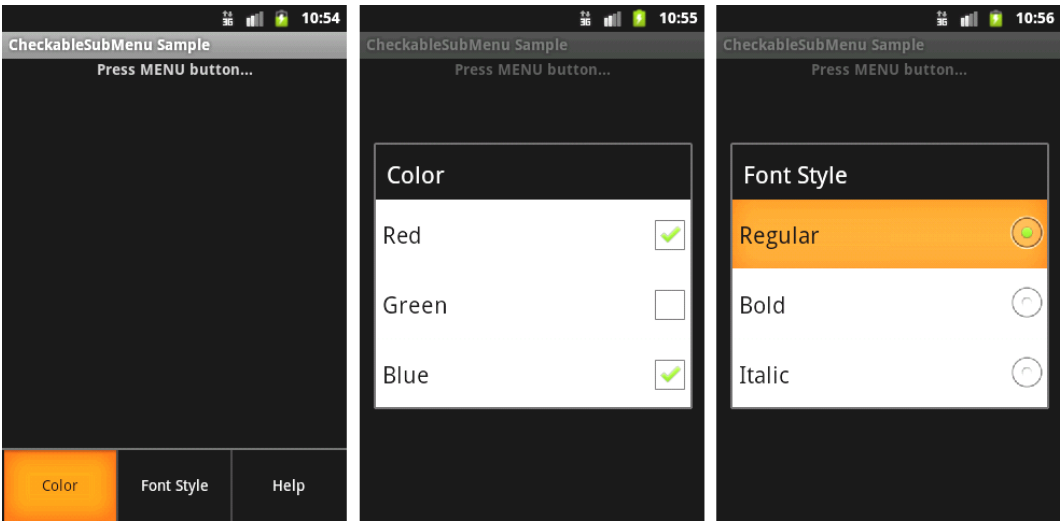


Рис. 11.6. Группы подменю с флажками и переключателями

Резюме

В этой главе мы изучили создание меню в приложениях. Платформа Android предлагает широкий выбор меню различных типов, что позволяет создавать для своих приложений гибкое и удобное взаимодействие с пользователем мобильного телефона. Существует еще один способ создания меню — определение структуры меню в XML-файле. Эту структуру меню можно загрузить в Activity, но данный способ мы будем рассматривать несколько позже, в *главе 15*, посвященной использованию файлов ресурсов в приложениях.

Далее мы переходим к главе, где будем изучать непосредственно компонент Activity, использование в приложении нескольких Activity и их взаимодействие между собой и компонентами других приложений и системы с помощью объектов Intent.



ГЛАВА 12

Activity

В предыдущих главах все приложения состояли из одного Activity. Реальные приложения, как правило, имеют несколько окон — состоят из нескольких Activity, которыми надо уметь управлять и которые должны взаимодействовать между собой.

Как правило, один из Activity приложения (окно которого открывается при запуске приложения) помечен как главный. Также из открытого Activity можно запустить другой Activity, даже если он определен в другом приложении. Пользователю будет казаться, что все запускаемые им Activity являются частями одного приложения, хотя на самом деле они могут быть определены в разных приложениях и работать в разных процессах.

Процессы в системе Android

Когда хотя бы один из компонентов приложения (или все приложение) будет востребован, система Android запускает процесс, который содержит единственный основной поток для выполнения. По умолчанию все компоненты приложения работают в этом процессе и потоке.

Однако можно принять меры, чтобы компоненты работали в других процессах и порождали дополнительные потоки для любого процесса.

Все компоненты инициализируются в основном потоке процесса. Отдельные потоки для каждого экземпляра обычно не создаются. Следовательно, все методы обратного вызова, определенные в компоненте и вызываемые системой, всегда работают в основном потоке процесса. Это означает, что компонент не должен выполнять в методах обратного вызова длительные операции (например, загрузку файлов из сети или циклы вычисления) или блокировать системный вызов, т. к. это блокирует любые другие компоненты в этом процессе. Для таких операций порождают отдельные потоки.

Система Android может решить завершить процесс, в случае нехватки памяти или если память востребована другими, более важными процессами. Прикладные компоненты, выполняющиеся в этих процессах, будут уничтожены. Процесс будет перезапущен для компонентов в случае их повторного вызова.

При выборе процесса для уничтожения Android оценивает относительную важность этого процесса с точки зрения пользователя, т. е. видимый пользователю компонент данного процесса, например Activity на переднем плане, считается более важным ком-

понентом, чем компонент Service (служба, т. е. программа, работающая в фоновом режиме и не имеющая GUI), выполняющийся в другом процессе. Также система в первую очередь завершит процесс с Activity, который больше не виден на экране, а не процесс с видимым Activity. Поэтому решение о завершении процесса зависит от состояния компонентов, выполняющихся в данном процессе.

Порядок, в котором процессы уничтожаются для освобождения ресурсов, определяется приоритетом. Система Android пытается поддерживать процесс приложения максимально долго, но, в конечном счете, будет вынуждена удалить старые процессы, если заканчивается свободная память. Чтобы определить, какой процесс сохранить или уничтожить, система Android создает иерархию важности процессов, основанную на компонентах, запущенных в данный момент времени, а также состоянии этих компонентов (рис. 12.1).



Рис. 12.1. Приоритет и статус процессов

Процессы с самой низкой важностью уничтожаются первыми. Есть пять уровней в иерархии важности. Следующий список представляет их в порядке убывания важности.

1. *Активный процесс* (Foreground Process). Процесс считается активным, если выполняется любое из следующих условий:

- в процессе выполняется Activity, с которым взаимодействует пользователь;
- в процессе выполняется служба, связанная с Activity, с которым взаимодействует пользователь;
- процесс имеет объект Service, и выполняется один из методов обратного вызова, определенных в этом объекте;
- процесс имеет объект BroadcastReceiver, и выполняется его метод обратного вызова для приема Intent.

Одновременно могут существовать только несколько приоритетных процессов. Они будут уничтожены только в крайнем случае — если памяти настолько мало, что они все вместе не в состоянии продолжать работу.

2. *Видимый процесс* (Visible Process) — компонент из данного процесса еще может вызываться пользователем. Это может быть процесс Activity, который не находится в фокусе, но все еще виден пользователю. Видимым может также быть процесс службы, которая в данный момент связана с Activity, находящимся на переднем плане (или частично закрытым другим Activity). Это может произойти, например, при вызове диалога, который не занимает весь экран, когда Activity потерял фокус, но виден пользователю и находится позади диалога. Видимый процесс считается важным и не будет уничтожен, пока останутся процессы с более низким приоритетом.
3. *Сервисный процесс* (Service Process) — процесс, в котором выполняется Service и который не относится ни к одной из двух предыдущих категорий. Хотя сервисные процессы обычно не привязаны к интерфейсу, видимому пользователем, они выполняют задания, нужные пользователю, например фоновая работа медиаплеера или загрузка данных из сети, так, что система сохраняет их при наличии свободной памяти наряду со всеми активными и видимыми процессами.
4. *Фоновый процесс* (Background Process) — процесс, в котором выполняется Activity, который в настоящее время не виден пользователю. Эти процессы не имеют никакого прямого воздействия на пользовательский ввод и могут быть уничтожены в любое время, чтобы освободить память для активного, видимого или сервисного процесса. Обычно имеется много фоновых процессов, они сохраняются в списке LRU (Least Recently Used, "не использующиеся дольше всех"), чтобы гарантировать, что находящийся в конце этого списка процесс, в котором выполняется Activity, был бы уничтожен в последнюю очередь.
5. *Пустой процесс* (Empty Process) — не содержит никаких активных компонентов приложения. Единственная причина сохранять такой процесс — использовать его, как кэш, чтобы уменьшить время запуска при вызове компонента. Система уничтожает эти процессы в первую очередь.

Если в одном процессе выполняются несколько компонентов, Android определяет приоритет процесса по компоненту с самым высоким приоритетом. Например, если в процессе выполняется служба и видимый Activity. Если от некоторого процесса зависят другие процессы, его ранг также может быть увеличен.

Состояния Activity

Когда пользователь работает с мобильным телефоном, он постоянно открывает, закрывает Activity или перемещает их на задний план. Activity может находиться в трех состояниях:

- *активный* (active или running) — Activity находится на переднем плане экрана мобильного устройства и является центром для интерактивного взаимодействия с пользователем;
- *приостановленный* (paused) — Activity потерял фокус, но все еще видим пользователю. То есть другой Activity находится сверху и частично перекрывает данный Activity. Приостановленный Activity может быть уничтожен системой в критических ситуациях при нехватке памяти;
- *остановленный* (stopped) — если данный Activity полностью закрыт другим Activity. Он больше не видим пользователю и может быть уничтожен системой, если память необходима для другого, более важного процесса.

Если Activity, который был уничтожен системой, снова нужно отобразить на экране, он должен быть полностью перезапущен и восстановлен в своем предыдущем состоянии. Activity при переходе от одного состояния к другому получает уведомления через защищенные методы:

- ❑ `onCreate()` — вызывается при создании Activity. Внутри этого метода настраивают статический интерфейс Activity — создают представления, связывают данные со списками и т. д. Этот метод принимает один параметр — объект `Bundle`, содержащий предыдущее состояние Activity (если это состояние было сохранено);
- ❑ `onRestart()` — вызывается после того, как Activity был остановлен и снова был запущен пользователем. Всегда сопровождается вызовом метода `onStart()`;
- ❑ `onStart()` — вызывается непосредственно перед тем, как Activity становится видимым. Сопровождается вызовом метода `onResume()`, если Activity получает передний план, или вызовом метода `onStop()`, если становится скрытым;
- ❑ `onResume()` — вызывается непосредственно перед тем, как Activity начинает взаимодействие с пользователем. Всегда сопровождается вызовом метода `onPause()`;
- ❑ `onPause()` — вызывается, когда система собирается запустить другой Activity. Обычно этот метод используется, чтобы передать несохраненные изменения на сохранение и остановить выполнение задач, требующих ресурсов центрального процессора и не нужных после приостановки Activity. Сопровождает любой метод `onResume()`, если Activity возвращается снова на передний план, или метод `onStop()`, если окно Activity становится невидимым для пользователя;
- ❑ `onStop()` — вызывается, когда Activity становится невидимым. Это может произойти при его уничтожении, или если был запущен другой Activity (существующий или новый), перекрывший окно текущего Activity. Всегда сопровождает любой вызов метода `onRestart()`, если Activity возвращается на передний план для взаимодействия с пользователем, или метода `onDestroy()`, если этот Activity уничтожается;
- ❑ `onDestroy()` — вызывается перед уничтожением Activity. Это последний запрос, который получает Activity от системы. Данный метод вызывается по окончании работы Activity, при вызове метода `finish()` или в случае, когда система разрушает этот экземпляр Activity для освобождения ресурсов. Эти два сценария уничтожения можно определить вызовом метода `isFinishing()`.

Эти методы можно реализовать при необходимости в классе Activity, чтобы выполнить определенные действия при изменении состояния данного Activity. При реализации любого из этих методов необходимо всегда сначала вызывать версию этого метода из суперкласса. Например:

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

Из всех вышеперечисленных методов обязательным при написании кода класса Activity является только метод `onCreate()`, чтобы задать начальную установку параметров при инициализации Activity. Также часто надо реализовывать метод `onPause()`, чтобы сохранить пользовательские настройки Activity и подготовиться к прекращению взаимодействия с пользователем.

Запуск Activity с использованием объектов Intent

Компоненты Android-приложений, в том числе и Activity, запускаются через объекты Intent. Это средство для позднего связывания во время выполнения между компонентами одного или нескольких приложений.

В каждом случае система Android находит соответствующий Activity, чтобы ответить на Intent, инициализируя его в случае необходимости. Объекты Intent могут быть разделены на две группы:

- ❑ *явный Intent* — определяет целевой компонент по имени (составляющее поле имени, упомянутое ранее, имеет набор значения);
- ❑ *неявный Intent* — не называет адресата (поле для составляющего имени — пробел). Неявные Intent часто используются, чтобы активизировать компоненты в других приложениях.

Явный Intent обычно используют для сообщений внутри приложения, например, когда один Activity запускает другой Activity из этого приложения.

Неявный Intent используют для запуска компонентов других приложений. В файле манифеста приложения обычно декларируется фильтр Intent. В отсутствие определяемого адресата система Android просматривает фильтры Intent всех приложений и находит компонент (Activity, Service или Broadcast Receiver), фильтр Intent которого является наиболее подходящим для выполнения данного неявного Intent.

Intent-фильтры и запуск заданий

Intent-фильтры декларируют ограничения компонента по приему неявных объектов Intent, которые он может обрабатывать. Если компонент не имеет никаких Intent-фильтров, он может принимать только явные объекты Intent. Компонент с фильтрами может принимать и явные, и неявные Intent.

В фильтре Intent декларируется только три составляющих объекта Intent: действие, данные, категория. В манифесте приложения фильтр Intent объявляется в элементе `<intent-filter>`. Например, в любом приложении есть главный Activity, который устанавливается как точка входа для задания:

```
...
<activity
    android:name=".ContactListActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
...
```

Фильтр такого вида в элементе `<action>` помечает Activity, как запускаемый по умолчанию. Элемент `<category>` заставляет значок и метку для Activity отображаться на пане-

ли **Application Launcher**, давая пользователям возможность запускать задание и возвращаться к этому заданию в любое время после того, как оно было запущено.

ОБРАТИТЕ ВНИМАНИЕ

Фильтр Intent — экземпляр класса `IntentFilter`. Однако, так как система Android должна знать о возможностях компонента прежде, чем она сможет запустить этот компонент, фильтры Intent всегда устанавливаются только в файле манифеста приложения как элементы `<intent-filter>`, а не в коде приложения.

У фильтра есть поля, которые соответствуют действию, данным и категориям объекта Intent. Неявный Intent проверяется в фильтре по всем трем полям. Чтобы Intent запустил компонент, которому принадлежит фильтр, он должен пройти все три теста. Если один из них не проходит, то система не запустит этот компонент — по крайней мере, на основе этого фильтра. Однако, так как у компонента могут быть несколько фильтров, Intent, который не проходит через один из фильтров компонента, может пройти через другой. Каждый фильтр описывает возможность компонента и набор Intent, которые компонент желает получать.

Запуск Activity с помощью явного объекта Intent

Объект Intent является структурой данных, содержащей абстрактное описание выполняемой операции. Чтобы вызвать другой Activity, в объект Intent надо передать имя этого Activity. Имя устанавливается методами `setComponent()`, `setClass()` или `setClassName()`.

Далее, чтобы запустить Activity, объект Intent передают в метод `Context.startActivity()`. Этот метод принимает единственный параметр — объект Intent, описывающий Activity, который будет запускаться. Например, вызвать Activity с именем `NewActivity` в коде программы можно следующим образом:

```
// Создаем объект Intent
Intent intent = new Intent();
// Устанавливаем имя вызываемого компонента
intent.setClass(getApplicationContext(), NewActivity.class);
// Запускаем компонент
startActivity(intent);
```

Взаимодействие Activity и изменение их состояния лучше рассмотреть на практическом примере. Создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ActivityEvents`;
- ☐ **Application nam** — `Activity Events`;
- ☐ **Package name** — `com.samples.app.activityevents`;
- ☐ **Create Activity** — `MainActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch12_ActivityEvents`.

В файле компоновки `main.xml` добавьте кнопку с надписью **Call Activity** и идентификатором `bCallActivity`, а также текстовое поле `TextView` с идентификатором `text` для ото-

бражения событий, происходящих при переключении Activity. Код файла компоновки приведен в листинге 12.1.

Листинг 12.1. Файл компоновки main.xml

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

    <Button
        android:text="Call Activity"
        android:id="@+id/bCallActivity"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:layout_width="fill_parent"/>

    <TextView
        android:id="@+id/text"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:textStyle="bold"/>

</LinearLayout>
```

В приложении будут два Activity с именами MainActivity и SecondActivity. В классе MainActivity переопределим все защищенные методы класса Activity для отслеживания событий, происходящих в Activity. Код класса MainActivity представлен в листинге 12.2.

Листинг 12.2. Файл класса окна приложения MainActivity.java

```
package com.samples.app.activityevents;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity
    implements View.OnClickListener {
    private TextView text;

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

```
        text = (TextView)findViewById(R.id.text);
        text.append("onCreate()\n");
    }

    @Override
    public void onClick(View arg0) {
        Intent intent = new Intent();
        intent.setClass(getApplicationContext(), SecondActivity.class);
        startActivity(intent);
    }

    @Override
    public void onRestart() {
        super.onRestart();
        text.append("onCreate()\n");
    }

    @Override
    public void onStart() {
        super.onStart();
        text.append("onStart()\n");
    }

    @Override
    public void onResume() {
        super.onResume();
        text.append("onResume()\n");
    }

    @Override
    public void onPause() {
        super.onPause();
        text.append("onPause()\n");
    }

    @Override
    public void onStop() {
        super.onStart();
        text.append("onStop()\n");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        text.append("onDestroy()\n");
    }
}
```

Теперь нам надо создать второй Activity. На панели **Package Explorer** в директории вашего проекта щелкните правой кнопкой мыши на значке пакета com.samples.

app.activityevents, в котором находится файл MainActivity, и в контекстном меню выберите пункт **New | Class**. Откроется диалоговое окно создания нового класса **New Java Class**. В этом окне введите в поле **Name** название класса `SecondActivity`, модификатор оставьте заданный по умолчанию — **public**. В поле **Superclass** введите имя базового класса, в нашем случае это `android.app.Activity` (надо вводить полное имя класса). Внешний вид окна показан на рис. 12.2.

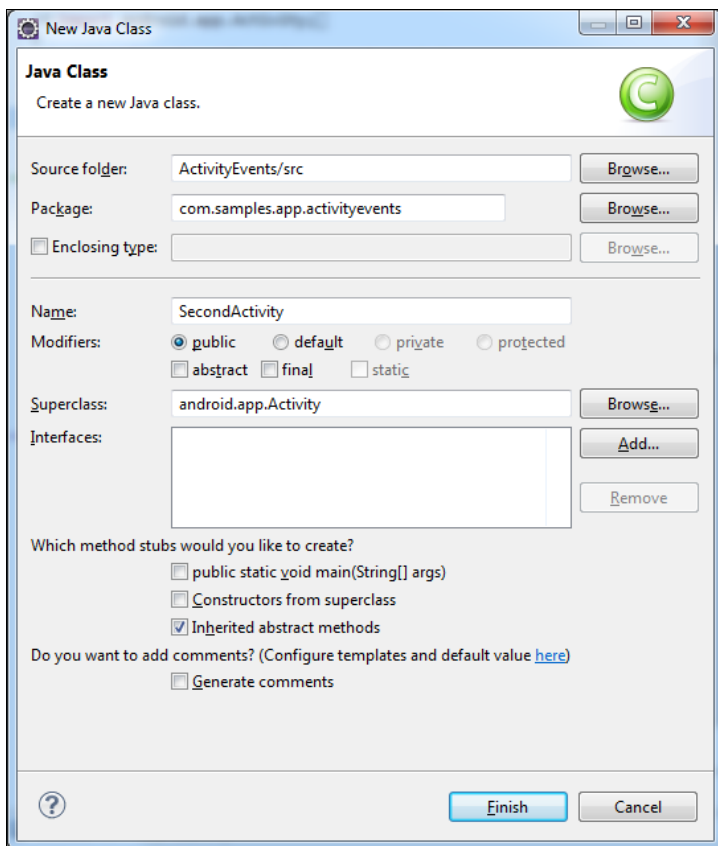


Рис. 12.2. Создание нового класса, производного от Activity

Класс `SecondActivity` будет пустым, он не будет иметь даже своего файла компоновки (пока он нам не нужен). Код класса `SecondActivity` представлен в листинг 12.3.

Листинг 12.3. Файл класса окна приложения `SecondActivity.java`

```
package com.samples.app.activityevents;

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

public class SecondActivity extends Activity {
    @Override
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    this.setTitle("Second Activity");
}
}

```

Чтобы приложение могло "видеть" второй Activity, необходимо указать его в файле манифеста приложения. Для этого выберите в представлении Project Explorer файл AndroidManifest.xml и в открывшемся редакторе манифеста перейдите на вкладку **Application**. На панели **Application Nodes** нажмите кнопку **New** и в открывшемся диалоговом окне выберите элемент **Activity** (рис. 12.3).

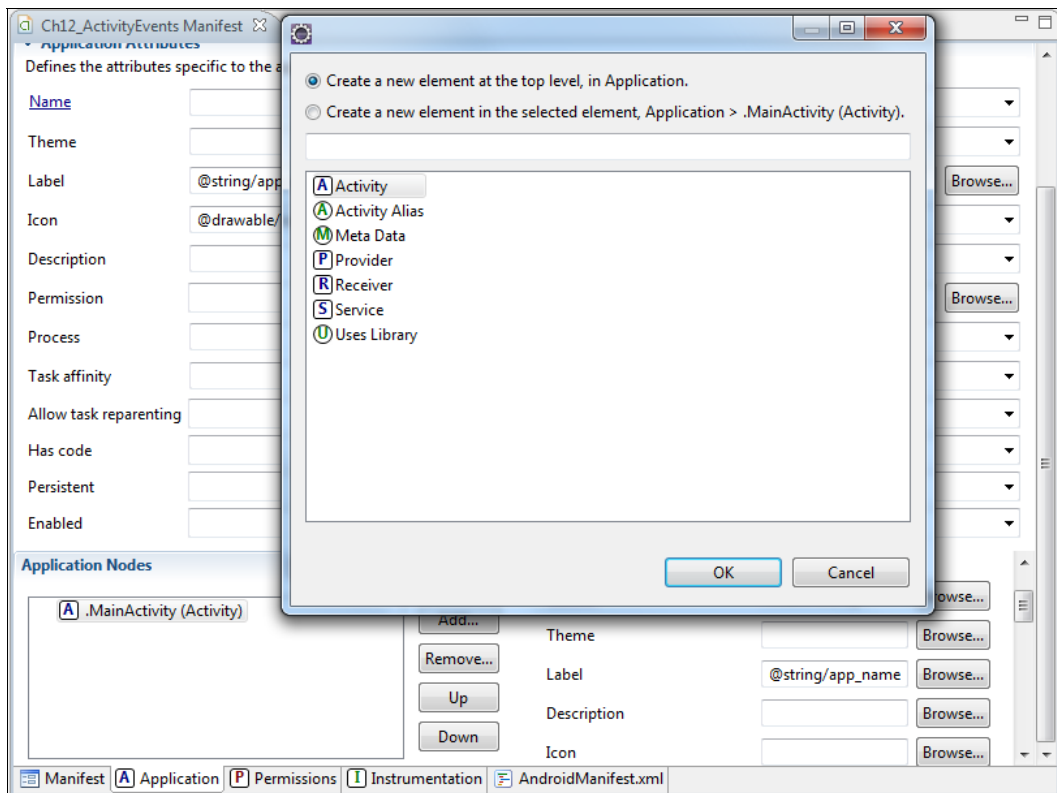


Рис. 12.3. Добавление нового элемента в файл AndroidManifest.xml

На панель **Application Nodes** будет добавлен новый узел Activity. Если его выделить, в правой части окна отобразится панель с заголовком **Attributes for Activity**. На этой панели в строке **Name** щелкните по кнопке **Browse** и в открывшемся окне выберите класс **SecondActivity**, как показано на рис. 12.4.

После этих действий в файле AndroidManifest.xml появится дополнительный элемент `<activity>` с атрибутом `android:name="SecondActivity"`, как представлено в листинге 12.4.

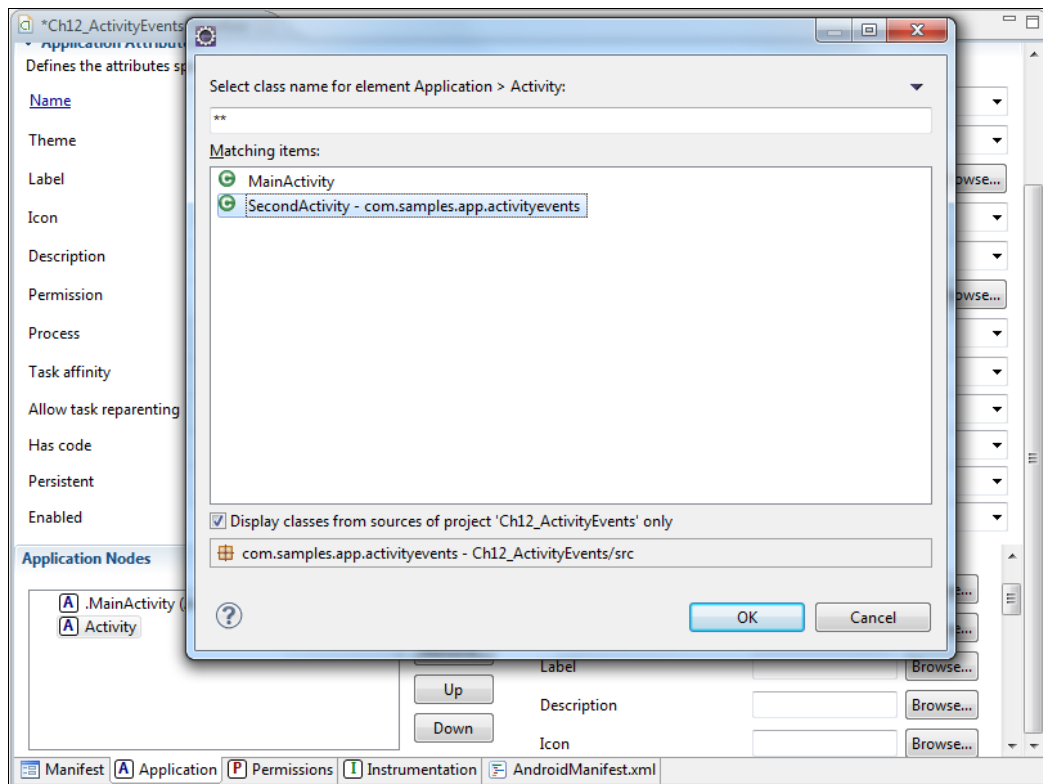


Рис. 12.4. Окно выбора Activity

Листинг 12.4. Файл AndroidManifest.xml

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

    <application
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="SecondActivity">
        </activity>
    </application>
</manifest>
```

Скомпилируйте приложение и запустите его на эмуляторе Android. При запуске приложения главный Activity отработает последовательно три состояния: `onCreate()`, `onStart()`, `onResume()`. Если нажать кнопку **Call Activity**, запустится второй Activity приложения — `SecondActivity`. Если теперь нажать клавишу <BACK> на эмуляторе, на экран снова будет выведен `MainActivity`. Пока `MainActivity` находился в остановленном состоянии, были последовательно отработаны события `onPause()`, `onStop()`, а при возвращении `MainActivity` на передний план — события `onCreate()`, `onStart()`, `onResume()`, как представлено на рис. 12.5.

Если теперь кратковременно нажать клавишу <OFF>, экран эмулятора будет заблокирован и появится стандартная заставка Android. `MainActivity` уйдет на задний план, но

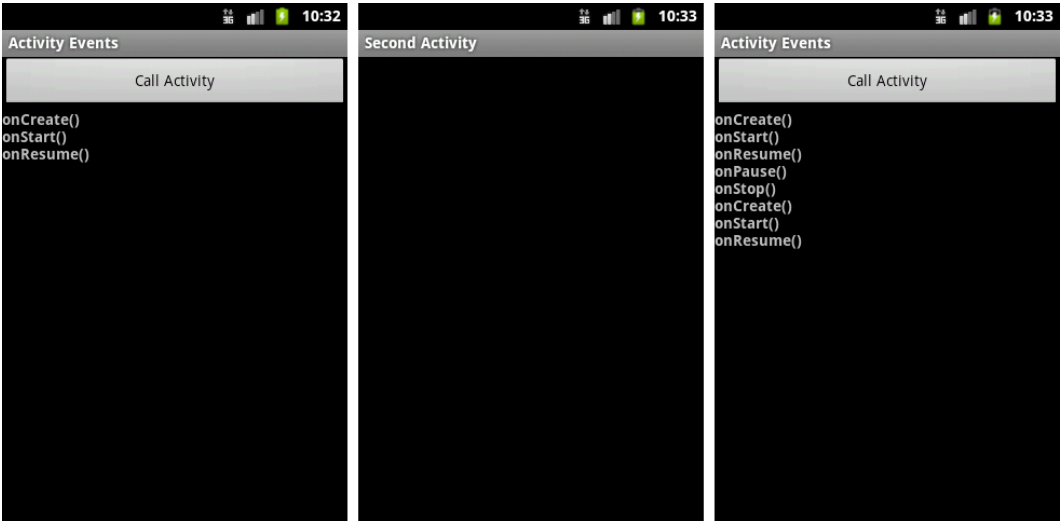


Рис. 12.5. События, происходящие при переключении Activity

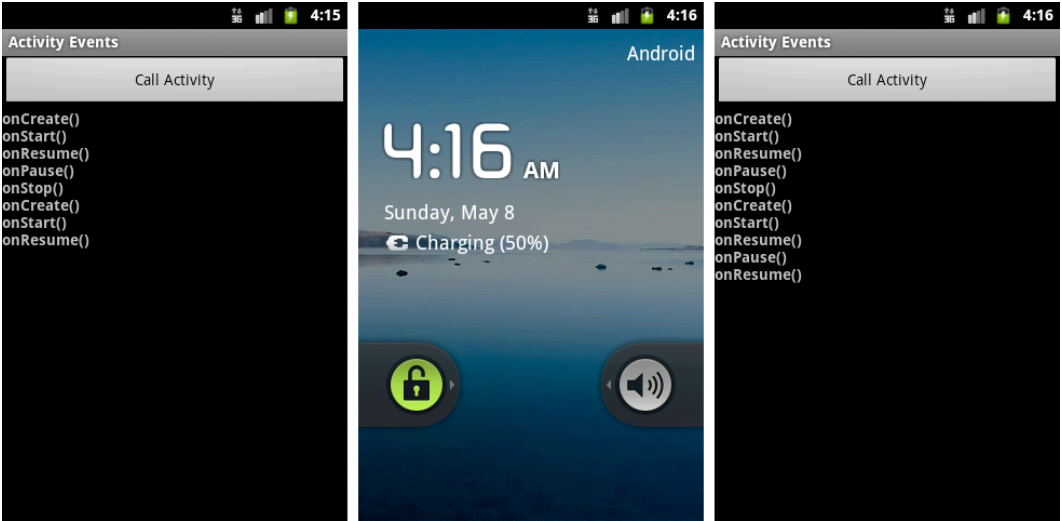


Рис. 12.6. Приостановка выполнения Activity

приложение не будет закрыто. Разблокировать экран можно клавишей <MENU> эмулятора. Наш `MainActivity` снова перейдет на передний план, но будут отработаны только события `onPause()` и `onResume()`, т. е. выполнение `MainActivity` приостановилось, но сам объект `Activity` не был уничтожен системой (рис. 12.6).

Стек Activity

В системе Android все `Activity` сохраняются в стеке. Когда один `Activity` запускает другой, новый `Activity` помещается в стек и становится активным `Activity`. Предыдущий `Activity` также остается в стеке, но опускается вниз. Когда пользователь нажимает клавишу <BACK> на мобильном телефоне, текущий `Activity` выталкивается из стека, и его замещает предыдущий `Activity`, который снова отображается на экране.

Стек с находящимися в нем `Activity` называется *заданием*. Все `Activity` в задании перемещаются вместе, как один модуль. Все задание, т. е. весь стек `Activity` приложения, может находиться на переднем плане или в фоне. Стек содержит объекты, и если задание имеет больше одного открытого экземпляра того же самого подкласса `Activity`, то стек имеет отдельный вход для каждого экземпляра. `Activity` в стеке никогда не перестраиваются, только помещаются и выталкиваются (рис. 12.7).

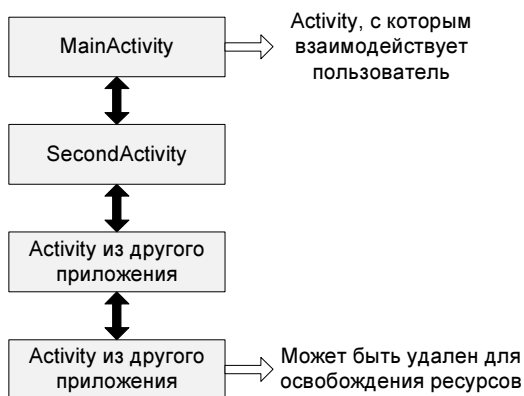


Рис. 12.7. Стек Activity

Если пользователь оставляет задание в течение долгого времени, система очищает задание от всех `Activity`, кроме корневого `Activity`. Чтобы зафиксировать состояние `Activity` перед его полным уничтожением (событие `onDestroy()`), в классе, реализующем `Activity`, необходимо реализовать метод `onSaveInstanceState()`. Android вызывает этот метод перед созданием `Activity`, т. е. раньше вызова `onPause()`.

Система передает методу объект `Bundle`, в который можно записать параметры, динамическое состояние `Activity` как пары *имя-значение*. Когда `Activity` будет снова вызван, объект `Bundle` передается системой в качестве параметра в метод `onCreate()` и в метод `onRestoreInstanceState()`, который вызывается после `onStart()`, чтобы один из них или они оба могли установить `Activity` в предыдущее состояние.

В отличие от метода `onPause()` и других методов, рассмотренных ранее, методы `onSaveInstanceState()` и `onRestoreInstanceState()` не относятся к методам жизненного

цикла Activity. Система будет вызывать их не во всех случаях. Например, Android вызывает `onSaveInstanceState()` прежде, чем Activity становится уязвимым к уничтожению системой, но не вызывает его, когда экземпляр Activity разрушается пользовательским действием (при нажатии клавиши <BACK>). В этом случае нет никаких причин для сохранения состояния Activity.

Поскольку метод `onSaveInstanceState()` вызывается не во всех случаях, его необходимо использовать только для сохранения промежуточного состояния Activity. Для сохранения данных лучше использовать метод `onPause()`.

Когда система завершает Activity в принудительном порядке, чтобы освободить ресурсы для других приложений, пользователь может снова вызвать этот Activity с сохраненным предыдущим состоянием.

Обмен данными между Activity

Обмен данными между Activity актуален для большинства приложений. Иногда требуется вернуть результат выполнения из вызываемого Activity в вызывающий Activity при его закрытии. Например, можно запустить Activity, который позволяет пользователю выбирать человека в списке контактов. При закрытии Activity возвращает данные человека, который был выбран: его полное имя и телефон.

Чтобы запустить Activity и получить результат его выполнения, необходимо вызвать метод `startActivityForResult(Intent, int)` со вторым параметром, идентифицирующим запрос. Результат возвращается через метод `onActivityResult(int, int, Intent)`, определенный в родительском Activity.

Кроме этого, можно передавать дополнительные параметры (extra-параметры) в вызываемый Activity. Это пары *ключ-значение* для информации, которую нужно предоставить вызываемому компоненту. Объект `Intent` имеет ряд методов `put...()` для вставки различных типов дополнительных данных и аналогичный набор методов `get...()` для чтения данных. Extra-параметры устанавливаются и читаются как объекты класса `Bundle` с использованием методов `putExtras()` и `getExtras()`.

Например, запустить Activity с именем класса `SecondActivity` и передать ему два дополнительных параметра можно следующим образом:

```
// Идентификатор запроса
private static final int ACTION_EDIT = 101;

// Идентификаторы extra-параметров
private static final String ID_EXTRA_1 = "Param1";
private static final String ID_EXTRA_2 = "Param2";
...

// Создаем объект Intent
Intent intent = new Intent();

// Добавление extra-параметров
String param1 = "Some Text... ";
int param2 = "12345";
```

```

intent.putExtra(ID_EXTRA_1, param1);
intent.putExtra(ID_EXTRA_2, param2);
...
// Определение класса запускаемого Activity
intent.setClass(this, SecondActivity.class);

// Вызов Activity
startActivityForResult(intent, ACTION_EDIT);

```

В дочернем Activity мы можем прочесть переданные extra-параметры, получив сначала объект класса `Bundle` как набор параметров, представляющих собой пары `ключ-значение`, а затем приводя их к нужному типу данных, используя методы класса `Bundle` `get()`, `getBoolean()`, `getString()` и т. д. В эти методы в качестве параметра передается идентификатор параметра, который является уникальным ключом для всего набора пар `ключ-значение`. Эти ключи определяются теми же параметрами, которые были определены в вызывающем Activity (в нашем случае — это `ID_EXTRA_1` и `ID_EXTRA_2`). В целом код обработки может выглядеть примерно так:

```

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

    // Получаем объект Bundle и читаем параметры
    Bundle extras = getIntent().getExtras();
    String param1 = extras.getString(MainActivity.ID_EXTRA_1);
    int param2 = extras.getInt(MainActivity.ID_EXTRA_2);
}

```

Когда дочерний Activity закрывается, в нем можно вызвать метод `setResult(int)`, чтобы вернуть данные в родительский Activity. Этот метод возвращает код результата закрытия Activity, который может быть стандартным результатом, определяемым константами `RESULT_CANCELED`, `RESULT_OK` или определяемым пользователем результатом `RESULT_FIRST_USER`.

```

Intent intent = new Intent();
intent.putExtra(MainActivity.ID_EXTRA_1, param1);
intent.putExtra(MainActivity.ID_EXTRA_2, param2);

// Возвращаем результат в вызывающий Activity
setResult(RESULT_OK, intent);
finish();

```

Кроме того, дочерний Activity может произвольно вернуть назад объект `Intent`, содержащий любые дополнительные данные. Вся эта информация в родительском Activity появляется через метод обратного вызова `Activity.onActivityResult()`, вместе с идентификатором, который был передан в метод `startActivityForResult()` при вызове Activity:

```

protected void onActivityResult(
    int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
}

```

```

if (resultCode == RESULT_OK) {
    Bundle extras = data.getExtras();

    switch (requestCode) {
        case IDM_ADD:
            String param1 = extras.getString(ID_EXTRA_1),
            int param2 = extras.getInt(ID_EXTRA_2);
            ...
            break;
        ...
    }
    ...
}
}

```

Если дочерний Activity завершится неудачно или будет закрыт пользователем без подтверждения ввода, то родительский Activity получит результат с кодом `RESULT_CANCELED`. Реализуем все это в практическом приложении. Создайте в Eclipse новый проект и в окне **Create New Project** введите следующие значения:

- ☐ **Project name** — `ContactEditor`;
- ☐ **Application name** — `Exchange Data`;
- ☐ **Package name** — `com.samples.app.contactedit`;
- ☐ **Create Activity** — `MainActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch12_ContactEditor`.

В главном окне будут данные телефонного контакта, который можно будет редактировать в другом окне. Всего в приложении будут два Activity:

- ☐ `MainActivity` — главное окно со списком контактов и меню для редактирования;
- ☐ `SecondActivity` — окно редактирования контакта.

В файле манифеста не забудьте добавить помимо `MainActivity` еще и `SecondActivity`, аналогично листингу 12.4 предыдущего примера. Для этого Activity фильтры Intent не требуются, устанавливаются только атрибуты `android:name` и `android:label`. Если вы забудете это сделать, второй Activity нельзя будет вызвать, система его не увидит.

В файле компоновки главного окна приложения `main.xml` будут отображены данные контакта: имя человека, его телефон и кнопка вызова дочернего Activity для модификации контакта. Полный код файла компоновки представлен в листинге 12.5.

Листинг 12.5. Файл компоновки главного окна приложения `main.xml`

```

<?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" >

```

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Name: "
        android:textSize="18sp"
        android:layout_margin="2sp"/>
    <TextView
        android:id="@+id/textName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_margin="2sp" />
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Phone: "
        android:textSize="18sp"
        android:layout_margin="2sp" />
    <TextView
        android:id="@+id/textPhone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_margin="2sp" />
</LinearLayout>
<Button
    android:id="@+id/bEditData"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="Edit Data" />
</LinearLayout>

```

В классе `MainActivity` мы реализуем вызов дочернего `Activity` и передачу ему двух строковых параметров `NAME` и `PHONE`. Также нам потребуется реализовать метод `onActivityResult()` для чтения результата из дочернего `Activity`. Полный код класса `MainActivity` представлен в листинге 12.6.

Листинг 12.6. Файл класса `MainActivity.java`

```

package com.samples.app.senddata;

import android.app.Activity;

```

```

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity
    implements View.OnClickListener {

    private TextView textName;
    private TextView textPhone;

    // Константы, идентифицирующие extra-параметры
    public final static String NAME = "Name";
    public final static String PHONE = "Phone";

    private final static int ACTION_EDIT = 101;

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

        textName = (TextView)findViewById(R.id.textName);
        textPhone = (TextView)findViewById(R.id.textPhone);

        textName.setText("Andrew Ivanov");
        textPhone.setText("123456789");
    }

    @Override
    public void onClick(View v) {
        // Создаем объект Intent
        Intent intent = new Intent();
        intent.setClass(getApplicationContext(), SecondActivity.class);

        // Добавляем extra-параметры
        intent.putExtra(NAME, textName.getText());
        intent.putExtra(PHONE, textPhone.getText());

        // Вызываем Activity
        startActivityForResult(intent, ACTION_EDIT);
    }

    protected void onActivityResult(
        int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == RESULT_OK) {
            Bundle extras = data.getExtras();

```



```

        textName.setText (extras.getString (NAME) );
        textPhone.setText (extras.getString (PHONE) );
    }
}
}

```

Класс `SecondActivity` представляет функциональность для редактирования контакта: два текстовых поля для редактирования имени и телефона и командные кнопки для сохранения изменений или отмены ввода и перехода в основной `Activity`.

Код файла компоновки для окна редактирования контакта `second.xml` представлен в листинге 12.7

Листинг 12.7. Файл компоновки дочернего окна `second.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Name:" />
        <EditText
            android:id="@+id/editName"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent" />
    </LinearLayout>

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" >
        <TextView
            android:text="Pnone:"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <EditText
            android:id="@+id/editPhone"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent" />
    </LinearLayout>

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent">

```

```

<Button
    android:id="@+id/bSave"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:layout_weight="1"
    android:onClick="onClick"
    android:text="Save"/>

<Button
    android:id="@+id/bCancel"
    android:text="Cancel"
    android:layout_width="fill_parent"
    android:layout_weight="1"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>

</LinearLayout>
</LinearLayout>

```

Класс `SecondActivity` должен принимать от вызывающего `Activity` данные редактируемого контакта и передавать отредактированные данные обратно в родительский `Activity`. Код этого класса представлен в листинге 12.8.

Листинг 12.8. Файл класса дочернего окна `SecondActivity.java`

```

package com.samples.app.senddata;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class SecondActivity extends Activity
    implements View.OnClickListener {
    private EditText editName;
    private EditText editPhone;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);

        editName = (EditText)findViewById(R.id.editName);
        editPhone = (EditText)findViewById(R.id.editPhone);

        Bundle extras = getIntent().getExtras();
        editName.setText(extras.getString(MainActivity.NAME));
        editPhone.setText(extras.getString(MainActivity.PHONE));
    }
}

```

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.btnSave:
            Intent intent = new Intent();
            intent.putExtra(
                MainActivity.NAME, editName.getText().toString());
            intent.putExtra(
                MainActivity.PHONE, editPhone.getText().toString());

            setResult(RESULT_OK, intent);
            finish();
            break;
        case R.id.btnCancel:
            setResult(RESULT_CANCELED);
            finish();
            break;
    }
}
```

При запуске приложения на экране устройства появляется окно главного Activity с контактными данными. При нажатии кнопки **Edit Data** будет запущен дочерний Activity с текстовыми полями для редактирования данных. В зависимости от результата закрытия окна (кнопкой **Save** или **Cancel**), измененные данные будут прочитаны в родительском Activity и выведены на экран (рис. 12.8).

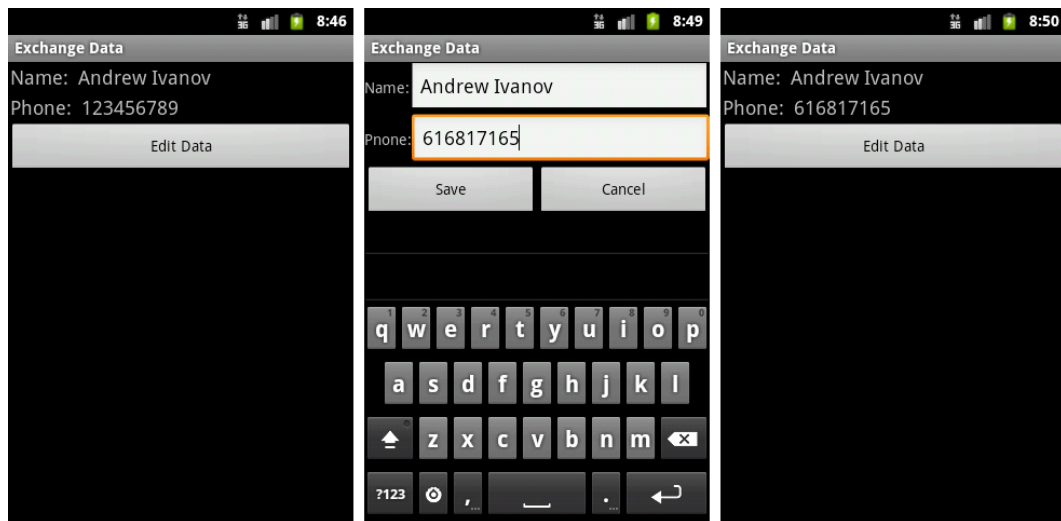


Рис. 12.8. Обмен данными между Activity

Вызов Activity из другого приложения

Для вызова Activity из другого приложения обычно используется неявный объект Intent. Сейчас мы создадим отдельное приложение, которое будет вызывать MainActivity приложения ContactEditor.

Сначала сделаем изменения в файле манифеста нашего приложения ContactEditor, добавив дополнительный фильтр Intent к Activity ContactListActivity, как показано в листинге 12.9.

Листинг 12.9. Изменения в файле манифеста приложения AndroidManifest.xml

```
...
<activity
    android:name=".ContactListActivity"
    android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.samples.app.contact.VIEW_CONTACTS" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
...
```

Этот фильтр нужен, чтобы система смогла разыскать и запустить необходимый Activity среди множества приложений, содержащих Activity, которые были установлены на этом устройстве.

Строка `action android:name="com.samples.app.contact.VIEW_CONTACTS"` определяет имя действия. Для вызова Activity ContactListActivity из другого приложения необходимо создать объект Intent и передать ему в качестве параметра строку `"com.samples.app.contact.VIEW_CONTACTS"`, определенную в Intent-фильтре компонента вызываемого приложения.

Далее создадим в Eclipse новый проект:

- ☐ **Project name** — ContactLauncher;
- ☐ **Application name** — Contact launcher sample;
- ☐ **Package name** — com.samples.app.contactlauncher;
- ☐ **Create Activity** — ContactLauncherActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch12_ContactLauncher.

Приложение будет представлять окно с одной кнопкой. Файл компоновки приложения приведен в листинге 12.10.

Листинг 12.10. Файл компоновки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">
    <Button
        android:id="@+id/btn_launch"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Launch Contact Editor"
        android:onClick="onClick"/>
</LinearLayout>
```

В коде класса `ContactLauncherActivity` в обработчике события кнопки будет создаваться объект `Intent` с действием `com.samples.app.contact.VIEW_CONTACTS`. Вызов метода `startActivity()` с созданным объектом `Intent` в качестве входного параметра запустит `ContactListActivity` из приложения для редактирования контактов. Код класса `ContactLauncherActivity` представлен в листинге 12.11.

Листинг 12.11. Файл класса окна приложения `ContactLauncherActivity.java`

```
package com.samples.app.contactlauncher;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class ContactLauncherActivity extends Activity
    implements View.OnClickListener {
    // Константа, идентифицирующая вызываемый Activity
    private final static String ACTION_VIEW_CONTACTS =
        "com.samples.contact.VIEW_CONTACTS";

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

    @Override
    public void onClick(View v) {
        // Вызов Activity приложения Contacts
        startActivity(new Intent(ACTION_VIEW_CONTACTS));
    }
}
```

Запустите проект на выполнение. При нажатии кнопки должен запуститься главный Activity приложения `ContactEditor` (рис. 12.9).

Таким образом можно запускать любой Activity, добавив соответствующие изменения в файл манифеста, аналогично листингу 12.9. Точно так же система Android позволяет запускать Activity посторонних приложений.

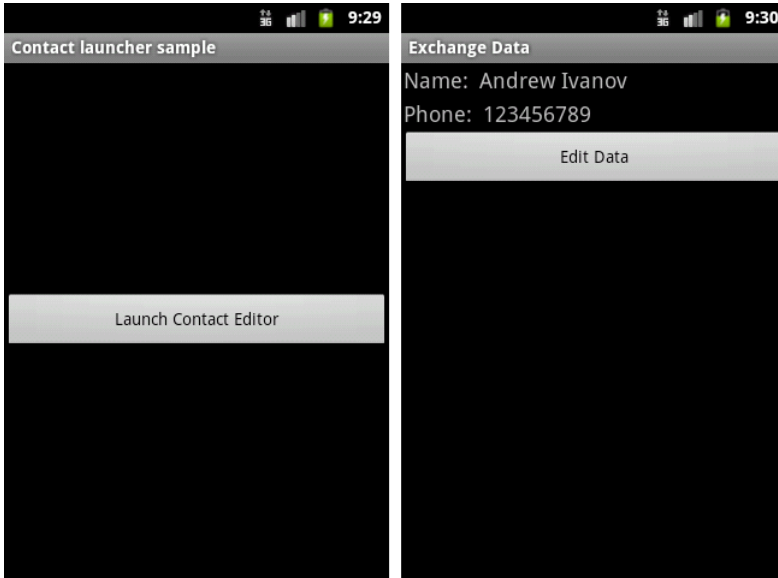


Рис. 12.9. Запуск Activity из другого приложения

Вызов стандартных Activity

Система Android и приложения, которые идут в комплекте с платформой, также используют объекты `Intent` для активизации определенных системой компонентов (например, различных приложений и служб при загрузке системы).

Для вызова системных компонентов в классе `Intent` определено множество констант `Standard Activity Actions`, определяющих действия для запуска стандартных Activity. Например, действие `INTENT_ACTION_MUSIC_PLAYER` инициализирует обращение по телефону и запускает Activity, представляющий собой медиаплеер. Это окно можно вызвать следующим образом:

```
Intent intent = new Intent("Intent.INTENT_ACTION_MUSIC_PLAYER");
startActivity(intent);
```

Кроме класса `Intent`, в других классах также определены действия для вызова специализированных Activity. Например, в классах `MediaStore` и `MediaStore.Audio.Media` определены константы действий для запуска окон с медиаплеером, поиска музыки и записи с микрофона. Также действия определены в классах `DownloadManager` для запуска Activity, управляющего загрузкой файлов через Интернет, `RingtoneManager` для запуска окна, устанавливающего режим звонка, и т. д.

Давайте теперь создадим приложение, вызывающее стандартные Activity. Например, сделаем вызов двух Activity:

- ☐ `Media.RECORD_SOUND_ACTION`;
- ☐ `MediaStore.INTENT_ACTION_MUSIC_PLAYER`.

Создайте в IDE Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `IntentActionsMedia`;
- ☐ **Application name** — `Standard Intent Actions`;
- ☐ **Package name** — `com.samples.app.intentactions`;
- ☐ **Create Activity** — `IntentActionsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch12_IntentActionsMedia`.

В файле компоновки окна `main.xml` будет две кнопки:

- ☐ для вызова окна записи звука (диктофона) с надписью **Record Sound** и идентификатором `bRecordSound`;
- ☐ для вызова окна медиаплеера с надписью **Music Player** и идентификатором `bMusicPlayer`.

Код файла `main.xml` приведен в листинге 12.12.

Листинг 12.12. Файл `main.xml`

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

    <Button
        android:text="Record Sound"
        android:id="@+id/bRecordSound"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:layout_width="fill_parent"/>

    <Button
        android:text="Music Player"
        android:id="@+id/bMusicPlayer"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:layout_width="fill_parent"/>

</LinearLayout>
```

В классе окна приложения `IntentActionsActivity` в теле обработчика события `onClick()` будет создаваться объект `Intent` с параметрами для вызова соответствующих стандартных `Activity`. Код класса `IntentActionsActivity` представлен в листинге 12.13.

Листинг 12.13. Файл класса окна приложения `IntentActionsActivity.java`

```
package cm.samples.app.intentactions;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.Media;
import android.view.View;

public class IntentActionsActivity extends Activity
    implements View.OnClickListener {

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

    @Override
    public void onClick(View v) {
        String action = "";

        switch (v.getId()) {
            case R.id.bRecordSound:
                action = Media.RECORD_SOUND_ACTION;
                break;
            case R.id.bMusicPlayer:
                action = MediaStore.INTENT_ACTION_MUSIC_PLAYER;
                break;
        }

        Intent intent = new Intent(action);
        startActivity(intent);
    }
}
```

После компиляции приложения и установки его на эмулятор мы можем запускать из приложения два стандартных `Activity`: диктофон для записи звука и медиаплеер. Внешний вид приложения и открываемых окон показан на рис. 12.10.

Однако большинство стандартных `Activity` просто так запустить нельзя, т. к. при запуске они требуют наличие соответствующих разрешений, которые мы будем рассматривать далее.

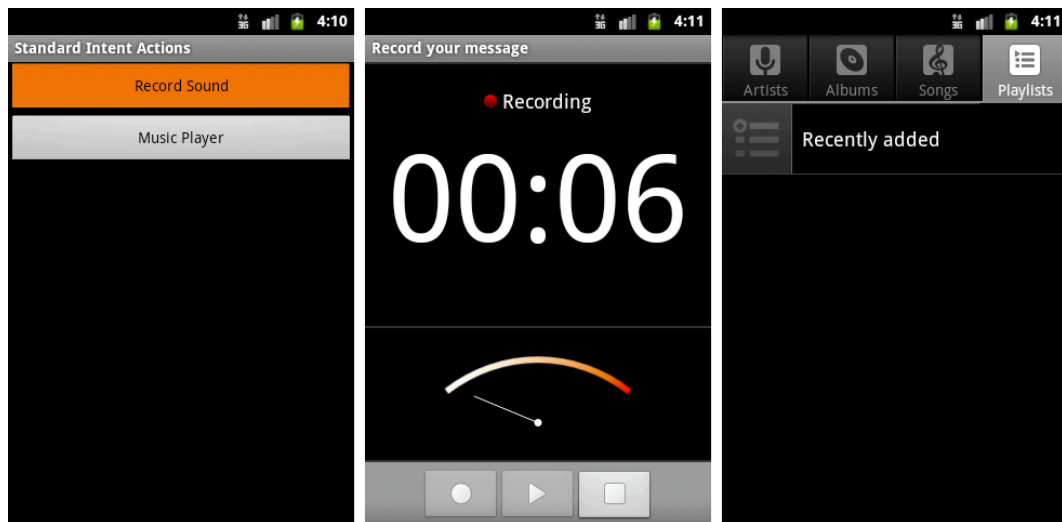


Рис. 12.10. Вызов стандартных Activity

Резюме

В этой главе мы рассмотрели важную тему: организацию процессов и взаимодействие компонентов в Android-приложениях с помощью объектов Intent. Мы изучили способы вызова из своего приложения стандартных системных Activity. Кроме того, мы рассмотрели передачу данных в вызываемый Activity и получение информации обратно из вызываемого Activity с использованием extra-параметров объектов Intent.

Хотя мы рассматривали только взаимодействие объектов Activity, для других компонентов приложения, Service и Broadcast Receiver, взаимодействие осуществляется аналогично.

В следующей главе мы изучим запуск посторонних компонентов приложений Android с помощью механизма разрешений.



ГЛАВА 13

Доступ к компонентам через разрешения

В системе Android, по умолчанию, доступ почти ко всем системным компонентам ограничен для внешних приложений. Поэтому если приложению требуется использовать какую-либо системную функциональность, например, необходимо послать сообщение SMS, подключиться к мобильному Интернету или к удаленному сетевому сервису, нужно задать для приложения соответствующие разрешения.

Вызов Activity с использованием разрешений

В предыдущей главе мы говорили, что для вызова системных компонентов в классе `Intent` определено множество констант `Standard Activity Actions`, определяющих действия для запуска стандартных Activity. Например, действие `ACTION_DIAL` инициализирует телефонный вызов и запускает Activity, представляющий собой окно для набора телефонного номера. Это окно можно вызвать следующим образом:

```
Intent intent = new Intent("Intent.ACTION_DIAL");
startActivity(intent);
```

Кроме класса `Intent`, в других классах тоже определены действия для вызова специализированных Activity. Также действия определены в классах `DownloadManager` для запуска Activity, управляющего загрузкой файлов через Интернет, `RingtoneManager` для запуска окна, устанавливающего режим звонка, и т. д.

Разрешение требуется только для действия `Intent.ACTION_DIAL`, для двух других действий разрешения устанавливать не нужно.

Давайте теперь создадим приложение, вызывающее несколько стандартных Activity. Например, сделаем вызов Activity наборной панели для вызова абонента:

```
Intent.ACTION_DIAL;
```

Создайте в IDE Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `IntentActions`;
- ☐ **Application name** — `Standard Intent Actions`;
- ☐ **Package name** — `com.samples.app.intentactions`;
- ☐ **Create Activity** — `IntentActionsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch13_IntentActions.

Итак, первым шагом нам надо объявить разрешение в манифесте приложения.

Установка разрешений в файле манифеста

Разрешение можно задать напрямую в коде файла манифеста приложения, добавив элемент `<uses-permission>` с атрибутом, содержащим имя нужного разрешения, или, что более удобно (поскольку при редактировании файла манифеста не работают всплывающие подсказки, а разрешений очень много), использовать для этого окно редактора файла манифеста, а именно его вкладку **Permissions**. На этой вкладке нажмите кнопку **Add** и в появившемся диалоговом окне выберите элемент **Uses Permission**, как показано на рис. 13.1.

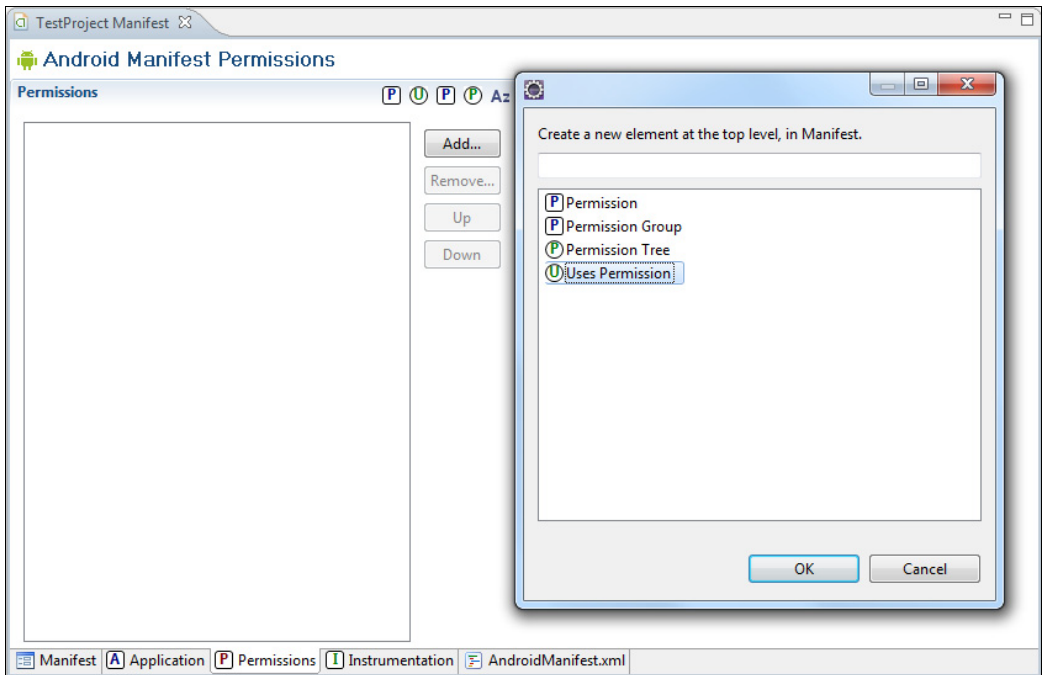


Рис. 13.1. Установка разрешений в файле AndroidManifest.xml

После этого надо выбрать тип разрешения. На панели **Attributes for Uses Permission** в выпадающем списке **Name** расположен полный список всех доступных разрешений, которые вы можете использовать у себя в приложении (рис. 13.2).

После этих операций в файл манифеста приложения будет добавлено разрешение `android.permission.CALL_PHONE`, как показано в листинге 13.1.

Листинг 13.1. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```

package="cm.samples.app.intentactions"
android:versionCode="1"
android:versionName="1.0">

<application android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity android:name=".IntentActionsActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

<uses-permission
    android:name="android.permission.CALL_PHONE">
</uses-permission>
</manifest>

```

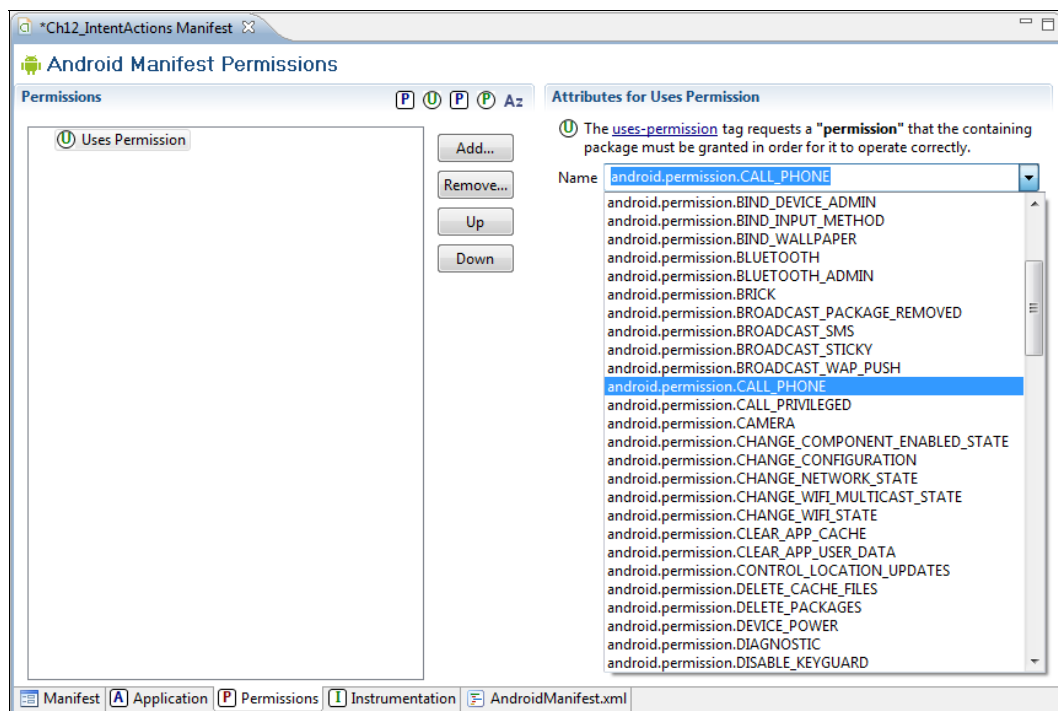


Рис. 13.2. Выбор необходимого разрешения для доступа к системным компонентам

В файл компоновки окна `main.xml` будет помещена кнопка для вызова окна набора телефонного номера с надписью **Phone Call** и идентификатором `bPhoneCall` (листинг 13.2).

Листинг 13.2. Файл main.xml

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

    <Button
        android:text="Phone Call"
        android:id="@+id/bPhoneCall"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:onClick="onClick"/>

</LinearLayout>
```

В классе окна приложения `IntentActionsActivity` в теле обработчика события `onClick()` будет создаваться объект `Intent` с параметрами для вызова соответствующих стандартных `Activity`. Код класса `IntentActionsActivity` представлен в листинге 13.7.

Листинг 13.3. Файл класса окна приложения IntentActionsActivity.java

```
package cm.samples.app.intentactions;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.provider.MediaStore;
import android.provider.MediaStore.Audio.Media;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class IntentActionsActivity extends Activity
    implements OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Intent.ACTION_DIAL);
        startActivity(intent);
    }
}
```

После компиляции приложения и установки его на эмулятор мы можем запускать из приложения стандартный Activity с ограниченным доступом — наборную панель для телефонного вызова абонента. Внешний вид приложения и открываемых окон показан на рис. 13.3.

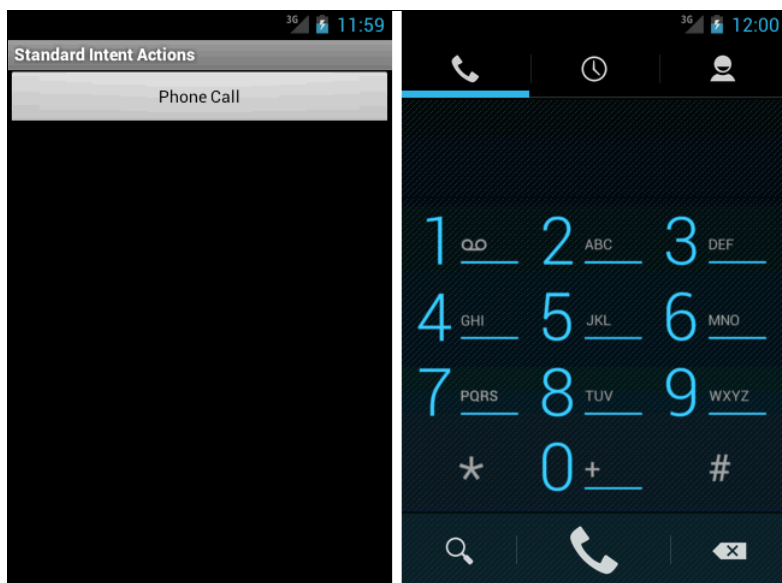


Рис. 13.3. Вызов стандартного Activity: наборной панели телефона с использованием разрешения

При проектировании приложения программисты довольно часто забывают подключить разрешения. Впрочем, это сразу обнаруживается при запуске приложения — при отсутствии нужного разрешения генерируется исключение.

Резюме

В этой главе мы кратко изучили механизмы разрешений. Система Android предоставляет нам гибкую систему защиты как для системных компонентов, так и для собственных разрабатываемых компонентов. При создании приложений в этой книге мы будем очень часто использовать разрешения при работе с системными компонентами Android и постоянно добавлять требуемые разрешения в файл манифеста приложения.

В следующей главе мы рассмотрим фрагменты — компонент, который появился относительно недавно, в версии Android 3.0.



ГЛАВА 14

Фрагменты

Фрагмент представляет часть пользовательского интерфейса в Activity и позволяет строить более гибкий пользовательский интерфейс. По сути, фрагменты представляют собой части Activity. Можно объединять несколько фрагментов в одном Activity и повторно использовать эти же фрагменты в других Activity.

Впервые фрагменты Android появились в Android 3.0 (API Level 11). С переходом системы Android на планшетные ПК с большими размерами и разрешением экрана, чем у мобильных устройств, появилась потребность в создании и поддержке более динамичных и гибких графических интерфейсов пользователя.

Классы фрагментов

В программе фрагмент похож на Activity — у него есть класс Java, и он загружает компоновку, определяющую свой пользовательский интерфейс, из XML-файла. XML-файл содержит такие же виджеты, как и для компоновки Activity. Основное отличие — класс, реализующий фрагмент, должен наследоваться от базового класса `Fragment`.

Помимо класса `Fragment`, существуют более специализированные классы фрагментов, такие как `DialogFragment`, `ListFragment`, `PreferenceFragment` и `WebViewFragment`. Иерархия классов, представляющих фрагменты, показана на рис. 14.1.

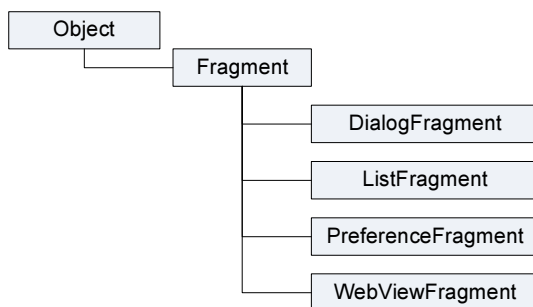


Рис. 14.1. Иерархия классов фрагментов

В этой главе мы рассмотрим работу с классами `Fragment` и `DialogFragment`. `ListFragment` мы рассмотрим, когда будем изучать способы отображения наборов данных (см. гла-

ву 17). Класс `PreferenceFragments` используется при сохранении пользовательских настроек приложения (см. глава 20), и здесь мы его рассматривать не будем.

Фрагмент имеет собственный жизненный цикл, похожий на цикл `Activity`, и соответственно имеет собственный набор методов, вызываемых при работе с этим компонентом:

- ❑ `onAttach()` — вызывается при привязке фрагмента к родительскому `Activity`;
- ❑ `onCreate()` — вызывается при начальном создании фрагмента (аналогично одноименному методу у `Activity`);
- ❑ `onCreateView()` — создает и возвращает иерархию всех `View`, связанных с данным фрагментом;
- ❑ `onActivityCreated()` — сообщает фрагменту, что родительский `Activity` завершил свой собственный метод `onCreate()`;
- ❑ `onStart()` — делает фрагмент видимым пользователю (после запуска такого же метода в родительском `Activity`);
- ❑ `onResume()` — вызывается, когда фрагмент вновь становится видимым (аналогичный метод есть у `Activity`);
- ❑ `onPause()` — фрагмент больше не взаимодействует с пользователем, потому что выполнение его родительского `Activity` также приостановлено, и система собирается запустить другой `Activity`;
- ❑ `onStop()` — фрагмент больше не видим потребителю, потому что его родительский `Activity` был перекрыт другим `Activity` и стал невидимым;
- ❑ `onDetach()` — отвязывает фрагмент от родительского `Activity`;
- ❑ `onDestroyView()` — позволяет фрагменту освободить ресурсы, связанные с его представлениями;
- ❑ `onDestroy()` — вызывается при окончательном освобождении ресурсов фрагмента.

Как вы видите, методы жизненного цикла фрагмента напоминают аналогичные методы для `Activity`, рассмотренные нами в главе 12, хотя и существуют некоторые отличия. В частности, у фрагмента есть дополнительные методы, вызываемые во время создания при привязке к родительскому `Activity` и при освобождении ресурсов, при отвязке фрагмента от родительского `Activity`.

Создание фрагментов

Создание фрагментов напоминает создание и добавление в приложение дополнительных `Activity`. Каждый фрагмент определяется в отдельном классе, производном от базового класса `Fragment`:

```
public class MyFragment extends Fragment {  
    ...  
}
```

Для фрагмента, так же как и для `Activity`, графический интерфейс пользователя определяется в XML-файле компоновки.

Для прорисовки графического интерфейса в классе фрагмента необходимо переопределить метод `onCreateView()`. При вызове этого метода ему передается в качестве пара-

метров объект `LayoutInflater` для обработки XML-компоновки фрагмента и объект типа `ViewGroup`, который определяет родительский контейнер (`Activity`), в который будет встраиваться фрагмент. Метод `onCreateView()` должен вернуть объект `View`, представляющий фрагмент, например, так:

```
@Override
public View onCreateView(LayoutInflater inflater,
    ViewGroup container, Bundle savedInstanceState) {

    return inflater.inflate(R.layout.myfragment, container, false);
}
```

Чтобы добавить фрагмент в компоновку родительского `Activity`, используется элемент `<fragment>`. Основным атрибутом для фрагмента является атрибут `android:name`, которому надо задать полное имя (вместе с именем пакета) класса фрагмента. Остальные атрибуты элемента `<fragment>` такие же, как и у обычного `View`:

```
<fragment
    android:name="com.samples.ui.fragment.MyFragment"
    android:id="@+id/fragment1"
    android:layout_width="fill_parent"
    android:layout_height="match_parent" />
```

В отличие от `Activity`, объявлять фрагмент в файле манифеста нет необходимости.

Давайте теперь реализуем создание фрагментов в практическом приложении. Откройте в IDE Eclipse новый проект и в мастере **Create New Project** введите следующие значения:

- ☐ **Project name** — `Fragments`;
- ☐ **Application name** — `Fragments Sample`;
- ☐ **Package name** — `com.samples.ui.fragment`;
- ☐ **Create Activity** — `FragmentsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch14_Fragments`.

В приложении будет один родительский `Activity` и два фрагмента: `Fragment1` и `Fragment2`. Для первого фрагмента создадим файл компоновки `fragment1.xml`, в котором будет находиться один виджет `TextView` с надписью **Fragment #1** и фоном серого цвета. Код файла `fragment1.xml` представлен в листинге 14.1.

Листинг 14.1. Файл компоновки фрагмента `fragment1.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#EEEEEE"
    android:gravity="center" >
```

```

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Fragment #1"
    android:textSize="25sp"
    android:textColor="#000000"/>
</LinearLayout>

```

В классе `Fragment1` будет присутствовать только метод `onCreateView()`, как показано в листинге 14.2.

Листинг 14.2. Файл класса фрагмента `Fragment1.java`

```

package com.samples.ui.fragment;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class Fragment1 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        // Возвращаем объект View, представляющий фрагмент
        return inflater.inflate(
            R.layout.fragment1, container, false);
    }
}

```

Также в каталог `res/layout/` добавьте еще один новый файл компоновки для второго фрагмента и назовите его `fragment2.xml`. В этом фрагменте будет тоже находиться один виджет `TextBox`, но темно-серого цвета, чтобы его отличать от первого фрагмента. Код для файла компоновки второго фрагмента показан в листинге 14.3.

Листинг 14.3. Файл компоновки фрагмента `fragment2.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#AAAAAA"
    android:gravity="center">
    <TextView
        android:layout_width="wrap_content"

```

```
        android:layout_height="wrap_content"
        android:text="Fragment #2"
        android:textSize="25sp"
        android:textColor="#000000"/>
</LinearLayout>
```

Класс второго фрагмента `Fragment2` в целом аналогичен классу `Fragment1` и представлен в листинге 14.4.

Листинг 14.4. Файл класса фрагмента `Fragment2.java`

```
public class Fragment2 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(
            R.layout.fragment2, container, false);
    }
}
```

В файл компоновки родительского `Activity` `main.xml` добавьте оба созданных фрагмента. Код файла компоновки родительского `Activity` показан в листинге 14.5.

Листинг 14.5. Файл компоновки родительского `Activity` `main.xml`

```
<?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="horizontal">
    <fragment
        android:name="com.samples.ui.fragment.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="fill_parent"
        android:layout_height="match_parent" />
    <fragment
        android:name="com.samples.ui.fragment.Fragment2"
        android:id="@+id/fragment2"
        android:layout_weight="1"
        android:layout_width="fill_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

Запустите проект на выполнение. Если вы все сделали правильно, на экране отобразятся два фрагмента, расположенных внутри родительского `Activity`, как представлено на рис. 14.2.

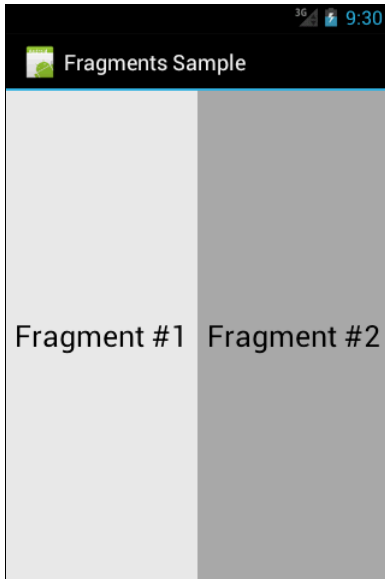


Рис. 14.2. Activity с двумя дочерними фрагментами

Динамическое добавление фрагментов

Главное удобство использования фрагментов заключается в легкости их добавления и удаления во время выполнения программы, что позволяет создавать очень гибкий пользовательский интерфейс для приложений.

Чтобы добавить фрагменты в Activity, сначала надо получить экземпляр класса `FragmentManager`:

```
FragmentManager fragmentManager = getFragmentManager();
```

Добавление фрагмента должно производиться в транзакции. Для этого используется класс `FragmentTransaction`. У этого класса есть набор методов для запуска и сохранения транзакции — `beginTransaction()` и `commit()`. Для добавления и удаления фрагментов в классе `FragmentTransaction` определены методы `add()` и `remove()`. Например, добавить фрагмент в родительский Activity можно следующим образом:

```
Fragment1 fragment1 = new Fragment1();
FragmentTransaction transaction = fragmentManager.beginTransaction();
fragmentTransaction.add(R.id.fragmentLayout, fragment2);
// После добавления обязательно надо сохранить изменения
fragmentTransaction.commit();
```

Используем теперь такой способ в реальном приложении. Создайте новый проект:

- ☐ **Project name** — `DynamicFragment`;
- ☐ **Application name** — `Dynamic Fragments`;
- ☐ **Package name** — `com.samples.ui.dynamicfrag`;
- ☐ **Create Activity** — `DynamicFragmentsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch14_DynamicFragment.

Для данного проекта мы будем применять фрагменты, созданные в предыдущем проекте (классы `Fragment1` и `Fragment2` и их компоновку, листинги 14.1—14.4). В файле компоновки для родительского `Activity` будет две кнопки для добавления фрагментов и дочернее представление `LinearLayout`, в которое мы будем "складывать" динамические фрагменты (листинг 14.6).

Листинг 14.6. Файл компоновки родительского Activity main.xml

```
<?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" >

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="Add Fragment1" />

        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text=" Add Fragment2" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/fragmentLayout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

    </LinearLayout>

</LinearLayout>
```

В классе родительского `Activity` в обработчике событий кнопок мы будем добавлять соответственно экземпляры классов `Fragment1` и `Fragment2`, используя `FragmentManager`. Полный код класса представлен в листинге 14.7.

Листинг 14.7. Файл класса DynamicFragmentsActivity.java

```
package com.samples.ui.dynamicfragments;

import android.app.Activity;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

public class DynamicFragmentsActivity extends Activity implements OnClickListener {

    private FragmentManager fragmentManager;
    private FragmentTransaction fragmentTransaction;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        fragmentManager = getFragmentManager();
    }

    @Override
    public void onClick(View v) {
        fragmentTransaction =
            fragmentManager.beginTransaction();
        switch (v.getId()) {
            case R.id.button1:
                Fragment1 fragment1 = new Fragment1();
                fragmentTransaction.add(
                    R.id.fragmentLayout, fragment1);
                break;
            case R.id.button2:
                Fragment2 fragment2 = new Fragment2();
                fragmentTransaction.add(
                    R.id.fragmentLayout, fragment2);
                break;
        }
        fragmentTransaction.commit();
    }
}
```

После компиляции и развертывания приложения, поочередно нажимая кнопки для создания фрагментов, на родительский Activity можно поместить практически любое количество фрагментов обоих типов (рис. 14.3).

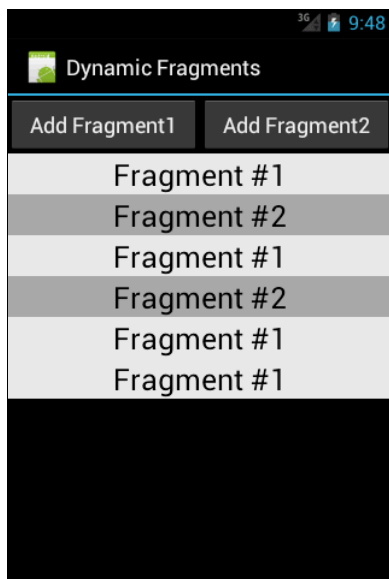


Рис. 14.3. Фрагменты, добавленные во время выполнения программы

Передача данных между фрагментами

Так как Activity может содержать набор фрагментов, возникает вопрос — как обеспечить обмен данными между фрагментами и контейнерным Activity.

Передавать данные между фрагментами проще, чем между отдельными Activity. При сборке проекта в файле R.java будут уже созданы все идентификаторы виджетов как родительского Activity, так и виджетов, входящих в состав фрагментов. Поскольку фрагменты встраиваются в Activity, надо сначала получить доступ к родительскому Activity — использовать вызов уже известного вам по множеству созданных приложений метода `findViewById()` класса `Activity`:

```
TextView text = (TextView) getActivity().findViewById(R.id.text);
```

Теперь напомним пример приложения, в котором будет происходить чтение данных введенных в фрагменте. Создайте в Eclipse новый проект:

- ☐ **Project name** — `FragmentsChangeData`;
- ☐ **Application name** — `FragmentsChangeData`;
- ☐ **Package name** — `com.samples.ui.fragmentschangedata`;
- ☐ **Create Activity** — `FragmentsChangeDataActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch14_FragmentsChangeData`.

В файл компоновки первого фрагмента добавим текстовое поле, в которое будем во время выполнения вводить произвольный текст. Код файла `fragment1.xml` представлен в листинге 14.8.

Листинг 14.8. Файл компоновки фрагмента fragment1.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#EEEEEE"
    android:gravity="center" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment #1"
        android:textSize="25sp"
        android:textColor="#000000"/>
    <EditText
        android:id="@+id/edit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textMultiLine"
        android:textColor="#000000"/>
</LinearLayout>

```

В файле компоновки второго фрагмента будет кнопка для чтения данных и виджет `TextView` для вывода прочитанной информации. Код файла `fragment2.xml` представлен в листинге 14.9.

Листинг 14.9. Файл компоновки фрагмента fragment2.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#AAAAAA"
    android:gravity="center">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment #2"
        android:textSize="25sp"
        android:textColor="#000000"/>
    <Button
        android:id="@+id/bGetText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#000000"
        android:text="Get Text" />

```

```
<TextView
    android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#000000"/>
</LinearLayout>
```

В класс, представляющий первый фрагмент, ничего добавлять не будем. Этот класс будет полностью аналогичен листингу 14.2.

В класс второго фрагмента в метод `onStart()` добавим код для чтения данных из первого фрагмента, как показано в листинге 14.10.

Листинг 14.10. Файл класса фрагмента `Fragment2.java`

```
package com.samples.ui.fragmentschangedata;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class Fragment2 extends Fragment
    implements View.OnClickListener {

    private TextView text;
    private EditText edit;
    private Button bGet;

    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {

        return inflater.inflate(
            R.layout.fragment2, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();

        text = (TextView) getActivity().findViewById(R.id.text);
        edit = (EditText) getActivity().findViewById(R.id.edit);
        bGet = (Button) getActivity().findViewById(R.id.bGetText);

        bGet.setOnClickListener(this);
    }
}
```

```

@Override
public void onClick(View v) {
    text.setText(edit.getText());
}
}

```

Скомпилируйте и запустите приложение. Если в текстовом поле первого фрагмента ввести текст, этот текст можно будет прочитать во втором фрагменте, нажав кнопку **Get Text**, как показано на рис. 14.4.

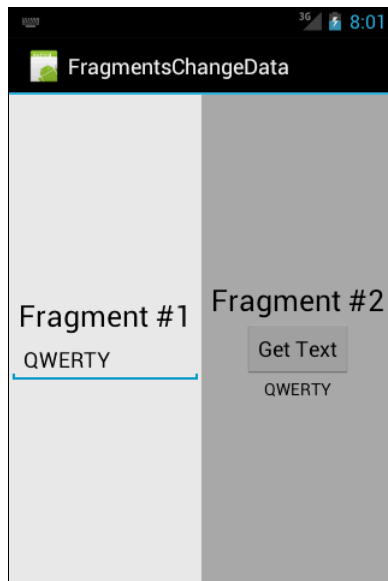


Рис. 14.4. Передача данных между фрагментами

DialogFragment

Еще один тип фрагмента, который предоставляет нам Android, — это `DialogFragment` (диалоговый фрагмент). `DialogFragment` появляется сверху родительского `Activity` и является модалным.

Для создания класса, представляющего этот диалог, необходимо использовать базовый класс `DialogFragment`:

```

public class Fragment1 extends DialogFragment {
    ...
}

```

Далее в этом классе нам нужно переопределить метод `onCreateDialog()`, аналогичный одноименному методу, который вызывается при создании обычных диалоговых окон, описанных в *главе 10*.

В классе, где будет происходить вызов диалога, надо будет создать экземпляр класса `Fragment1`, а затем вызвать диалог методом `show()` класса `DialogFragment`. В метод `show()` надо добавить два параметра: объект класса `FragmentManager`, который можно получить вызовом метода `getFragmentManager()`, и идентификатор диалога — строковую константу. Например, можно написать так:

```

Fragment1 dialogFragment = new Fragment1();
dialogFragment.show(getFragmentManager(), "dialog");

```

Для того чтобы в родительском Activity можно было поймать события нажатия на кнопки диалога, их необходимо подсоединить в методе `onCreateDialog()`, в теле обработчиков событий кнопок диалога. Например, для кнопки подтверждения диалога надо подсоединить метод `doPositiveClick()`:

```
public void onClick(DialogInterface dialog, int whichButton) {
    ((DialogFragmentsActivity) getActivity()).doPositiveClick();
}
```

А для кнопки закрытия диалога — метод `doNegativeClick()`:

```
public void onClick(DialogInterface dialog, int whichButton) {
    ((DialogFragmentsActivity) getActivity()).doNegativeClick();
}
```

В классе, представляющем родительский Activity, надо будет реализовать методы `doPositiveClick()` и `doNegativeClick()`, например, можно закрыть приложение:

```
public void doPositiveClick() {
    // закрываем приложение
    this.finish();
}

public void doNegativeClick() {
    // возврат в Activity, ничего не делаем
}
```

В общем создание диалогового фрагмента очень похоже на создание обычного диалога, поэтому создадим проект и уже по ходу написания кода вы увидите все отличия диалогового фрагмента и обычного диалога. Создайте в Eclipse новый проект:

- ☐ **Project name** — `DialogFragments`;
- ☐ **Application name** — `DialogFragments`;
- ☐ **Package name** — `com.samples.dialogfragments`;
- ☐ **Create Activity** — `DialogFragmentsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch14_DialogFragments`.

В классе, представляющем диалоговый фрагмент, определим конструктор, принимающий один строковый параметр — заголовок диалогового окна, и переопределим метод `onCreateDialog()`. Диалог будет иметь две кнопки: **OK** и **Cancel**. При нажатии на кнопку **OK** приложение будет закрываться, при нажатии на кнопку **Cancel** закроется диалог, а управление вернется обратно в родительский Activity.

Код класса `Fragment1` показан в листинге 14.11.

Листинг 14.11. Файл класса `Fragment1.java`

```
package com.samples.dialogfragments;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
```

```

import android.content.DialogInterface;
import android.os.Bundle;

public class Fragment1 extends DialogFragment {

    public Fragment1(String title) {
        Bundle args = new Bundle();
        args.putString("title", title);
        setArguments(args);
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        String title = getArguments().getString("title");
        return new AlertDialog.Builder(getActivity())
            .setIcon(R.drawable.android3d)
            .setTitle(title)
            .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                public void onClick(
                    DialogInterface dialog, int whichButton) {
                    ((DialogFragmentsActivity)getActivity()).doPositiveClick();
                }
            })
            .setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
                public void onClick(
                    DialogInterface dialog, int whichButton) {
                    ((DialogFragmentsActivity)getActivity()).doNegativeClick();
                }
            }).create();
    }
}

```

В коде класса родительского Activity нам надо в обработчике события нажатия кнопки создать экземпляр класса `Fragment1` и вызвать диалог методом `show()`. Также необходимо определить два метода, `doPositiveClick()` и `doNegativeClick()`, для обработки события нажатия кнопок диалога. В теле метода `doPositiveClick()` сделаем выход из приложения, используя вызов метода `finish()`, а тело метода `doNegativeClick()` оставим пустым.

Полный код для класса окна приложения представлен в листинге 14.12.

Листинг 14.12. Файл класса главного окна приложения `DialogFragmentsActivity.java`

```

package com.samples.dialogfragments;

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

public class DialogFragmentsActivity extends Activity
    implements View.OnClickListener {
    Fragment1 dialogFragment;

```

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

@Override
public void onClick(View v) {
    dialogFragment = new Fragment1(
        "Are you sure you want to exit?");
    dialogFragment.show(getFragmentManager(), "dialog");
}

public void doPositiveClick() {
    this.finish();
}

public void doNegativeClick() {
}
}
```

После компиляции и запуска приложения на экран будет выведено окно с единственной кнопкой **Call Dialog**, при нажатии на которую будет отображаться модальный диалог, созданный на основе `DialogFragment` (рис. 14.5).

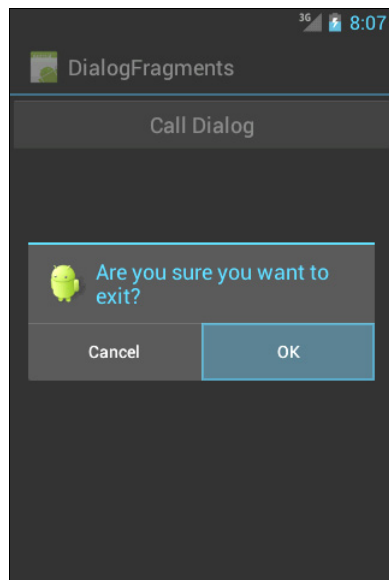
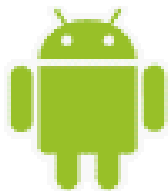


Рис. 14.5. Использование `DialogFragment`

Резюме

В этой главе мы познакомились с фрагментами, применяемыми в основном как компонент для построения динамических графических интерфейсов пользователя для экранов с большими размерами и разрешением.

В следующей главе мы будем изучать важную тему — использование ресурсов различного типа в приложениях.



ЧАСТЬ III

Ресурсы, графика и обработка данных

Глава 15. Использование ресурсов

Глава 16. Файловая система и карта памяти

Глава 17. Адаптеры данных и компоненты для отображения данных

Глава 18. База данных SQLite

Глава 19. Content Provider

Глава 20. Сохранение пользовательских настроек

Глава 21. Локализация приложений

Глава 22. Графика

Глава 23. Создание анимации



ГЛАВА 15

Использование ресурсов

Ресурсы и активы — неотъемлемая часть Android-приложения. Это внешние элементы, которые включаются в приложение: изображения, аудио, видео, строки, компоновки, темы и т. д. Каждое приложение содержит каталог для ресурсов `res/` и каталог для активов `assets/`.

Ресурсы используются чаще, чем активы. Реальное различие между ресурсами и активами заключается в следующем:

- ❑ информация в каталоге ресурсов будет доступна в приложении через класс `R`, который автоматически генерируется средой разработки. То есть хранение файлов и данных в ресурсах (в каталоге `res/`) делает их легкодоступными для использования в коде программы;
- ❑ для чтения информации, помещенной в каталог активов `assets/` (необработанный формат файла), необходимо использовать `AssetManager` для чтения файла как потока байтов.

В этой главе вы получите информацию о стандартных ресурсах, которые обычно используются в Android-приложении, и о том, как обращаться к ним в коде программы.

Доступные типы ресурсов

В предыдущих главах книги вы уже познакомились с загрузкой строковых и графических ресурсов в приложение, однако существует еще много других типов ресурсов, которые можно применять в разработке. Android предоставляет возможность использовать в приложениях следующие типы ресурсов.

- ❑ *Простые значения* — ресурсы могут быть выражены как строки с использованием различных форматирований, чтобы однозначно указать тип создаваемого ресурса.
- ❑ *Цвет* — кодировка цвета в шестнадцатеричном выражении определяет значение RGB и alpha-канал (цвет мы рассматривали в *разд. "TextView" главы 6*).
- ❑ *Строки с дополнительным форматированием*. Можно добавлять дополнительное форматирование для строки, используя три стандартных HTML-тега: `<b>`, `<i>` и `<u>`. Методы, которые будут обрабатывать строковые ресурсы с HTML-форматированием, должны уметь обрабатывать эти теги.

- ❑ *Графические ресурсы.* Есть множество видов графических ресурсов, которые можно создавать и использовать в приложении:
 - файлы растровой графики в различных форматах: PNG (который наиболее предпочтителен для использования), JPEG и GIF;
 - `PaintDrawable` — графические объекты, которые представляют собой прямоугольник с закругленными углами заданного цвета. Этот элемент может быть определен в любом из файлов внутри каталога `res/values/`;
 - графика `NinePatch` — изображение PNG, которое можно растягивать. В Android графика `NinePatch` используется для прорисовки виджетов — кнопок, закладок и т. д.
- ❑ *Анимация* — система Android может выполнить простую анимацию на графике или на серии графических изображений.
- ❑ *Меню* — меню и подменю также могут быть определены как XML-ресурсы и загружены в приложение.
- ❑ *XML-файлы компоновки.* Этот вид ресурса вы уже хорошо изучили в предыдущих главах книги.
- ❑ *Стили* — это один или более атрибутов, относящихся к одному элементу. Стилль может быть применен как атрибут к элементу в файле компоновки.
- ❑ *Темы* — это один или более атрибутов, относящихся к целому экрану. Например, можно применить определенную в Android готовую тему `Theme.dialog` к `Activity`, разрабатываемому для плавающих диалоговых окон.

Создание ресурсов

Как уже было сказано в *главе 3*, все ресурсы создаются и сохраняются в предопределенных подкаталогах в каталоге `res/` проекта. Android SDK имеет утилиту `aapt` для компиляции ресурсов.

Далее приводится список каталогов и вложенных файлов для каждого типа ресурса.

- ❑ `res/anim/` — файлы анимации.
- ❑ `res/drawable/` — графика.
- ❑ `res/layout/` — XML-файлы для схем компоновки.
- ❑ `res/values/` — XML-файлы, которые могут быть скомпилированы во многие виды ресурсов. В отличие от других каталогов, `res/` может содержать любое число файлов. При создании файлов ресурсов надо обязательно соблюдать соглашение об именовании файлов ресурсов по типу элементов, определенных в них:
 - `arrays.xml` — массивы;
 - `colors.xml` — значения цвета для графики и строк текста;
 - `dimens.xml` — размерности;
 - `strings.xml` — строковые значения;
 - `styles.xml` — стили.

- ❑ `res/xml/` — произвольные XML-файлы, которые могут загружаться во время выполнения.
- ❑ `res/raw/` — произвольные файлы, предназначенные для копирования непосредственно на устройство.

Ссылки на ресурсы

Значение атрибута XML-элемента (или ресурса) может также быть ссылкой на ресурс. Такие ссылки часто используются в файлах компоновки для строк и изображений (которые находятся в другом файле), хотя ссылкой может быть любой тип ресурса, например цветовая кодировка или целочисленные значения.

Например, если мы имеем цветовые ресурсы, мы можем записать файл компоновки, который устанавливает значение цвета для текста, находящегося в одном из этих ресурсов:

```
android:textColor="@color/opaque_red"
```

Обратите внимание, здесь на использование префикса `@` для того, чтобы ввести ссылку ресурса — текст после этого префикса — имя ресурса. В этом случае мы не должны были указывать пакет, потому что мы ссылаемся на ресурс в нашем собственном пакете. Для ссылки на системный ресурс мы должны записать:

```
android:textColor="@android:color/opaque_red"
```

Использование ресурсов в коде программы

Во время компиляции генерируется класс `R`, который является оболочкой ресурсов и содержит идентификаторы всех ресурсов в программе. Класс `R` имеет несколько вложенных классов, один для каждого типа ресурса, поддерживаемого системой Android, и для которого в проекте существует файл ресурса. Класс `R` может содержать следующие вложенные классы:

- ❑ `R.anim` — идентификаторы для файлов из каталога `res/anim/`;
- ❑ `R.array` — идентификаторы для файлов из каталога `res/values/`;
- ❑ `R.bool` — идентификаторы для битовых массивов в файлах `arrays.xml` из каталога `res/values/`;
- ❑ `R.color` — идентификаторы для файлов `colors.xml` из каталога `res/values/`;
- ❑ `R.dimen` — идентификаторы для файлов `dimens.xml` из каталога `res/values/`;
- ❑ `R.drawable` — идентификаторы для файлов из каталога `res/drawable/`;
- ❑ `R.id` — идентификаторы представлений и групп представлений для файлов XML-компоновки из каталога `res/layout/`;
- ❑ `R.integer` — идентификаторы для целочисленных массивов в файлах `arrays.xml` из каталога `res/values/`;
- ❑ `R.layout` — идентификаторы для файлов компоновки из каталога `res/layout/`;

- ❑ `R.raw` — идентификаторы для файлов из каталога `res/raw/`;
- ❑ `R.string` — идентификаторы для файлов `strings.xml` из каталога `res/values/`;
- ❑ `R.style` — идентификаторы для файлов `styles.xml` из каталога `res/values/`;
- ❑ `R.xml` — идентификаторы для файлов из каталога `res/xml/`.

Каждый вложенный класс содержит один или несколько идентификаторов для скомпилированных ресурсов, которые используются в коде программы для загрузки ресурса. Далее приведен синтаксис для обращения к ресурсу:

```
R.resource_type.resource_name
```

В приложениях также можно использовать системные ресурсы. Все системные ресурсы определены под классом `android.R`. Например, стандартный значок приложения на экране можно отобразить так:

```
android.R.drawable.sym_def_app_icon
```

Или загрузить стандартный стиль:

```
android.R.style.Theme_Black
```

Загрузка простых типов из ресурсов

Загружать простые типы из внешних ресурсов в Android очень легко. Для этого используется метод `getResources()`, который возвращает объект класса `Resources`. Этот класс имеет большой набор методов для чтения из ресурсов данных самого разнообразного типа, например, `getInteger()` — для получения целочисленных данных, `getText()` — для текстовых данных и т. д. Есть также группа методов для чтения массивов данных — `getStringArray()`, `getTextArray()`, а также данных в XML-формате — `getXml()`.

Чтобы изучить на практике загрузку простых типов ресурсов и их вызов из программного кода, создадим в Eclipse новый проект:

- ❑ **Project name** — `SimpleValues`;
- ❑ **Application name** — `Resource Sample`;
- ❑ **Package name** — `com.samples.res.simplevalues`;
- ❑ **Create Activity** — `SimpleValuesActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch15_SimpleValues`.

При создании проекта в каталоге `res/values/` мастер генерирует единственный файл `strings.xml`. Добавьте в этот каталог следующие XML-файлы:

- ❑ `arrays.xml` — для массивов строк и целочисленных значений;
- ❑ `colors.xml` — для значений цвета;
- ❑ `dimen.xml` — для значений размеров текста;
- ❑ `drawables.xml` — для графических примитивов.

Заполните эти файлы разнообразными данными, как показано в листингах 15.1—15.5.

Листинг 15.1. Файл ресурсов res/values/arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="names">
        <item>Andrew</item>
        <item>Helen</item>
        <item>Jack</item>
    </string-array>

    <integer-array name="digits">
        <item>1999</item>
        <item>2002</item>
        <item>2010</item>
    </integer-array>
</resources>
```

Листинг 15.2. Файл ресурсов res/values/colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="textColor">#0000FF</color>
    <color name="backgroundColor">#FF0000</color>
</resources>
```

Листинг 15.3. Файл ресурсов res/values/dimen.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="textPointSize">18pt</dimen>
</resources>
```

Листинг 15.4. Файл ресурсов res/values/drawables.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <drawable name="grayDrawable">#DDD</drawable>
</resources>
```

Листинг 15.5. Файл ресурсов res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Resource Sample</string>
    <string name="some_text">Some Text</string>
</resources>
```

В файле компоновки создадим структуру с текстовыми полями для отображения на экране загружаемых ресурсов. Файл компоновки приложения `main.xml` приведен в листинге 15.6.

Листинг 15.6. Файл компоновки `main.xml`

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

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:layout_height="wrap_content"
            android:text="String array:"
            android:layout_width="wrap_content"
            android:paddingRight="5px"
            android:textColor="@color/textColor"/>
        <TextView
            android:id="@+id/text_strings"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:padding="5px"
            android:textColor="@color/textColor"/>
        </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView
            android:layout_height="wrap_content"
            android:text="Int array:"
            android:layout_width="wrap_content"
            android:paddingRight="20px"
            android:padding="5px"
            android:textColor="@color/textColor"/>
        <TextView
            android:id="@+id/text_digits"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textColor="@color/textColor"/>
        </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
```



```

<TextView
    android:id="@+id/text_style"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</LinearLayout>

</LinearLayout>

```

В классе, реализующем окно приложения `SimpleValuesActivity`, показаны варианты загрузки ресурсов с использованием методов класса `Resources` в зависимости от их типа с помощью сгенерированного плагином класса `R`. Полный код класса приведен в листинге 15.7.

Листинг 15.7. Файл класса окна приложения `SimpleValuesActivity.java`

```

package com.samples.res.simplevalues;

import android.app.Activity;
import android.graphics.drawable.ColorDrawable;
import android.os.Bundle;
import android.view.Window;
import android.widget.TextView;

public class SimpleValuesActivity extends Activity {

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

        final TextView textStrings =
            (TextView)findViewById(R.id.text_strings);

        // Загрузка массива строк из res/values/arrays.xml
        String[] names = getResources().getStringArray(R.array.names);
        for(int i = 0; i < names.length; i++) {
            textStrings.append("Name[" + i + "]: " + names[i] + "\n");
        }

        final TextView textDigits =
            (TextView)findViewById(R.id.text_digits);

        // Загрузка массива целых чисел из res/values/arrays.xml
        int[] digits = getResources().getIntArray(R.array.digits);
        for(int i = 0; i < digits.length; i++) {
            textDigits.append("Digit[" + i + "]: " + digits[i] + "\n");
        }

        // Текстовое поле с атрибутами, загружаемыми из ресурсов
        final TextView textStyle = (TextView)findViewById(R.id.text_style);
    }
}

```

```

// Загрузка текста из res/values/strings.xml
textStyle.setText(
    getResources().getText(R.string.some_text));

// Загрузка цвета текста из res/values/colors.xml
textStyle.setTextColor(
    getResources().getColor(R.color.textColor));

// Загрузка размера текста из res/values/dimen.xml
textStyle.setTextSize(
    getResources().getDimension(R.dimen.textPointSize));

// Загрузка цвета для фона из res/values/colors.xml
textStyle.setBackgroundColor(
    getResources().getColor(R.color.backgroundColor));

// Загрузка цвета фона окна Activity
// из res/values/drawables.xml
Window w = this.getWindow();

w.setBackgroundDrawable(
    (ColorDrawable) getResources().getDrawable(
        R.drawable.grayDrawable));
}
}

```

После компиляции и запуска проекта внешний вид приложения должен быть таким, как на рис. 15.1.



Рис. 15.1. Загрузка простых типов ресурсов

Загрузка файлов произвольного типа

Каталог `res/raw/` предназначен для хранения файлов произвольного типа, которые сжимаются при компиляции проекта и копируются на устройство в "нормальном" виде. Например, в этот каталог можно поместить какой-либо текстовый файл, который будет загружаться в приложение.

При загрузке файла с произвольным типом данных используются стандартные `java`-классы для работы с потоками ввода-вывода из пакета `java.io`. Для файлов, размещенных в каталоге `res/raw/`, среда Eclipse генерирует идентификатор ресурса в файле `R.java`, и в коде программы к ним можно обращаться не по имени файла, а по идентификатору ресурса.

Чтобы показать на практике вариант использования загрузки файлов из каталога `res/raw/`, создадим новый проект:

- ☐ **Project name** — `RawFiles`;
- ☐ **Application name** — `Raw Resources Sample`;
- ☐ **Package name** — `com.samples.res.raw`;
- ☐ **Create Activity** — `RawActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch15_RawFiles`.

В файле компоновки `main.xml` создайте единственный элемент `TextView`, как показано в листинге 15.8.

Листинг 15.8. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        android:id="@+id/text"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:layout_width="wrap_content"
        android:gravity="left|center"/>

</LinearLayout>
```

Загрузка файла из ресурсов производится с использованием обычных манипуляций, применяемых при вводе-выводе файлов в `java`-программах. Сначала создается объект `InputStream`, который можно получить с помощью метода `openRawResource()` класса `Resources`. Затем данные читаются в буфер и выводятся в текстовое поле.

Код класса `Activity`, в котором производится загрузка ресурса, приведен в листинге 15.9.

Листинг 15.9. Файл класса окна приложения RawActivity.java

```
package com.samples.res.raw;

import java.io.DataInputStream;
import java.io.InputStream;

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

public class RawActivity extends Activity {

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

        final TextView text = (TextView)findViewById(R.id.text);

        // Создаем объект InputStream
        InputStream iFile = getResources().openRawResource(R.raw.file);
        try {
            // Создаем буфер для чтения данных
            StringBuffer sBuffer = new StringBuffer();

            // Преобразуем объект InputStream в DataInputStream,
            // специализированный для чтения потоков данных
            DataInputStream dataIO = new DataInputStream(iFile);
            String strLine = null;

            // Читаем данные в буфер
            while((strLine=dataIO.readLine()) != null){
                sBuffer.append(strLine + "\n");
            }

            // После чтения обязательно закрываем объекты DataInputStream
            // и InputStream
            dataIO.close();
            iFile.close();

            text.setText(sBuffer.toString());
        }
        catch (Exception e) {
            text.setText("Error loading file: " + e.getMessage());
        }
    }
}
```

Сделаем компиляцию и запустим проект на выполнение. Приложение должно загрузить текстовый файл из каталога ресурсов и вывести на экран его содержимое, как показано на рис. 15.2.

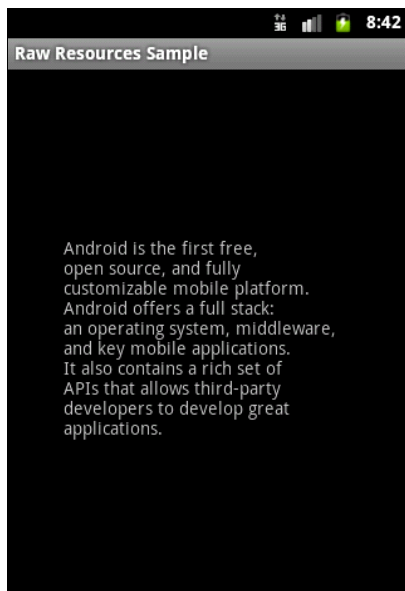


Рис. 15.2. Загрузка ресурсов из файла в каталоге `res/raw/`

Создание меню в XML

В предыдущих приложениях мы создавали для окон Activity схемы компоновки в виде XML-документов, сохраняемых в каталоге `res/layouts/`. Аналогично можно определять меню в XML-документах и загружать их в приложение. Файлы меню сохраняются в отдельном каталоге `res/menu/`.

В XML-файле меню есть три элемента:

- ☐ `<menu>` — корневой элемент файла меню;
- ☐ `<group>` — контейнерный элемент, определяющий группу меню;
- ☐ `<item>` — элемент, определяющий пункт меню.

Элементы `<item>` и `<group>` могут быть дочерними элементами `<group>`. Конечно, корневой узел любого файла должен быть элементом меню.

Как пример приложения с загрузкой меню из XML мы определим то же самое меню со значками, созданное нами в разд. "Меню со значками" главы 11. Создадим в Eclipse новый проект и в диалоге **Create New Project** заполним поля:

- ☐ **Project name** — `IconMenuApp`;
- ☐ **Application name** — `Load Menu from XML resource`;
- ☐ **Package name** — `com.samples.res.resmenuxml`;
- ☐ **Create Activity** — `ResMenuXmlActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch15_MenuXML`.

В XML-файле, который будет определять меню, создадим пять пунктов меню — **Open**, **Save**, **Edit**, **Help** и **Exit** — с иконками, взятыми из приложения в разд. "Меню со знач-

ками" главы 11, и сохраним этот файл под именем `options.xml` в каталоге `res/menu/` проекта. Полный код файла `options.xml` представлен в листинге 15.10.

Листинг 15.10. Файл меню `options.xml`

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/open"
        android:title="Open"
        android:icon="@drawable/ic_menu_open"
        android:orderInCategory="1"/>
    <item
        android:id="@+id/save"
        android:title="Save"
        android:icon="@drawable/ic_menu_save"
        android:orderInCategory="2"/>
    <item
        android:id="@+id/edit"
        android:title="Edit"
        android:icon="@drawable/ic_menu_edit"
        android:orderInCategory="3"/>
    <item
        android:id="@+id/help"
        android:title="Help"
        android:icon="@drawable/ic_menu_help"
        android:orderInCategory="4"/>
    <item
        android:id="@+id/exit"
        android:title="Exit"
        android:icon="@drawable/ic_menu_exit"
        android:orderInCategory="5"/>
</menu>
```

Файл компоновки для приложения с единственным виджетом `TextView` приведен в листинге 15.11.

Листинг 15.11. Файл компоновки `main.xml`

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Press MENU button..."
```

```
android:gravity="center"
android:textStyle="bold"/>
```

```
</LinearLayout>
```

Загрузка меню из ресурса несколько отличается от создания меню в коде приложения, которое мы рассматривали в *главе 11*. В коде метода обратного вызова `onCreateOptionsMenu()` достаточно получить ссылку на ресурс XML-файла, определяющего меню:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options, menu);
    return true;
}
```

При загрузке меню из XML-файла идентификаторы меню определяют в файле и, следовательно, нет необходимости объявлять их в коде программы.

В классе `R` будут сгенерированы константы, к которым можно обращаться в коде, как и к идентификаторам остальных ресурсов. Эти идентификаторы можно использовать в методе обратного вызова `onOptionsItemSelected()`, например, таким образом:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
    switch (item.getItemId()) {
        case R.id.open:
            ...
            break;
        case R.id.save:
            ...
            break;
        default:
            return false;
    }
    ...
    return true;
}
```

Полный код класса `ResMenuXmlActivity` показан в листинге 15.12.

Листинг 15.12. Файл класса окна приложения `ResMenuXmlActivity.java`

```
package com.samples.res.resmenuxml;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.widget.Toast;

public class ResMenuXmlActivity extends Activity {

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

    // Загрузка меню из XML-файла
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.options, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message;
        switch (item.getItemId()) {
            case R.id.open:
                message = "Open item selected";
                break;
            case R.id.save:
                message = "Save item selected";
                break;
            case R.id.help:
                message = "Help item selected";
                break;
            case R.id.edit:
                message = "Edit item selected";
                break;
            case R.id.exit:
                message = "Exit item selected";
                break;
            default:
                return false;
        }
        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}
```

Скомпилируйте и запустите проект на выполнение. Внешний вид приложения с загрузкой меню из XML-файла не отличается от меню, созданного в коде программы (рис. 15.3).

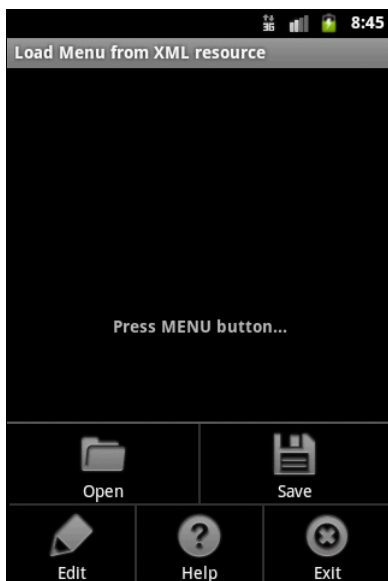


Рис. 15.3. Меню, загруженное из XML-файла

Загрузка XML-документов

Библиотеки Android имеют набор классов для чтения и записи XML-документов с произвольной структурой и содержанием. Чтобы упаковать статический XML-документ с вашим приложением, поместите его в каталог `res/xml/`, и тогда вы получите возможность обращаться в коде программы к этому документу.

Рассмотрим загрузку XML-документа произвольной структуры из ресурсов в код программы. Создадим в Eclipse приложение, способное читать список контактов, определенных в XML-файле. В окне **New Android Project** заполните поля:

- ☐ **Project name** — `Files_Xml`;
- ☐ **Application name** — `XMLResource Sample`;
- ☐ **Package name** — `com.samples.res.resxml`;
- ☐ **Create Activity** — `ResXmlActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch15_FilesXML`.

В каталоге `res/` создайте подкаталог `xml/`, в котором будет располагаться XML-файл контактов. В этом файле мы напишем список контактов, уже неоднократно применяемый в приложениях в этой книге, и сохраним его под именем `contacts.xml`. Полный код файла `contacts.xml` представлен в листинге 15.13.

Листинг 15.13. XML-документ `contacts.xml`

```
<contacts>
  <contact first_name="Jacob" last_name="Anderson" phone="412412411"/>
  <contact first_name="Emily" last_name="Duncan" phone="161863187"/>
</contacts>
```

```

<contact first_name="Michael" last_name="Fuller" phone="467657565"/>
<contact first_name="Emma" last_name="Greenman" phone="025665746"/>
<contact first_name="Joshua" last_name="Harrison" phone="475289568"/>
<contact first_name="Madison" last_name="Johnson" phone="891863187"/>
<contact first_name="Matthew" last_name="Cotman" phone="161863187"/>
<contact first_name="Olivia" last_name="Lawson" phone="943657875"/>
<contact first_name="Andrew" last_name="Chapman" phone="876867896"/>
<contact first_name="Daniel" last_name="Honeyman" phone="987654388"/>
<contact first_name="Isabella" last_name="Jackson" phone="312439787"/>
<contact first_name="William" last_name="Patterson" phone="687699693"/>
<contact first_name="Joseph" last_name="Godwin" phone="965467575"/>
<contact first_name="Samantha" last_name="Bush" phone="907865645"/>
<contact first_name="Christopher" last_name="Gateman" phone="896874556"/>
</contacts>

```

Файл компоновки `main.xml` для главного окна приложения приведен в листинге 15.14.

Листинг 15.14. Файл компоновки `main.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView
        android:id="@+id/text"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:gravity="left|center"
        android:textStyle="bold"/>
</LinearLayout>

```

Загрузить файл `contacts.xml`, созданный ранее, можно следующим образом:

```
XmlPullParser parser = getResources().getXml(R.xml.contacts);
```

Метод `getXml()` возвращает `XmlPullParser`, используя который можно прочитать загруженный XML-документ в цикле `while`:

```

while (parser.getEventType() != XmlPullParser.END_DOCUMENT) {
    if (parser.getEventType() == XmlPullParser.START_TAG
        && parser.getName().equals("contact")) {
        data.append(parser.getAttributeValue(0) + " "
            + parser.getAttributeValue(1) + ": "
            + parser.getAttributeValue(2) + "\n");
    }
    parser.next();
}

```

Полный код класса `ResXmlActivity` главного окна приложения показан в листинге 15.15.

Листинг 15.15. Файл класса окна приложения ResXmlActivity.java

```
package com.samples.res.resxml;

import org.xmlpull.v1.XmlPullParser;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class ResXmlActivity extends Activity {

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

        final TextView text = (TextView)findViewById(R.id.text);
        StringBuilder data = new StringBuilder();

        try {
            XmlPullParser parser = getResources().getXml(R.xml.contacts);

            while (parser.getEventType() != XmlPullParser.END_DOCUMENT) {
                if (parser.getEventType() == XmlPullParser.START_TAG
                    && parser.getName().equals("contact")) {
                    data.append(parser.getAttributeValue(0) + " "
                        + parser.getAttributeValue(1) + ": "
                        + parser.getAttributeValue(2) + "\n");
                }
                parser.next();
            }
            text.setText(data);
        }
        catch (Throwable t) {
            Toast.makeText(this,
                "Error loading XML document: " + t.toString(), 4000)
                .show();
        }
    }
}
```

Выполните компиляцию и запустите проект на выполнение. Приложение загрузит файл `contacts.xml` и отобразит его содержимое в виде списка. Внешний вид приложения, запущенного в эмуляторе мобильного устройства, показан на рис. 15.4.

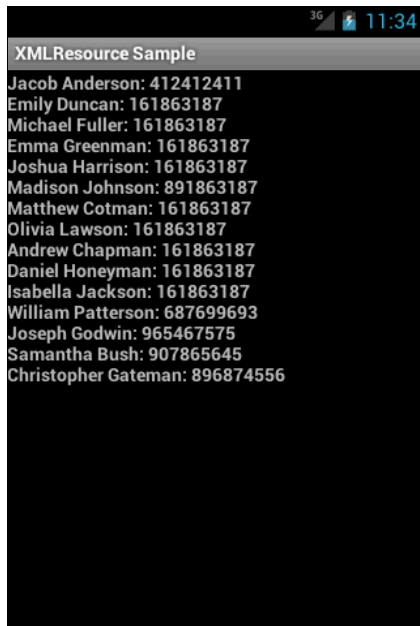


Рис. 15.4. Чтение XML-документа из ресурсов

Стили и темы

Проектируя приложение, вы можете использовать стили и темы для создания оригинального дизайна окон приложения и отдельных элементов пользовательского интерфейса.

Стили и темы — это такие же ресурсы, как и строки, изображения и т. д. Android предоставляет заданные по умолчанию стили и темы, которые вы можете использовать в приложениях. При необходимости вы можете определить свой собственный стиль и тему для создаваемого приложения.

Стили

Стиль — это один или несколько сгруппированных атрибутов форматирования, которые вы можете применить как модуль к отдельным элементам пользовательского интерфейса в XML-схеме компоновки для Activity. Например, вы можете определить стиль, который устанавливает определенный размер текста и цвет фона, а затем применить его к выбранным компонентам пользовательского интерфейса. Вот пример объявления стиля в XML-файле:

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

Как было показано ранее, вы можете использовать элементы `<item>`, чтобы установить определенные значения форматирования для стиля. Атрибут `name` в элементе может быть стандартной строкой, шестнадцатеричным значением цвета или ссылкой на любой другой тип ресурса.

Обратите внимание на атрибут `parent` в элементе `<style>`. Этот атрибут позволяет определять ресурс, от которого текущий стиль наследует значения свойств. Стиль может наследоваться от любого типа ресурса, который содержит требуемый стиль. Вообще, ваши собственные стили должны всегда наследоваться, прямо или косвенно, от стандартных стилей Android. В этом случае необходимо только определить значения, которые требуется изменить в наследуемом стиле.

Для ссылки на стиль используется атрибут `style`. Например, так можно присоединить стиль для элемента `TextView`:

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

Теперь этот виджет `TextView` будет иметь стиль `CustomText`, который был объявлен ранее в примере с XML-кодом.

Темы

Тема — группа из одного или нескольких атрибутов форматирования, которые вы можете применить как модуль ко всем действиям в приложении или только единственном `Activity`. Например, вы могли определить тему, которая выбирает цвета для рамки окна, группового переднего плана и фона и устанавливает текстовые размеры и цвета для меню, затем применить эту тему в разных приложениях.

Темы похожи на определения стилей. Точно так же, как стили, темы объявляются в XML-файле элементами `<style>` и ссылаются на них тем же самым способом. Различие состоит в том, что тема добавляется ко всему приложению или к отдельному `Activity` через элементы `<application>` и `<activity>` в файле манифеста приложения, т. к. темы не могут быть применены к отдельным представлениям.

Вот объявление примера темы:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="theme_custom">
        <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="panelBackgroundColor">#FFFFFFF</item>
        <item name="panelTextColor">?panelForegroundColor</item>
        <item name="panelTextSize">14</item>
```

```

    <item name="menuItemTextColor"?panelTextColor</item>
    <item name="menuItemTextSize"?panelTextSize</item>
</style>
</resources>

```

Чтобы установить эту тему для всех Activity приложения, откройте файл `AndroidManifest.xml` и отредактируйте элемент `<application>`, добавив атрибут `android:theme` с названием темы:

```
android:theme="@style/theme_custom"
```

Если вы хотите определить тему, которая будет загружаться только для одного Activity, добавьте атрибут с темой в элемент `<activity>` данного Activity.

Наряду с другими встроенными ресурсами, в Android есть несколько стандартных тем, которые можно загрузить в приложение. Например, вы можете использовать тему `Theme.Dialog`, чтобы заставить Activity отобразиться на экране как диалоговое окно. В файле манифеста объявить ссылку на стандартную тему Android можно следующим образом:

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

Определение собственных стилей и тем

Чтобы создать собственный стиль или тему в приложении, необходимо предпринять следующие действия:

1. Создать файл с названием `styles.xml` в каталоге приложения `res/values/`. В файл добавить корневой элемент `<resources>`.
2. Для каждого стиля или темы добавить элемент `<style>` с уникальным именем, например `name="theme_custom"`, и при необходимости родительский атрибут. Имя стиля используется для ссылки на этот стиль позже, и родительский атрибут указывает то, что данный стиль наследуется от другого стиля.
3. В элементе `<style>` создать набор из одного или нескольких элементов `<item>`. Каждый элемент `<item>` задает какие-либо свойства стиля.

Давайте сейчас разработаем приложение, в котором определим собственные стиль и тему. В Eclipse создайте новый проект и в окне **New Android Project** заполните поля:

- ☐ **Project name** — `StylesAndThemes`;
- ☐ **Application name** — `Styles and Themes`;
- ☐ **Package name** — `com.samples.res.stylesandthemes`;
- ☐ **Create Activity** — `StylesAndThemesActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch15_StylesAndThemes`.

В каталоге `res/values/` создадим XML-файл, в котором определим стили для текстового поля — `SpecialText` и для приложения — `Stars`, и сохраним его под именем `styles.xml`. Код файла приведен в листинге 15.16.

Листинг 15.16. Файл styles.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="SpecialText">
        <item name="android:textSize">28sp</item>
        <item name="android:textColor">#FF0000</item>
        <item name="android:background">#000000</item>
    </style>

    <style name="Stars">
        <item name="android:windowBackground">@drawable/stars</item>
        <item name="android:windowNoTitle">true</item>
    </style>
</resources>
```

В файле компоновки main.xml определим единственный элемент `TextView` и для этого элемента зададим созданный в файле styles.xml стиль, используя для этого атрибут `style`:

```
style="@style/SpecialText"
```

Полный код файла компоновки main.xml представлен в листинге 15.17.

Листинг 15.17. Файл компоновки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        style="@style/SpecialText"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:gravity="center"
        android:layout_width="wrap_content"
        android:padding="5px"/>

</LinearLayout>
```

В файле манифеста приложения теперь необходимо объявить ссылки на тему для приложения, используя атрибут `android:theme`:

```
android:theme="@style/Stars"
```

Код файла манифеста приложения приведен в листинге 15.18.

Листинг 15.18. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.res.stylesandthemes"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name"
        android:theme="@style/Stars">
        <activity
            android:name=".StylesAndThemesActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Внешний вид приложения с собственным стилем для текстового поля и темой для всего приложения показан на рис. 15.5.

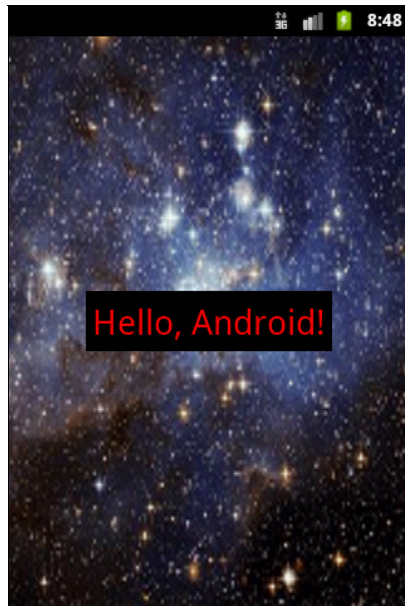


Рис. 15.5. Загрузка стилей и темы из ресурсов

Активы

Активы — это файлы произвольного типа и содержания, располагаемые в каталоге `assets/`. Этот каталог находится на одном уровне с каталогом `res/` в дереве проекта. Особенностью этого каталога является то, что файлы в `assets/` не генерируют идентифика-

торы ресурса в классе `R`. Поэтому при использовании активов в программном коде необходимо явно определять путь к файлу для доступа к нему. Путь к файлу — это относительный путь, начинающийся с `assets/`. Для обращения к этим файлам используется класс `AssetManager`. Этот каталог, в отличие от подкаталога `res/`, допускает произвольную глубину подкаталогов и произвольные имена файлов и подкаталогов.

Приведем практический пример: необходимо разработать приложение, в котором будут использоваться шрифты стороннего производителя, не входящие в стандартную библиотеку системных шрифтов Android. Самый простой способ сделать это состоит в том, чтобы упаковать нужные шрифты вместе с приложением. Для этого необходимо создать каталог `assets/` в корне каталога проекта и поместить в этот каталог свои файлы шрифтов.

В качестве примера приложения для работы со шрифтами, загружаемыми как активы в приложение, создайте новый проект и в окне **Create New Project** введите следующие значения:

- ☐ **Project name** — `ResFontsApp`;
- ☐ **Application name** — `Load fonts from assets sample`;
- ☐ **Package name** — `com.samples.res.fonts`;
- ☐ **Create Activity** — `FontsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch15_AssetsFonts`.

Файл компоновки `main.xml` состоит из контейнера `LinearLayout` и четырех дочерних виджетов `TextView`, которые будут отображать текст с использованием загружаемых шрифтов.

Код XML-файла компоновки приложения `main.xml` представлен в листинге 15.19.

Листинг 15.19. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:gravity="center">

    <TextView
        android:text="Libertine Italic"
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:padding="10sp"/>

    <TextView
        android:text="Abaddon font"
```

```

        android:id="@+id/text2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="36sp"
        android:padding="10sp"/>
<TextView
    android:text="a Papa font"
    android:id="@+id/text3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="36sp"
    android:padding="10sp"/>
<TextView
    android:text="A Yummy Apology"
    android:id="@+id/text4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="36sp"
    android:padding="10sp"/>

</LinearLayout>

```

ПРИМЕЧАНИЕ

Для примеров были использованы свободно распространяемые шрифты: Linux-шрифт Libertine и несколько свободных шрифтов, доступных на сайте <http://www.urbanfonts.com/free-fonts.htm>.

В классе, реализующем Activity, мы загружаем из ресурсов объект `TextView`, а затем создаем объект класса `Typeface`, используя вызов статического метода `Typeface.createFromAsset()`. Объект класса `Typeface` определяет характеристики шрифта — стили, цвет, семейство и т. д.

Метод `createFromAsset()` принимает два параметра:

- объект `AssetManager`, который можно получить вызовом метода `getAssets()`;
- путь к файлу актива.

Например, загрузить шрифт для текстового поля `TextView` можно следующим способом:

```

TextView text = (TextView)findViewById(R.id.text_i);
Typeface face = Typeface.createFromAsset(
    getAssets(), "fonts/libertine_it.ttf");
text.setTypeface(face);

```

Полный код класса `FontsActivity` приводится в листинге 15.20.

Листинг 15.20. Файл класса окна приложения `FontsActivity.java`

```

package com.samples.res.fonts;

import android.app.Activity;
import android.graphics.Typeface;

```

```
import android.os.Bundle;
import android.widget.TextView;

public class FontsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Загружаем 4 типа шрифтов для текстовых полей
        final TextView text1 = (TextView)findViewById(R.id.text1);
        text1.setTypeface(Typeface.createFromAsset(
            getAssets(), "fonts/libertine_it.ttf"));

        final TextView text2 = (TextView)findViewById(R.id.text2);
        text2.setTypeface(Typeface.createFromAsset(
            getAssets(), "fonts/abaddon.ttf"));

        final TextView text3 = (TextView)findViewById(R.id.text3);
        text3.setTypeface(Typeface.createFromAsset(
            getAssets(), "fonts/apapa.ttf"));

        final TextView text4 = (TextView)findViewById(R.id.text4);
        text4.setTypeface(Typeface.createFromAsset(
            getAssets(), "fonts/ayummyapology.ttf"));
    }
}
```

Запустите проект на выполнение. Приложение будет загружать файлы со шрифтами из каталога `assets/` и использовать их в текстовых полях. Внешний вид приложения, запущенного в эмуляторе мобильного устройства, показан на рис. 15.6.



Рис. 15.6. Загрузка шрифтов, сохраненных в каталоге `assets/`

Резюме

В этой главе мы изучили возможности, предоставляемые платформой Android для доступа приложения к ресурсам, определяемым во внешних файлах, загрузку этих файлов и их использование в Android-приложениях.

Далее мы изучим файловую систему Android и рассмотрим работу с файлами — файловый ввод-вывод, создание, чтение и запись файлов в приложениях Android.



ГЛАВА 16

Файловая система и карта памяти

В этой главе мы изучим подключение и работу с внешней картой памяти, а также структуру файловой системы Android. Кроме того, мы рассмотрим организацию файловой системы на карте памяти и в мобильном устройстве, создание каталогов, сохранение и чтение файлов с карты памяти.

Подключение карты памяти в эмуляторе

Карты памяти используются в мобильных устройствах Android для сохранения пользовательских файлов, обычно содержащих медиа, таких как фотографии, музыка и видео. Эмулятор Android также позволяет имитировать подключенную карту памяти, как в реальном мобильном устройстве, используя место на жестком диске компьютера.

Для работы с примерами этой главы нам понадобится экземпляр Android Virtual Device с картой памяти. Симулировать подключенную карту памяти к эмулятору Android очень просто. Для этого запустите утилиту **Android Virtual Device Manager**, создайте новый или измените существующий экземпляр Android Virtual Device. В окне **Edit Android Virtual Device (AVD)** на панели **SD Card** введите размер карты памяти в мегабайтах или килобайтах, как показано на рис. 16.1.

Файловая система Android

Поскольку система Android построена на ядре Linux, она имеет такую же файловую систему, как и все настольные Linux-системы, за исключением небольших отличий, специфических для мобильных систем.

В настоящее время большинство устройств Android использует файловую систему YAFFS — Yet Another Flash File System, которая была специально оптимизирована для использования на мобильных устройствах и картах памяти с учетом того, что данные на таких устройствах сохраняются во флэш-памяти.

Однако у YAFFS есть одна большая проблема: одновременно только один процесс может записывать данные в файловую систему. Вместо того чтобы использовать блокировку на уровне файла (File-Level lock), у YAFFS создана блокировка на уровне раздела (Partition-Level lock). Но, если учитывать быстро растущую производительность устройств на основе Android, использование этой файловой системы заметно снижает производительность. Поэтому в настоящее время разработчики устройств на Android начи-

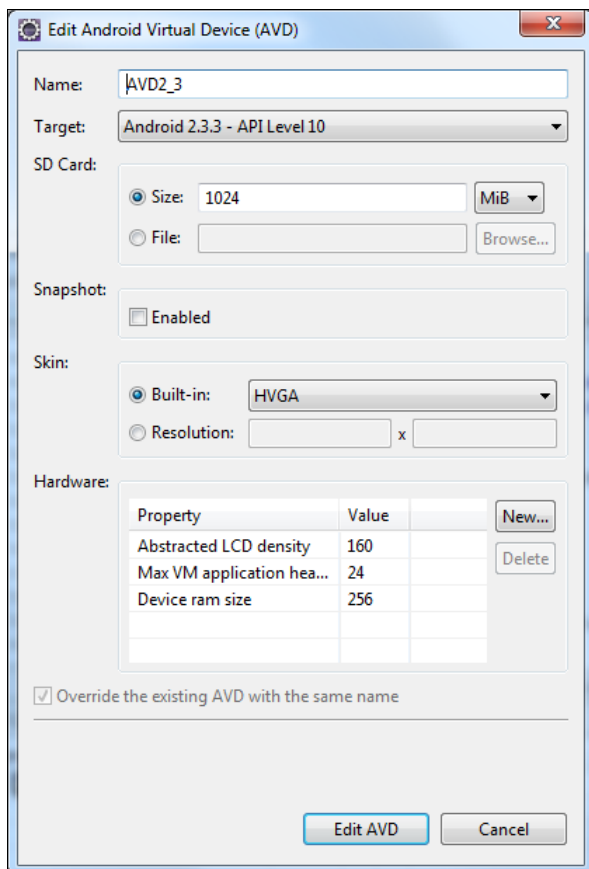


Рис. 16.1. Создание карты памяти в эмуляторе

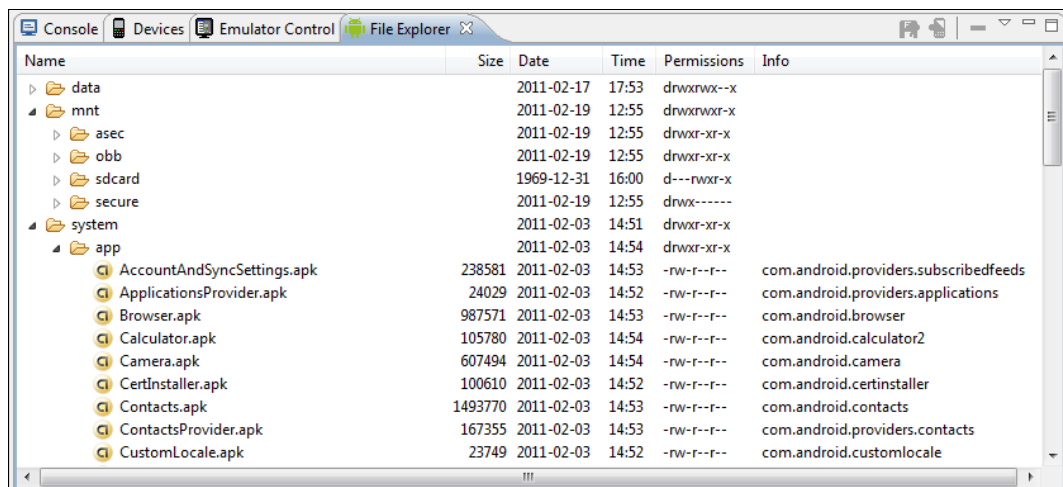


Рис. 16.2. Файловая система устройства Android

нают постепенно переходить на файловую систему Ext4, которая предназначена для использования в настольных компьютерах.

С помощью инструментов Android SDK вы можете посмотреть организацию файловой системы вашего мобильного устройства или эмулятора. Для просмотра файловой системы Android вы можете воспользоваться утилитой DDMS или открыть в IDE Eclipse представление **File Explorer**. Это представление отображает всю структуру файловой системы в виде дерева, показывает даты создания файлов и директорий, размеры файлов и разрешения. Типичное дерево файловой системы Android в **File Explorer** показано на рис. 16.2.

Стандартные директории Android

В библиотеке Android в пакете `android.os` есть класс `Environment`. С его помощью можно работать с директориями файловой системы Android.

Все методы для работы с директориями из класса `Environment` возвращают тип `File`. Класс `File` определен в пакете `java.io.File`. В языке Java этот тип определяет не только файл, но и директорию. Если требуется узнать, является ли возвращаемый объект файлом или директорией, используются методы `isFile()` или `isDirectory()` класса `File`.

Класс `Environment` предоставляет набор методов для чтения стандартных директорий файловой системы Android:

- ❑ `getDataDirectory()` — возвращает директорию для хранения данных;
- ❑ `getDownloadCacheDirectory()` — возвращает директорию для хранения загружаемых внешних файлов и кэша;
- ❑ `getRootDirectory()` — возвращает корневую директорию файловой системы Android;
- ❑ `getExternalStorageDirectory()` — возвращает корневую директорию внешней карты памяти мобильного устройства.

Можно сохранять все файлы в корневой директории карты памяти, однако хорошим тоном является создание подкаталогов для хранения файлов определенного типа — музыки, видео, фотографий и т. д. Эти директории лучше называть стандартными именами, определенными в качестве строковых констант в классе `Environment`, чтобы с ними могли работать приложения, созданные разными разработчиками программного обеспечения.

Метод `getExternalStoragePublicDirectory()` определяет путь к стандартным директориям на карте памяти для размещения файлов конкретного типа (медиа, графика и др.). В качестве параметра этому методу передается одна из строковых констант из класса `Environment`, определяющих конкретную стандартную директорию:

- ❑ `DIRECTORY_ALARMS` — директория для файлов звуковых оповещений;
- ❑ `DIRECTORY_DCIM` — директория для видео и фотографий, записанных со встроенной камеры;
- ❑ `DIRECTORY_DOWNLOADS` — директория для файлов, загружаемых пользователем телефона;
- ❑ `DIRECTORY_MOVIES` — директория для размещения видеофайлов;
- ❑ `DIRECTORY_MUSIC` — директория для размещения музыки;

- ❑ `DIRECTORY_NOTIFICATIONS` — директория для звуковых файлов, которые используются в уведомлениях (например, входящий SMS, низкий уровень заряда батареи телефона и др.);
- ❑ `DIRECTORY_PICTURES` — директория для размещения графических файлов;
- ❑ `DIRECTORY_PODCASTS` — директория для размещения подкастов, т. е. регулярно обновляемого с помощью RSS списка файлов;
- ❑ `DIRECTORY_RINGTONES` — директория для звуковых файлов, используемых при получении входящего звонка.

Для изучения файловой системы и директорий создадим приложение, которое выведет нам информацию о файловой системе мобильного устройства. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — Android Environment;
- ❑ **Application name** — Android Environment;
- ❑ **Package name** — `com.samples.os.environment`;
- ❑ **Create Activity** — `EnvironmentActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch16_AndroidEnvironment`.

Код файла разметки очень простой и содержит элемент `TextView` для вывода результатов работы программы.

В классе `EnvironmentActivity` главного окна приложения мы используем вызовы методов класса `Environment` для отображения директорий и путей к ним. Код класса `EnvironmentActivity` представлен в листинге 16.1.

Листинг 16.1. Файл класса главного окна приложения `EnvironmentActivity.java`

```
package com.samples.os.environment;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.widget.TextView;

public class EnvironmentActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TextView text = (TextView)findViewById(R.id.text);

        text.append(
            "Root:\t" + Environment.getRootDirectory() +
            "\nDownload Cache Dir:\t" +
                Environment.getDownloadCacheDirectory() +
```



```

"\nExternal Storage State:\t" +
    Environment.getExternalStorageState() +
"\nData Directory:\t" + Environment.getDataDirectory() +
"\nisExternal Storage Removable:\t" +
    Environment.isExternalStorageRemovable() +
"\nExternal Storage Dir:\t" +
    Environment.getExternalStorageDirectory() +
"\n\nExternal Storage Public Directory:\t" +
"\n\tAlarms:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_ALARMS) +
"\n\tDCIM:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_DCIM) +
"\n\tDownloads:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_DOWNLOADS) +
"\n\tMovies:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_MOVIES) +
"\n\tMusic:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_MUSIC) +
"\n\tNotification:\t" +
    Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_NOTIFICATIONS) +
"\n\tPictures:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_PICTURES) +
"\n\tPodcasts:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_PODCASTS) +
"\n\tRingtones:\t" + Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_RINGTONES));
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. Приложение выведет список всех директорий в файловой системе устройства и пути к ним. Внешний вид приложения и отображаемая им информация представлены на рис. 16.3.

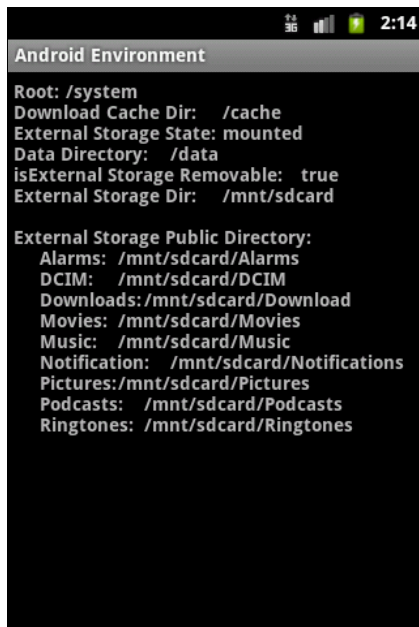


Рис. 16.3. Вывод директорий файловой системы Android

Проверка состояния карты памяти

Перед тем как приложение начнет работу с файловой системой карты памяти, желательно проверить состояние карты. Для этого в классе `Environment` есть метод `getExternalStorageState()`, который возвращает текущее состояние карты памяти. Возвращаемое значение имеет тип `String` и может принимать одно из значений строковых констант, определенных в классе `Environment`:

- ❑ `MEDIA_MOUNTED` — карта имеет точку монтирования к файловой системе мобильного устройства с доступом для чтения и записи;
- ❑ `MEDIA_UNMOUNTED` — карта памяти вставлена в устройство, но не подмонтирована к файловой системе;
- ❑ `MEDIA_BAD_REMOVAL` — карта памяти была удалена без размонтирования;
- ❑ `MEDIA_CHECKING` — карта памяти вставлена в устройство и проверяется на ошибки в файловой системе;
- ❑ `MEDIA_MOUNTED_READ_ONLY` — карта имеет точку монтирования к файловой системе мобильного устройства с доступом только для чтения;
- ❑ `MEDIA_NOFS` — карта памяти вставлена, но не отформатирована или имеет файловую систему, которая не поддерживается мобильным устройством;
- ❑ `MEDIA_REMOVED` — карта памяти отсутствует;
- ❑ `MEDIA_SHARED` — карта памяти вставлена в мобильное устройство и не подмонтирована, но доступна через USB для внешних устройств (например, PC);
- ❑ `MEDIA_UNMOUNTABLE` — карта памяти вставлена в устройство, но не может быть подмонтирована к файловой системе.

Если вы будете разрабатывать серийные приложения, которые в процессе своей работы обращаются к карте памяти, обязательно выполняйте проверку состояния карты, чтобы у пользователя не возникало проблем при работе с вашим приложением.

Чтение и запись файлов

В операционной системе Android можно сохранять файлы непосредственно на мобильном устройстве или на внешнем носителе данных, например на карте памяти. По умолчанию другие приложения не могут обращаться к этим файлам.

Операции ввода-вывода в Android аналогичны операциям в стандартных Java-программах. Android реализует потоки с помощью иерархии классов, определенных в пакете `java.io` (рис. 16.4). Кроме этих классов в пакете `java.io` определено множество специализированных классов для операций ввода-вывода.

Чтобы прочитать данные из файла, необходимо вызвать метод `Context.openFileInput()` и передать в качестве параметра имя файла. Метод возвращает стандартный Java-объект `FileInputStream`. Например, код для чтения данных из текстового файла может выглядеть так:

```
InputStream inStream = openFileInput(file.txt);  
InputStreamReader sr = new InputStreamReader(inStream);
```

```
// Создаем буфер для чтения файла
BufferedReader reader = new BufferedReader(sr);
String str;
StringBuffer buffer = new StringBuffer();
// Читаем данные в буфер
while ((str = reader.readLine()) != null) {
    buffer.append(str + "\n");
}

inStream.close();
```

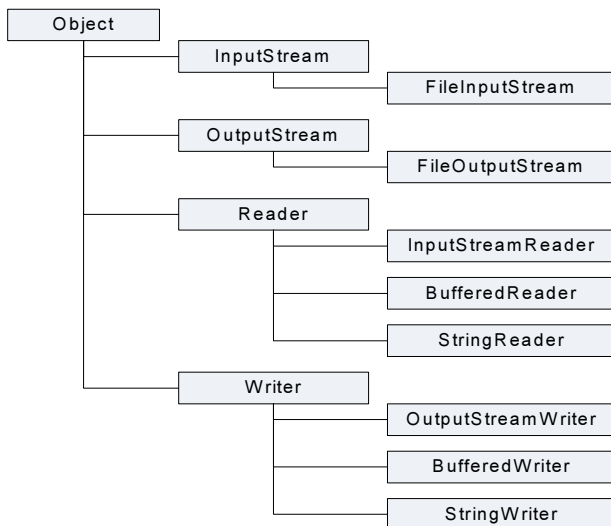


Рис. 16.4. Классы для файлового ввода-вывода в Android

Чтобы записывать в файл, необходимо вызвать метод `Context.openFileOutput()`, передав ему имя файла как параметр. Этот метод возвращает объект `FileOutputStream`. Вызов этих методов для данного файла из другого приложения не будет работать, обратиться вы можете только к своим файлам. Пример записи строки данных в файл `file.txt` может быть следующим:

```
String data;
...
OutputStream outStream = openFileOutput(file.txt, 0);
OutputStreamWriter sw = new OutputStreamWriter(outStream);

sw.write(data);
sw.close();
```

Если имеется статический файл, который надо упаковать с вашим приложением во время компиляции проекта, можно сохранить его в каталоге проекта в папке `res/raw/`, а затем открыть его методом `Resources.openRawResource()`. Этот метод возвращает объект `InputStream`, который можно использовать для чтения файла. После окончания работы с потоком не забудьте его закрыть, вызвав метод `close()`.

ОБРАТИТЕ ВНИМАНИЕ

Методы `openFileInput()` и `openFileOutput()` не принимают полного пути к файлу (например, `path/files/file.txt`), только простые имена файлов.

В качестве примера приложения, записывающего и читающего данные из файлов, создадим простейший текстовый редактор с виджетом `EditText` и меню для открытия файла и сохранения его после редактирования. Создайте в Eclipse новый проект, заполнив поля в окне **New Android Project**:

- ☐ **Project name** — `ContactEditor`;
- ☐ **Application name** — `ReadWriteFile Sample`;
- ☐ **Package name** — `com.samples.res.filesrw`;
- ☐ **Create Activity** — `EditorActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch16_Files_ReadWrite`.

Код XML-схемы компоновки `main.xml`, в которой находится единственный элемент `EditText`, приводится в листинге 16.2.

Листинг 16.2. Файл компоновки `main.xml`

```
<?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">

    <EditText
        android:id="@+id/edit"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:singleLine="false"/>

</LinearLayout>
```

В классе `EditorActivity`, определяющем главное окно приложения, создадим меню из трех пунктов — **Open**, **Save** и **Exit**. Во внутренних методах `openFile()` и `saveFile()` реализуем операции по открытию и сохранению файла, приведенные ранее. Код класса `EditorActivity` представлен в листинге 16.3.

Листинг 16.3. Файл класса окна приложения `EditorActivity.java`

```
package com.samples.app.filesrw;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Toast;

public class EditorActivity extends Activity {

    private static final int IDM_OPEN = 101;
    private static final int IDM_SAVE = 102;
    private static final int IDM_EXIT = 103;

    private final static String FILENAME = "file.txt";
    private EditText edit;

    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.main);

        edit = (EditText)findViewById(R.id.edit);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setIcon(R.drawable.ic_menu_open)
            .setAlphabeticShortcut('o');
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setIcon(R.drawable.ic_menu_save)
            .setAlphabeticShortcut('s');
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setIcon(R.drawable.ic_menu_exit)
            .setAlphabeticShortcut('x');

        return (super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case IDM_OPEN:
                openFile(FILENAME);
                break;
            case IDM_SAVE:
                saveFile(FILENAME);
                break;
        }
    }
}
```

```

        case IDM_EXIT:
            finish();
            break;
        default:
            return false;
    }
    return true;
}

private void openFile(String fileName) {
    try {
        InputStream inStream = openFileInput(FILENAME);

        if (inStream != null) {
            InputStreamReader tmp =
                new InputStreamReader(inStream);
            BufferedReader reader = new BufferedReader(tmp);
            String str;
            StringBuffer buffer = new StringBuffer();

            while ((str = reader.readLine()) != null) {
                buffer.append(str + "\n");
            }

            inStream.close();
            edit.setText(buffer.toString());
        }
    }
    catch (Throwable t) {
        Toast.makeText(getApplicationContext(),
            "Exception: " + t.toString(), Toast.LENGTH_LONG)
            .show();
    }
}

private void saveFile(String FileName) {
    try {
        OutputStreamWriter outStream =
            new OutputStreamWriter(openFileOutput(FILENAME, 0));

        outStream.write(edit.getText().toString());
        outStream.close();
    }
    catch (Throwable t) {
        Toast.makeText(getApplicationContext(),
            "Exception: " + t.toString(), Toast.LENGTH_LONG)
            .show();
    }
}
}

```

Внешний вид приложения, позволяющего читать и записывать текст в файл, приведен на рис. 16.5.

В результате мы получили простую записную книжку, хранящую записи сколь угодно долгое время в файле. В качестве упражнения попробуйте усовершенствовать приложение, добавив возможность создания новых файлов, их открытия и сохранения, а также в случае необходимости удаления файлов.

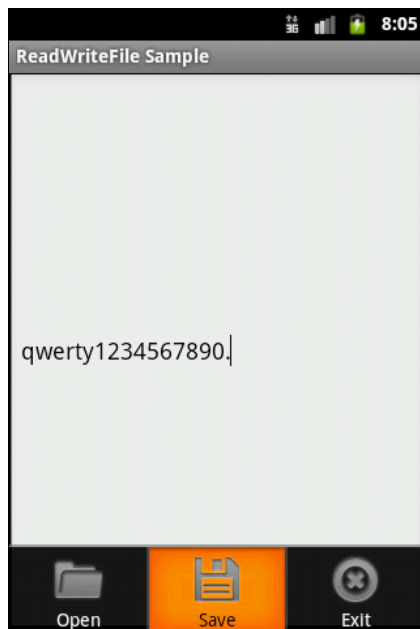


Рис. 16.5. Текстовый редактор с возможностью сохранения содержимого в файле

Сохранение и чтение файлов с SD-карты

С целью изучения функциональности, предоставляемой системой Android для работы с файлами, создадим новое приложение. Это будет текстовый редактор, в котором пользователь может написать текст — создать документ, а затем сохранить его на карту памяти в отдельную директорию, предусмотренную специально для этого приложения. Если эта директория отсутствует на карте памяти, она будет создана. Приложение также сможет выводить список всех файлов, находящихся в этой директории, и открывать выбранный файл в своем текстовом редакторе.

Создайте в IDE Eclipse новый проект Android и заполните поля мастера **New Android Project**:

- ☐ **Project name** — `SDCard_ReadWriteFiles`;
- ☐ **Application name** — `Wordpad`;
- ☐ **Package name** — `com.samples.sdcard.readwritefiles`;
- ☐ **Create Activity** — `EditorActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch16_SDCard_ReadWriteFiles`.

В файл манифеста приложения `AndroidManifest.xml` добавьте разрешение `android.permission.WRITE_EXTERNAL_STORAGE` для возможности записи файлов на карту памяти, как показано в листинге 16.4.

Листинг 16.4. Файл манифеста приложения AndroidManifest.xml

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.filesrw">
    <application>
        <activity android:name=".EditorActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
</manifest>

```

Поскольку в нашем приложении будут меню и диалоги, содержащие достаточно большое количество надписей, все строковые ресурсы для удобства вынесем в файл `strings.xml`, код которого показан в листинге 16.5.

Листинг 16.5. Файл ресурсов strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Wordpad</string>
    <string name="btn_ok">OK</string>
    <string name="btn_cancel">Cancel</string>
    <string name="title_open">Open document</string>
    <string name="title_save">Save document</string>
    <string name="menu_new">New</string>
    <string name="menu_open">Open</string>
    <string name="menu_save">Save</string>
    <string name="menu_exit">Exit</string>
</resources>

```

В файле компоновки главного окна приложения `main.xml` у нас будет только один элемент `EditText` с идентификатором `edit`, в котором пользователь будет вводить или редактировать текст. Код файла `main.xml` представлен в листинге 16.6.

Листинг 16.6. Файл компоновки окна приложения main.xml

```

<?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">

```



```
<EditText
    android:id="@+id/edit"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:singleLine="false"/>
```

```
</LinearLayout>
```

Диалоговое окно для сохранения файла сделаем нестандартным — в нем будет расположен виджет `EditText` для ввода имени сохраняемого файла. Поэтому нам потребуется отдельный файл компоновки для этого диалога, который мы создадим в каталоге `res/layout/` и назовем `savedialog.xml` (листинг 16.7).

Листинг 16.7. Файл компоновки окна диалога `savedialog.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/layout_save"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <EditText
        android:id="@+id/edit_filename"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:hint="Enter file name"/>

</LinearLayout>
```

В коде класса `EditorActivity` для главного окна приложения поместим текстовый редактор `EditText` и меню из четырех опций: **New**, **Open**, **Save**, **Exit**.

В классе `EditorActivity` будет реализована вся функциональность для создания и сохранения текстовых файлов, чтения директорий и файлов из файловой системы карты памяти, т. е. все, что мы рассматривали в этой главе. Код класса `EditorActivity` представлен в листинге 16.8.

Листинг 16.8. Файл класса окна приложения `EditorActivity.java`

```
package com.samples.sdcard.readwritefiles;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
```

```
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.os.Environment;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class EditorActivity extends Activity {
    // Идентификаторы пунктов меню
    private static final int IDM_NEW = 200;
    private static final int IDM_OPEN = 201;
    private static final int IDM_SAVE = 202;
    private static final int IDM_EXIT = 203;

    // Константа с именем директории на карте памяти
    private static final String DIRECTORY_DOCUMENTS = "/docs";

    // Расширение для файлов
    private static final String FILE_EXT = ".txt";

    private EditText editText;
    private String curFileName = "";
    private String dir;
    private int pos = 0;

    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.main);

        editText = (EditText)findViewById(R.id.edit);

        // Читаем директорию на карте памяти,
        // которую использует наше приложение
        dir = Environment.getExternalStorageDirectory().toString() +
            DIRECTORY_DOCUMENTS;
        File folder = new File(dir);

        // Если директория не существует, создаем ее
        if (!folder.exists()) {
            folder.mkdir();
        }
    }
}
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(Menu.NONE, IDM_NEW, Menu.NONE, R.string.menu_new)
        .setIcon(R.drawable.menu_new)
        .setAlphabeticShortcut('n');
    menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, R.string.menu_open)
        .setIcon(R.drawable.menu_open)
        .setAlphabeticShortcut('o');
    menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, R.string.menu_save)
        .setIcon(R.drawable.menu_save)
        .setAlphabeticShortcut('s');
    menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, R.string.menu_exit)
        .setIcon(R.drawable.menu_exit)
        .setAlphabeticShortcut('x');

    return (super.onCreateOptionsMenu(menu));
}
```

// Выбор элемента меню

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case IDM_NEW:
            curFileName = "";
            editText.setText("");
            this.setTitle(R.string.app_name);
            break;
        case IDM_OPEN:
            callOpenDialog();
            break;
        case IDM_SAVE:
            callSaveDialog();
            break;
        case IDM_EXIT:
            finish();
            break;
        default:
            return false;
    }
    return true;
}
```

// Создание диалога сохранения файла

```
private void callSaveDialog() {
    LayoutInflater inflater = this.getLayoutInflater();
    View root = inflater.inflate(R.layout.savedialog, null);

    final EditText editFileName =
        (EditText) root.findViewById(R.id.edit_filename);
    editFileName.setText(curFileName);
}
```

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setView(root);
builder.setTitle(R.string.title_save);

// Закрытие диалога с сохранением файла
builder.setPositiveButton(
    R.string.btn_ok, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int item) {
            saveFile(editFileName.getText().toString());
        }
    });

// Закрытие диалога без сохранения файла
builder.setNegativeButton(
    R.string.btn_cancel, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {}
    });

builder.show();
}

// Создание диалога открытия файла
private void callOpenDialog() {
    try {
        final String[] files = findFiles(dir);

        // Диалог создается только при наличии файлов в директории
        if (files.length > 0) {
            pos = 0;
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle(R.string.title_open);

            // Отображаем в диалоге список файлов
            builder.setSingleChoiceItems(
                files, 0, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int item) {
                        pos = item;
                    }
                });

            // Обработка закрытия диалога с выбранным файлом (OK)
            builder.setPositiveButton(R.string.btn_ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        curFileName = files[pos];
                        openFile(curFileName);
                    }
                });
        }
    }
}
```

```
// Обработка закрытия диалога (Cancel)
builder.setNegativeButton(R.string.btn_cancel,
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });

builder.setCancelable(false);
builder.show();
}
}

catch (Exception e) {
    Toast.makeText(this, e.toString(), 3000).show();
}

}

// Сохранение файла на карте памяти
private void saveFile(String fileName) {
    try {
        if (!fileName.endsWith(FILE_EXT)) {
            fileName += FILE_EXT;
        }

        File file = new File(dir, fileName);
        FileOutputStream fos = new FileOutputStream(file);

        fos.write(editText.getText().toString().getBytes());
        fos.close();
    }
    catch (Exception e) {
        Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
    }
}

// Открытие файла с заданным именем
// и загрузка его в текстовый редактор
private void openFile(String fileName) {
    try {
        File file = new File(dir, fileName);
        FileInputStream inStream = new FileInputStream(file);

        if (inStream != null) {
            InputStreamReader tmp =
                new InputStreamReader(inStream);
            BufferedReader reader = new BufferedReader(tmp);
            String str;
            StringBuffer buffer = new StringBuffer();
```

```

        while ((str = reader.readLine()) != null) {
            buffer.append(str + "\n");
        }

        inStream.close();
        editText.setText(buffer.toString());

        curFileName = fileName;

        // Отображаем название файла в заголовке Activity
        setTitle(getResources().getString(R.string.app_name) +
            ": " + curFileName);
    }
}
catch (Exception e) {
    Toast.makeText(this, e.toString(), 3000).show();
}
}

// Поиск файлов в выбранной директории
private String[] findFiles(String dirPath) {
    ArrayList<String> items = new ArrayList<String>();
    try {

        File f = new File(dirPath);
        File[] files = f.listFiles();

        for (int i = 0; i < files.length; i++) {
            File file = files[i];

            // Если это файл, а не каталог,
            // добавляем его в список файлов
            if(!file.isDirectory()) {
                items.add(file.getName());
            }
        }
    }
    catch (Exception e) {
        Toast.makeText(this, e.toString(), 3000).show();
    }
    return items.toArray(new String[items.size()]);
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. В результате мы создали простой текстовый редактор, способный сохранять файлы, создаваемые пользователем, на карту памяти в специально отведенную для них директорию. Внешний вид приложения представлен на рис. 16.6.

При открытии приложения отображается диалоговое окно открытия файла **Open document**, который выводит список текстовых файлов, находящихся в этой директо-

рии. С помощью этого диалога пользователь может выбрать и открыть в редакторе текстовый файл из списка (рис. 16.7).

При желании вы можете усовершенствовать это приложение, добавив в него возможность выбора других директорий, не только на карте памяти, поиск нужных файлов и много других полезных функций для работы с файловой системой Android.

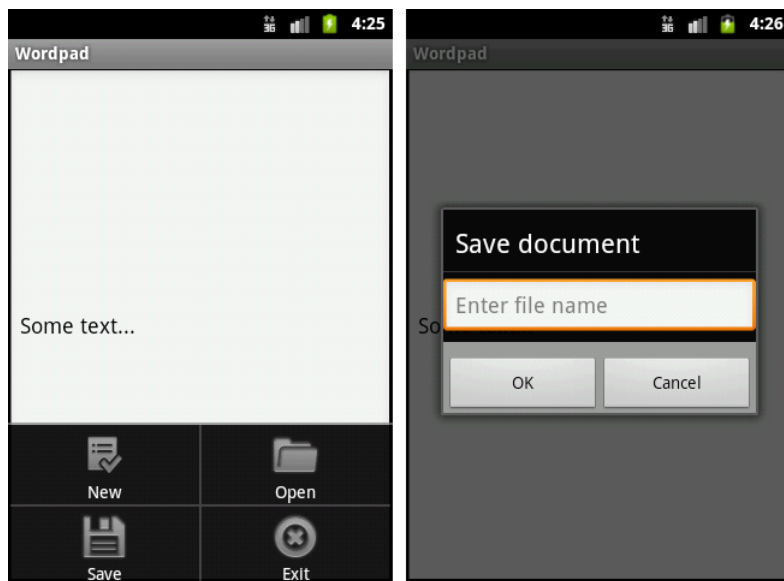


Рис. 16.6. Создание и сохранение документа

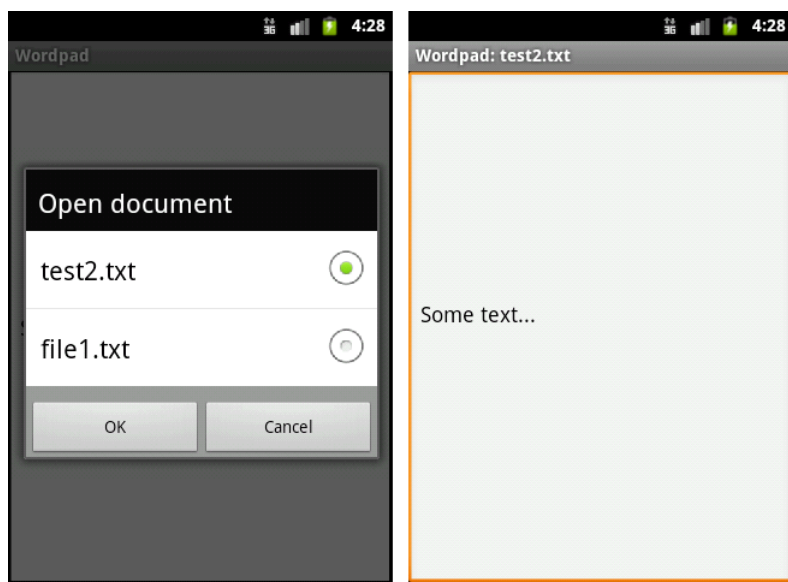


Рис. 16.7. Открытие документа

Резюме

В этой главе мы разобрали структуру файловой системы, доступ к файлам и директориям на мобильном устройстве, проанализировали подключение и использование внешней карты памяти, сохранение и открытие файлов из приложения.

В следующей главе мы рассмотрим виджеты для отображения наборов данных разного типа, предоставляющих текстовую или графическую информацию.



ГЛАВА 17

Адаптеры данных и компоненты для отображения данных

В этой главе рассматриваются виджеты-списки, отображающие информацию, которая может быть связана с внутренним или внешним источником данных. Источниками данных могут быть массивы, списки, внешние базы данных, причем виджеты могут использоваться не только для отображения данных в виде текста, но и для вывода информации, представленной в графическом виде: например, это может быть набор значков, фотографий или рисунков.

Отображение текстовых данных в списках

Элементы-списки в Android представляют пять классов:

- ☐ `ListView`;
- ☐ `GridView`;
- ☐ `Gallery`;
- ☐ `Spinner`;
- ☐ `SlidingDrawer`.

Это контейнерные виджеты, которые (кроме `SlidingDrawer`) являются подклассами `AdapterView`. Эти виджеты можно использовать для связывания с определенным типом данных и отображения их пользователю. Иерархия классов списков представлена на рис. 17.1.

Класс `AdapterView` предоставляет две основные функциональности для работы со списками:

- ☐ заполнение схемы размещения с данными;
- ☐ обработку выбора элемента данных пользователем.

`AdapterView` — подкласс `ViewGroup`, дочерние представления которого определены объектом `Adapter`, который связывает их с данными некоторого типа. Объект `Adapter` действует подобно курьеру между вашим источником данных (например, массивом строк) и `AdapterView`, который отображает эти данные.

Есть несколько реализаций класса `Adapter` для определенных задач, например `CursorAdapter` для чтения данных из объекта `Cursor` или `ArrayAdapter` для чтения из произвольного массива. Иногда необходимо заполнить группу представлений небольшим объемом информации, которая не может быть жестко закодирована, а будет связана

с внешним источником данных, например базой данных SQLite. Чтобы сделать это, необходимо использовать объект `AdapterView` как группу представлений и каждый дочерний `View` инициализировать и заполнять данными от объекта `Adapter`. Подробнее об этом будет рассказано в *главе 15*.

Класс `AdapterView` является базовым для класса `AbsListView`, который может использоваться для реализации классов виджетов, представляющих списки и таблицы (`ListView` и `GridView`), и класса `AbsSpinner` — для выпадающих списков и галереи с прокруткой (`Gallery` и `Spinner`).

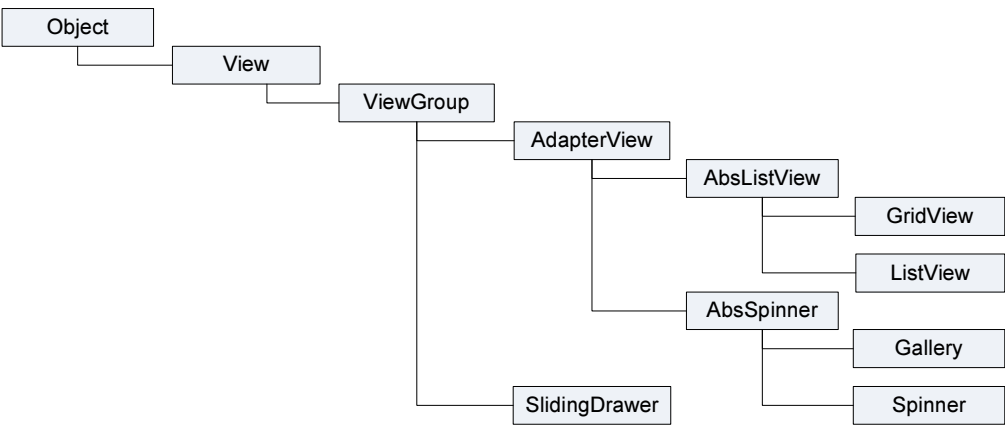


Рис. 17.1. Иерархия классов списков

Адаптеры данных

Для отображения массивов данных в виджетах применяются специальные *адаптеры*, которые предназначены для связывания списка данных и отображающего эти данные виджета. То есть адаптеры служат посредниками, своеобразным мостом между источником данных (базой данных SQLite, файлами, массивами, списками и т. д.) и виджетами для отображения данных.

В зависимости от типа источников данных, используются следующие классы адаптеров данных:

- ❑ `ArrayAdapter<T>;`
- ❑ `SimpleAdapter;`
- ❑ `CursorAdapter;`
- ❑ `ResourceCursorAdapter;`
- ❑ `SimpleCursorAdapter.`

Все эти классы наследуются от абстрактного класса `BaseAdapter` (рис. 17.2).

Самым простым адаптером для использования при связывании данных является шаблонный класс `ArrayAdapter<T>`. Этот класс создает оболочку вокруг массива данных, например, следующим образом:

```
String[] items = {"one", "to", "tree"};
ArrayAdapter<String> adapter = new ArrayAdapter<String>
    (this, android.R.layout.simple_list_item_1, items);
```

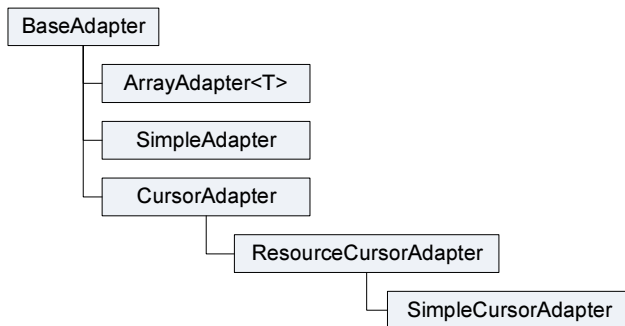


Рис. 17.2. Иерархия классов адаптеров данных

Конструктор класса `ArrayAdapter` принимает три параметра:

- ❑ объект `Context` — обычно это экземпляр класса, реализующий `Activity`. Класс `Context` предоставляет интерфейс среды выполнения прикладной программы. `Context` позволяет получить доступ к специфическим для приложения ресурсам и классам, а также запрашивает операции на уровне приложения, такие, например, как запуск `Activity`, передача и получение объектов `Intent` и т. д.;
- ❑ используемый идентификатор ресурса представления. В данном примере — встроенный системный идентификатор ресурса `simple_list_item_1`. Встроенные идентификаторы ресурса — это константы, определенные в классе `android.R.layout`, например: `simple_spinner_dropdown_item`, `simple_gallery_item`, `simple_list_item_checked` и др., которые, как правило, соответствуют стандартным виджетам;
- ❑ массив или список типа `List<T>` объектов для отображения в виджете.

По умолчанию `ArrayAdapter` вызывает метод `toString()` для объектов списка и создает оболочку для каждой строки в представлении определяемым встроенным системным идентификатором ресурса. `R.layout.simple_list_item_1` просто превращает эти строки в объекты `TextView`, являющиеся, например, элементами контейнерного виджета `ListView` (или любого другого виджета-списка).

Можно также создать собственный класс, наследуемый от класса `ArrayAdapter`, и переопределить в нем метод `getView()` для привязки ваших собственных виджетов, как будет показано далее в этой главе.

Класс `SimpleAdapter` применяется для статического связывания данных небольшого объема. Конструктор класса выглядит так:

```
SimpleAdapter (Context context, List<? extends Map<String, ?>> data,
    int resource,String[] from, int[] to)
```

Данные для `SimpleAdapter` представляются как список объектов `Map`, которые, в свою очередь, содержат данные для каждой строки: множества элементов ключ-значение, где ключ — это имя поля, значение — содержимое поля. То есть каждый элемент в `ArrayList` соответствует одной строке данных в списке.

Класс `SimpleCursorAdapter` используется, как правило, при формировании выборки из больших массивов данных. Этот класс удобно использовать при выборке результата запроса из базы данных SQLite (будет рассматриваться в *главе 18*).

Listview

Виджет `ListView` представляет собой вертикальный список с прокруткой. Связанные со списком данные `ListView` получает от объекта `ListAdapter`. Однако в отличие от предыдущих примеров, когда мы использовали при создании окна приложения в качестве базового класс `Activity`, при работе с `ListView` в качестве базового применяется класс `ListActivity`.

Класс `ListActivity` реализует отображение списка элементов, привязанных к источнику данных, например к массиву, и набор методов обратного вызова для обработки событий выбора элементов списка данных, т. е. `ListActivity` является хостом для объекта `ListView`, который может быть связан с различными источниками данных. У `ListActivity` есть заданная по умолчанию компоновка, которая состоит из единственного списка, растянутого на весь экран (точнее — на родительский контейнер).

Для связывания объекта `ListActivity` с адаптером данных используется метод `setListAdapter()`, которому передается экземпляр созданного адаптера.

В качестве примера использования элемента `ListView` в приложении создадим новый проект и в мастере **Create New Project** введите следующие значения:

- ☐ **Project name** — `ListViewApp`;
- ☐ **Application name** — `ListView Sample`;
- ☐ **Package name** — `com.samples.ui.listview`;
- ☐ **Create Activity** — `ListviewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_ListView`.

Поскольку у класса `ListActivity` уже есть своя собственная компоновка, внешний XML-файл компоновки окна приложения (`main.xml`) нам не нужен. При подключении адаптера данных методом `setListAdapter()` список отобразится в окне приложения. Код класса окна приложения со списком получается очень простым (листинг 17.1).

Листинг 17.1. Файл класса окна приложения `ListviewActivity.java`

```
package com.samples.ui.listview;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;

public class ListviewActivity extends ListActivity {

    @Override
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Массив-источник данных для списка контактов
    String[] contacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Daniel Honeyman", "Isabella Jackson", "William Patterson",
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

    // Создаем адаптер и отображаем данные в списке
    setListAdapter(new ArrayAdapter<String>(
        this, android.R.layout.simple_list_item_1, contacts));
}
```

Если скомпилировать и запустить приложение, на экране будет отображен список, растянутый на весь экран, как представлено на рис. 17.3.

Однако при проектировании окна приложения кроме списка часто требуется размещение дополнительных элементов управления. В таком случае без файла компоновки окна не обойтись. Сейчас мы усовершенствуем наше приложение со списком, добавив к нему текстовое поле, в котором будем отображать информацию о выбранном пользователем элементе списка.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch17_ListViewEvents.

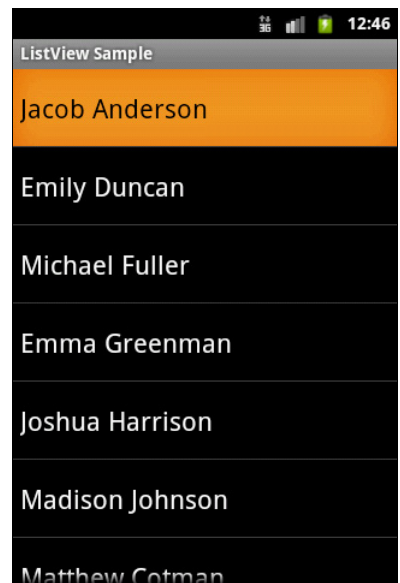


Рис. 17.3. Отображение данных в ListView

Откройте файл main.xml и создайте структуру компоновки с контейнером `LinearLayout` и вложенными виджетами `ListView` для отображения списка и `TextView` для отслеживания события выбора элемента списка, как в листинге 17.2 (в окне **Layout Designer** этот виджет находится в группе **Composite**).

Листинг 17.2. Файл компоновки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```

android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">

<TextView
    android:id="@+id/textSelect"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textStyle="bold"/>

<ListView
    android:id="@android:id/list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:drawSelectorOnTop="false"/>

</LinearLayout>

```

Для обработки события выбора элемента списка в классе `ListActivity` объявлен метод `onListItemClick()`, который необходимо будет определить в нашем классе, производном от `ListActivity`. При выборе пользователем элемента списка в этот метод будут переданы параметры, указывающие на позицию элемента в списке и идентификатор строки списка, которые можно использовать для вывода содержимого этого элемента списка.

Измененный код класса `ListViewActivity` представлен в листинге 17.3.

Листинг 17.3. Файл класса окна приложения `ListViewActivity.java`

```

public class ListViewActivity extends ListActivity {

    private TextView mTextView;

    private static String[] contacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Daniel Honeyman", "Isabella Jackson", "William Patterson",
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

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

        setListAdapter(new ArrayAdapter<String>(
            this, android.R.layout.simple_list_item_1, contacts));
        mTextView = (TextView)findViewById(R.id.textSelect);
    }

    // Обработчик события выбора элемента списка
    public void onListItemClick(
        ListView parent, View v, int position, long id) {

```

```
mTextView.setText(  
    String.format("Select: %s; pos: %s; id: %s",  
        contacts[position], position, id));  
}  
}
```

Запустите проект на выполнение. При выборе элемента списка в текстовом поле в верхней части окна приложения будет отображаться содержимое выбранного элемента, его позиция в списке (отсчет элементов в списке начинается с 0) и идентификатор (рис. 17.4).

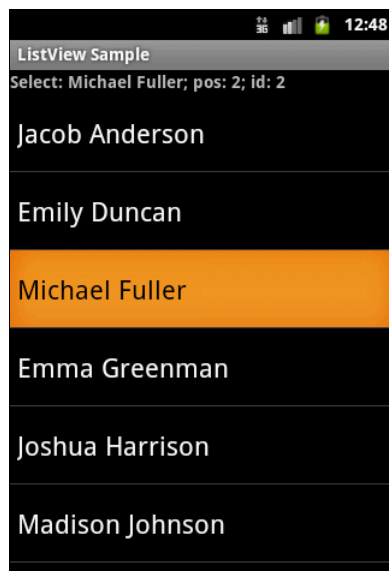


Рис. 17.4. Отображение данных в `ListView`

Загрузка нескольких источников данных в список

Данные в списки можно загружать не только статически, в методе `onCreate()`. Виджет `ListView` сам производит перерисовку при обновлении данных или при изменении источника данных, т. е. допускает динамическое переключение между несколькими источниками данных во время выполнения программы.

Например, нам может потребоваться отобразить сначала наш список с контактами, а потом, при выборе одного из контактов, показать для выбранного контакта детальную информацию. Сначала мы создаем два разных адаптера данных:

```
ArrayAdapter<String> daContacts = new ArrayAdapter<String>(  
    this, android.R.layout.simple_list_item_1, contacts);  
ArrayAdapter<String> daDetails = new ArrayAdapter<String>(  
    this, android.R.layout.simple_list_item_1, details);
```

Затем мы можем переключать список на нужный нам адаптер данных, используя метод `setListAdapter()`, аргументом для которого служит объект адаптера данных:

```
setListAdapter(daDetails);
```

или

```
setListAdapter(daContacts);
```

После подключения другого адаптера к списку необходимо вызвать метод `notifyDataSetChanged()`:

```
daContacts.notifyDataSetChanged();
```

Этот метод посылает сообщение списку, что необходимо обновить данные, которые выводит этот список.

Давайте теперь реализуем это в приложении. Создайте новый проект и в мастере **Create New Project** введите следующие значения:

- ❑ **Project name** — TwoLists;
- ❑ **Application name** — TwoLists;
- ❑ **Package name** — com.samples.ui.twolists;
- ❑ **Create Activity** — TwoListsActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch17_TwoLists.

В классе `TwoListsActivity` создадим два массива со списком контактов и с детальной информацией. При выборе контакта из списка будет загружаться второй список с детальной информацией. Для выхода из списка с детальной информацией у него будет опция **Back on Contacts**, при выборе которой снова загрузится список контактов.

Код класса `TwoListsActivity` представлен в листинге 17.4.

Листинг 17.4. Файл класса окна приложения `TwoListsActivity.java`

```
package com.samples.ui.twolists;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class TwoListsActivity extends ListActivity{

    // Массив-источник данных для списка контактов
    private static final String[] contacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Daniel Honeyman", "Isabella Jackson", "William Patterson",
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

    // Массив-источник данных для списка с детализацией
    private static final String[] details = {
        "Mobile", "Home", "Address", "EMail", "Back on Contacts"};

    private ArrayAdapter<String> daContacts, daDetails;
    private String strMonth, strDayOfWeek;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```



```

// Создаем адаптер данных для списка контактов
daContacts = new ArrayAdapter<String>(
    this, android.R.layout.simple_list_item_1, contacts);
// Создаем адаптер данных для списка с детализацией
daDetails = new ArrayAdapter<String>(
    this, android.R.layout.simple_list_item_1, details);

setListAdapter(daContacts);
}

@Override
protected void onListItemClick(ListView l, View v, int pos, long id) {
    super.onListItemClick(l, v, pos, id);

    if(getListAdapter() == daContacts) {
        // Переходим на список с детализацией
        setListAdapter(daDetails);
        daDetails.notifyDataSetChanged();
    }
    else {
        if ((String)getListView().getItemAtPosition(pos) ==
            "Back on Contacts") {
            // Переходим на список контактов
            setListAdapter(daContacts);
            daContacts.notifyDataSetChanged();
        }
    }
}
}
}

```

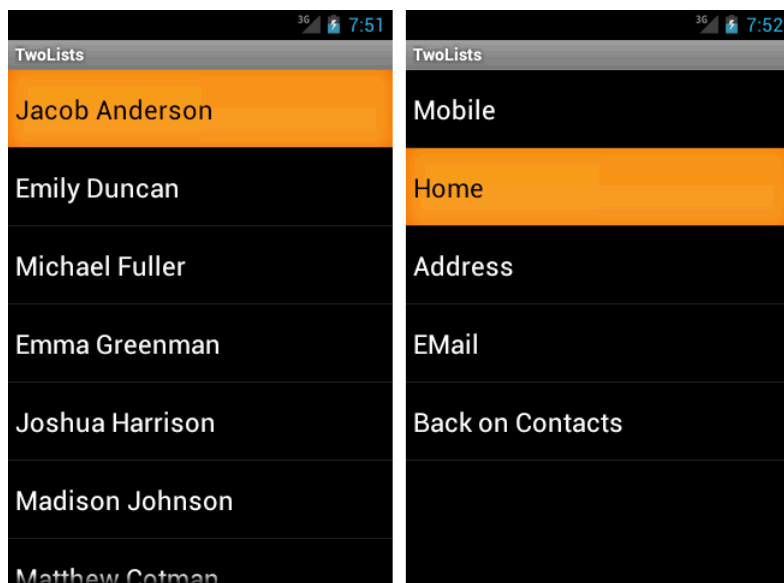


Рис. 17.5. Загрузка нескольких источников данных в один список

Запустите проект на выполнение. При выборе элемента из списка контактов будет загружен другой список, предоставляющий доступ к детальной информации выбранного контакта (рис. 17.5).

Список с единичным и множественным выбором

Если есть необходимость, можно использовать списки с единичным (с переключателями) или с множественным выбором (с флажками). Эти списки похожи на диалоговые окна и меню с переключателями и флажками (см. главы 10, 11).

Сначала рассмотрим список с единичным выбором. Создайте новый проект:

- ❑ **Project name** — `SingleChoiceList`;
- ❑ **Application name** — `ListView with Radio Buttons`;
- ❑ **Package name** — `com.samples.ui.singlechoicelist`;
- ❑ **Create Activity** — `SingleChoiceListActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_SingleChoiceList`.

При создании списка с единичным выбором компоновка для адаптера данных определяется значением `android.R.layout.simple_list_item_single_choice`. Однако просто выполнить метод `setListAdapter()` для привязки адаптера данных к `ListView`, недостаточно, надо еще задать режим отображения с помощью метода `setChoiceMode()` класса `ListView` с параметром `CHOICE_MODE_SINGLE`.

Сам объект `ListView` можно получить через вызов `getListView()`, который определен в классе `ListActivity`. Класс `SingleChoiceListActivity` показан в листинге 17.5.

Листинг 17.5. Метод `onCreate()` для списка с `SingleChoiceListActivity`

```
package com.samples.ui.singlechoicelist;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class SingleChoiceListActivity extends ListActivity {

    private static final String[] contacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Daniel Honeyman", "Isabella Jackson", "William Patterson",
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

// Создаем адаптер данных для списка
setListAdapter(new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_single_choice, contacts));

// Получаем объект ListView
final ListView listView = getListView();
listView.setItemsCanFocus(false);

// Устанавливаем режим отображения
listView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
}
}

```

Внешний вид окна приложения для списка с переключателями представлен на рис. 17.6.

Создание списка с множественным выбором аналогично созданию списка с переключателями. Отличие состоит в том, что компоновка для адаптера данных с множественным выбором определяется значением `android.R.layout.simple_list_item_multiple_choice`, и соответственно, после получения экземпляра `ListView` устанавливаем режим отображения для списка в значение `ListView.CHOICE_MODE_MULTIPLE`, как показано в листинге 17.6.

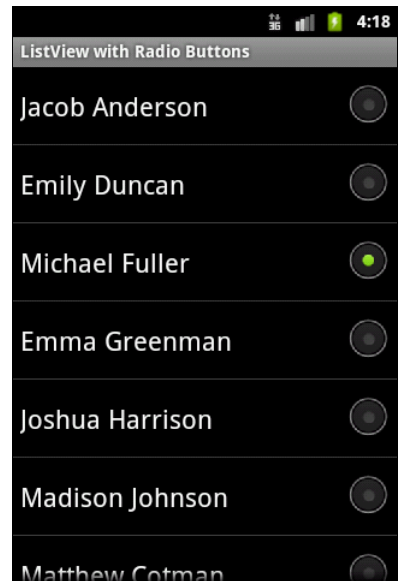


Рис. 17.6. Список с единичным выбором

Листинг 17.6. Метод `onCreate()` для списка с флажками

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Создаем адаптер данных для списка
    setListAdapter(new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_multiple_choice, contacts));

    // Получаем объект ListView
    final ListView listView = getListView();
    listView.setItemsCanFocus(false);

    // Устанавливаем режим отображения
    listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
}
}

```

Внешний вид окна для списка с флажками будет выглядеть так, как на рис. 17.7.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch17_MultipleChoiceList.

Кстати, при использовании списков с переключателями и флажками очень часто забывают вызвать метод `setChoiceMode()` с соответствующим параметром. При этом список будет отображаться как обычно, но состояние переключателя или флажка в выбранном элементе списка невозможно будет зафиксировать, после нажатия он будет сразу сбрасываться.

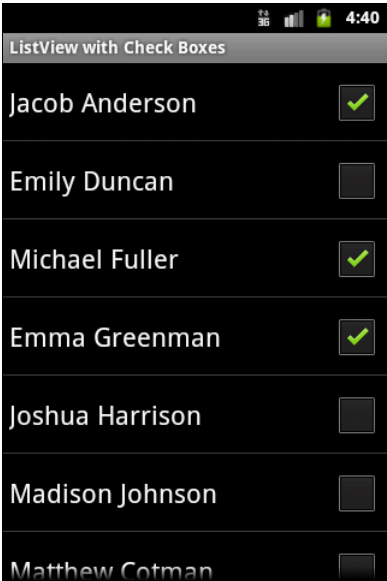


Рис. 17.7. Список с множественным выбором

Создание списка с нестандартной компоновкой

Кроме использования стандартных виджетов-списков можно создать список, определив собственную компоновку только для одной строки этого списка. Такой подход часто используется для связывания данных, представленных в виде плоских таблиц. Для связывания табличных данных используются классы `SimpleAdapter` и `SimpleCursorAdapter`.

В качестве примера разработаем приложение, отображающее статические данные в табличном виде. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `ListContact`;
- ☐ **Application name** — `Contacts Sample`;
- ☐ **Package name** — `com.samples.ui.listcontact`;
- ☐ **Create Activity** — `ListContactActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch17_ListContacts.

В приложении будем отображать список контактов, приведенный ранее, но в строке будет два столбца: поле `Name` с выравниванием влево и поле `Phone` с выравниванием вправо.

Для этого в файле компоновки определим строку с двумя элементами `TextView`, как показано в листинге 17.7.

Листинг 17.7. Файл компоновки main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:textSize="18sp"/>

    <TextView
        android:id="@+id/phone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:textSize="18sp"
        android:paddingRight="10px"/>

</RelativeLayout>

```

Для хранения строки, представляющей контакт, создадим отдельный класс `ContactItem`, расширяющий класс `HashMap`, в котором определим константы `NAME` и `PHONE` с именами полей и конструктор с параметрами — имя контакта и телефон. Код класса представлен в листинге 17.8.

Листинг 17.8. Класс `ContactItem`

```

package com.samples.ui.listcontact;

import java.util.HashMap;

public class ContactItem extends HashMap<String, String> {
    public static final String NAME = "name";
    public static final String PHONE = "phone";

    public ContactItem(String name, String phone) {
        super();
        super.put(NAME, name);
        super.put(PHONE, phone);
    }
}

```

В классе окна приложения `ListContactActivity` заполним данными объект `ArrayList` и отобразим его в виде таблицы (листинг 17.9).

Листинг 17.9. Класс ListContactActivity

```

package com.samples.ui.listcontact;

import java.util.ArrayList;

import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ListAdapter;
import android.widget.SimpleAdapter;

public class ListContactActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ArrayList<ContactItem> list = new ArrayList<ContactItem>();

        // Заполняем список контактов
        list.add(new ContactItem("Jacob Anderson", "412412411"));
        list.add(new ContactItem("Emily Duncan", "161863187"));
        list.add(new ContactItem("Michael Fuller", "896443658"));
        list.add(new ContactItem("Emma Greenman", "964990543"));
        list.add(new ContactItem("Joshua Harrison", "759285086"));
        list.add(new ContactItem("Madison Johnson", "950285777"));
        list.add(new ContactItem("Matthew Cotman", "687699999"));
        list.add(new ContactItem("Olivia Lawson", "161863187"));
        list.add(new ContactItem("Andrew Chapman", "546599645"));
        list.add(new ContactItem("Daniel Honeymen", "876545644"));
        list.add(new ContactItem("Isabella Jackson", "907868756"));
        list.add(new ContactItem("William Patterson", "687699693"));
        list.add(new ContactItem("Joseph Godwin", "965467575"));
        list.add(new ContactItem("Samantha Bush", "907865645"));
        list.add(new ContactItem("Christopher Gateman", "896874556"));

        // Создаем адаптер данных
        ListAdapter adapter = new SimpleAdapter(
            this, list, R.layout.main,
            new String[] {ContactItem.NAME, ContactItem.PHONE},
            new int[] {R.id.name, R.id.phone});
        setListAdapter(adapter);
    }
}

```

Внешний вид приложения со списком контактов представлен на рис. 17.8.

Это приложение нам пригодится в дальнейшем. Мы будем развивать его в следующих главах книги при рассмотрении работы с базами данных и изучении компонента Content Provider (см. главы 18, 19).



Contacts Sample	
Jacob Anderson	412412411
Emily Duncan	161863187
Michael Fuller	896443658
Emma Greenman	964990543
Joshua Harrison	759285086
Madison Johnson	950285777
Matthew Cotman	687699999
Olivia Lawson	161863187
Andrew Chapman	546599645
Daniel Honeyman	876545644
Isabella Jackson	907868756
William Patterson	687699693
Joseph Godwin	965467575
Samantha Bush	907865645
Christopher Gateman	896874556

Рис. 17.8. Список с компоновкой строки, определенной в отдельном XML-файле

ListFragment

С появлением в версии Android 3.0 фрагмента как нового компонента пользовательского интерфейса для устройств с большим разрешением экрана появился соответственно и фрагмент, реализующий список и представленный классом `ListFragment`. По сути, он является фрагментом, который содержит `ListView`, и может выводить на экран список элементов от источника данных, такого как массив или курсор.

Фрагмент списка очень полезен для случаев, когда надо выводить общую информацию в виде списка и рядом, в другом фрагменте, детализированную информацию. Тогда общую информацию мы выведем в фрагмент-список, а детализированные данные для выбранного элемента списка — в обычный фрагмент.

Чтобы создать фрагмент-список, надо наследовать его от базового класса `ListFragment`:

```
public class Fragment1 extends ListFragment {
    ...
}
```

Все остальные действия аналогичны работе с обычным `ListView`. Рассмотрим пример — создайте в Eclipse новый проект:

- ☐ **Project name** — `ListFragment`;
- ☐ **Application name** — `List Fragment`;
- ☐ **Package name** — `com.samples.data.listfragment`;
- ☐ **Create Activity** — `ListFragmentActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_ListFragment`.

В файле компоновки Activity мы разместим объявление двух фрагментов, как показано в листинге 17.10.

Листинг 17.10. Файл компоновки контейнерного Activity main.xml

```
<?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="horizontal" >

    <fragment
        android:name="com.samples.data.listfragment.Fragment1"
        android:id="@+id/fragment1"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <fragment
        android:name="com.samples.data.listfragment.Fragment2"
        android:id="@+id/fragment2"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

В файле компоновки первого фрагмента мы разместим список `ListView` (листинг 17.11).

Листинг 17.11. Файл компоновки первого фрагмента fragment1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="8dp"
    android:paddingRight="8dp">

    <ListView android:id="@+id/android:list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:drawSelectorOnTop="false"/>

</LinearLayout>
```

В файле компоновки второго фрагмента будет располагаться текстовое поле `TextView` для вывода детальной информации (листинг 17.12).

Листинг 17.12. Файл компоновки второго фрагмента fragment2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text"
        android:textSize="18sp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

В файле контейнерного Activity мы не будем производить никаких действий (листинг 17.13).

Листинг 17.13. Файл класса контейнерного Activity ListFragmentActivity.java

```
package com.samples.data.listfragment;

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

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

В классе `Fragment1` в теле метода `onCreate()` мы используем такое же создание адаптера данных и метод `setListAdapter()`, как и в случае с обычным списком `ListView`. Поскольку мы должны будем отображать детальную информацию во втором фрагменте, в классе первого фрагмента в теле метода `onStart()` надо создать ссылку на текстовое поле второго фрагмента, используя вызов `getActivity().findViewById()`.

Код класса `Fragment1` представлен в листинге 17.14.

Листинг 17.14. Файл класса фрагмента-списка Fragment1.java

```
package com.samples.data.listfragment;

import android.app.ListFragment;
import android.os.Bundle;
import android.view.LayoutInflater;
```

```
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.TextView;

public class Fragment1 extends ListFragment {

    private static final String[] contacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Daniel Honeyman", "Isabella Jackson", "William Patterson",
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

    private static final String[] phones = {
        "412412411", "161863187", "896443658", "964990543", "759285086",
        "950285777", "687699999", "161863187", "546599645", "876545644",
        "907868756", "687699693", "965467575", "907865645", "896874556"};

    private TextView text;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListAdapter listAdapter = new ArrayAdapter<String>(
            getActivity(), android.R.layout.simple_list_item_1, contacts);
        setListAdapter(listAdapter);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment1, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();
        // Текстовое поле, находящееся во втором фрагменте
        text = (TextView) getActivity().findViewById(R.id.text);
    }

    @Override
    public void onItemClick(ListView l, View v, int position, long id) {
        // Получаем текст, находящийся в выбранном контакте
        String selectedItem =
            getListView().getItemAtPosition(position).toString();
    }
}
```

```
// Отображаем текст во втором фрагменте
text.setText("Contact's details\n\n\tName: " +
            selectedItem + "\n\n\tPhone: " + phones[position]);
}
```

Класс, представляющий второй фрагмент, будет полностью аналогичен листингу 14.4 из главы 14.

Скомпилируйте и запустите проект на выполнение. У нас получилось приложение с двумя фрагментами, в левом фрагменте отображается весь список, а в правом — детальная информация о выбранном элементе списка (рис. 17.9).

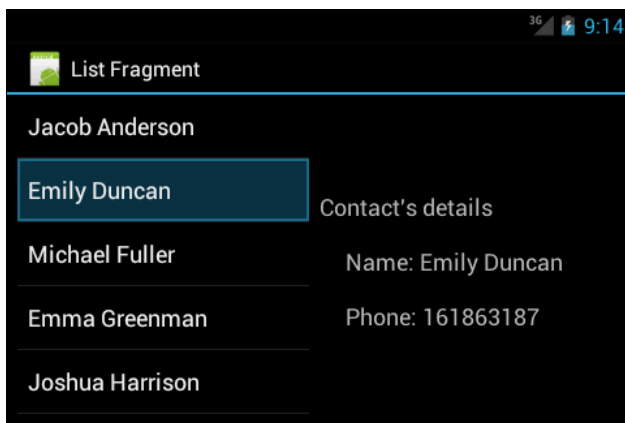


Рис. 17.9. ListFragment и фрагмент с детализированной информацией

GridView

Виджет `GridView` представляет собой *плоскую таблицу*. Для `GridView`, вместо того чтобы использовать автоматически генерируемые виджеты `TextView`, как в случае с элементом `ListView`, можно использовать собственные поля для отображения элементов данных, создав класс, производный от класса `ArrayAdapter`, и переопределив его метод `getView()`.

Число столбцов для `GridView` чаще задается статически. Число строк в элементе определяется динамически на основании числа элементов, которые предоставляет адаптер. Есть несколько свойств, определяющих число столбцов и их размеры:

- ❑ `android:numColumns` — определяет количество столбцов. Если поставлено значение `auto_fit`, то система вычислит количество столбцов, основанное на доступном пространстве;
- ❑ `android:verticalSpacing` — устанавливает размер пустого пространства между ячейками таблицы;
- ❑ `android:columnWidth` — устанавливает ширину столбцов;

- ❑ `android:stretchMode` — указывает, куда распределяется остаток свободного пространства для таблицы с установленным значением `android:numColumns="auto_fit"`. Принимает значения `columnWidth` для распределения остатка свободного пространства между ячейками столбца для их увеличения или `spacingWidth` — для увеличения пространства между ячейками.

Давайте в качестве примера использования элемента `GridView` в приложении для отображения текстовой информации создадим в Eclipse новый проект:

- ❑ **Project name** — `GridViewApp`;
- ❑ **Application name** — `GridView Sample`;
- ❑ **Package name** — `com.samples.ui.gridview`;
- ❑ **Create Activity** — `GridViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_GridViewText`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 17.15.

Листинг 17.15. Файл компоновки `main.xml`

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

    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    <GridView
        android:id="@+id/grid"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:verticalSpacing="35px"
        android:horizontalSpacing="5px"
        android:numColumns="auto_fit"
        android:columnWidth="100px"
        android:stretchMode="columnWidth"
        android:gravity="center"/>

</LinearLayout>
```

Код класса-оболочки для данных `DataAdapter`, производного от `ArrayAdapter`, представлен в листинге 17.16.

Листинг 17.16. Файл класса адаптера данных DataAdapter.java

```
package com.samples.ui.gridviewtext;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.TextView;

public class DataAdapter extends ArrayAdapter<String>
{
    private static final String[] contacts = {
        "J. Anderson", "E. Duncan", "M. Fuller",
        "E. Greenman", "J. Harrison", "M. Johnson",
        "M. Cotman", "O. Lawson", "A. Chapman",
        "M. Honeyman", "I. Jackson", "W. Patterson",
        "J. Godwin", "S. Bush", "C. Gateman",
        "J. Anderson", "E. Duncan", "M. Fuller",
        "E. Greenman", "J. Harrison", "M. Johnson",
        "M. Cotman", "O. Lawson", "A. Chapman",
        "M. Honeyman", "I. Jackson", "W. Patterson",
        "J. Godwin", "S. Bush", "C. Gateman" };
    private Context context;

    // Конструктор
    public DataAdapter(Context con, int resource) {
        super(con, resource, contacts);
        this.context = con;
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        TextView label = (TextView)convertView;

        if (convertView == null) {
            convertView = new TextView(context);
            label = (TextView)convertView;
        }
        label.setText(contacts[position]);

        return (convertView);
    }

    // Возвращает содержимое выделенного элемента списка
    public String GetItem(int position) {
        return contacts[position];
    }
}
```


Скомпилируйте проект и запустите на выполнение. Приложение выдаст список фамилий в виде таблицы. При выборе элемента списка в окне текстового поля вверху экрана будет отображаться содержимое выбранного элемента (рис. 17.10).

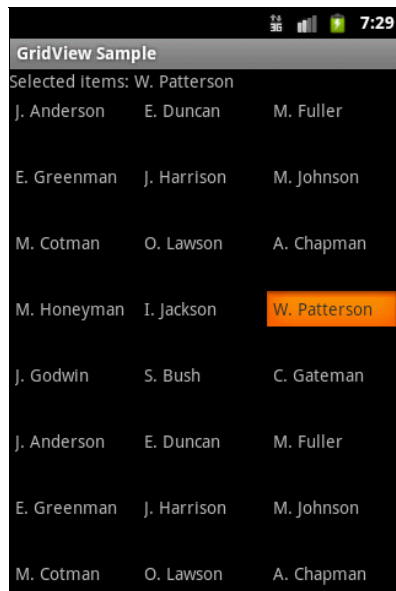


Рис. 17.10. Приложение с виджетом GridView

Отображение графики в списках

В библиотеке Android существуют специализированные виджеты для отображения графической информации. Кроме того, в виджетах-списках помимо текстовых данных можно также отображать графику. В этом разделе мы сначала изучим загрузку информации смешанного типа в список, а затем рассмотрим привязку графических данных к виджету GridView и изучим еще два специализированных виджета — Gallery и SlidingDrawer.

Отображение графики в GridView

Привязка графики к GridView не представляет никаких трудностей. Необходимо только источник данных (в нашем примере — это статический массив) связать с внешними изображениями, размещенными в ресурсах.

В качестве примера приложения с использованием GridView для отображения графической информации создайте в Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — GridViewImageApp;
- ☐ **Application name** — GridView with image Sample;
- ☐ **Package name** — com.samples.ui.gridview;
- ☐ **Create Activity** — GridViewActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch17_GridViewImage.

Для окна используйте файл компоновки, созданный в предыдущем примере приложения с помощью `GridView` для отображения текстовых данных (см. листинг 17.15).

Класс адаптера для связывания графических данных необходимо наследовать от `BaseAdapter`. Это базовый класс общей реализации адаптеров данных, который может применяться в списках. В предыдущих примерах для привязки текстовых данных мы использовали в качестве базового класса специализированный класс `ArrayAdapter<T>`.

В классе адаптера массив данных должен содержать идентификаторы графических ресурсов, которые будут располагаться в каталоге `res/drawable/`. Для данного примера возьмите файлы `photo1.jpg...photo8.jpg` из каталога `Resources/Images/` архива книги.

Создайте в Eclipse новый `java`-класс для адаптера списка `ImageAdapter`, который будет наследоваться от класса `BaseAdapter`. Полный код класса `ImageAdapter` представлен в листинге 17.18.

Листинг 17.18. Файл `ImageAdapter.java`

```
package com.samples.ui.gridviewimage;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
    private Context context;

    private static final int[] imagess = {
        R.drawable.photo1, R.drawable.photo2,
        R.drawable.photo3, R.drawable.photo4,
        R.drawable.photo5, R.drawable.photo6,
        R.drawable.photo7, R.drawable.photo8,
        R.drawable.photo1, R.drawable.photo2,
        R.drawable.photo3, R.drawable.photo4,
        R.drawable.photo5, R.drawable.photo6,
        R.drawable.photo7, R.drawable.photo8,
        R.drawable.photo1, R.drawable.photo2,
        R.drawable.photo3, R.drawable.photo4,
        R.drawable.photo5, R.drawable.photo6,
        R.drawable.photo7, R.drawable.photo8
    };

    public ImageAdapter(Context con) {
        context = con;
    }

    public int getCount() {
        return imagess.length;
    }
}
```



```

public Object getItem(int position) {
    return imagess[position];
}

public long getItemId(int position) {
    return imagess[position];
}

// Создание нового ImageView для каждого элемента данных
public View getView(
    int position, View convertView, ViewGroup parent) {
    ImageView view;
    if (convertView == null) {
        view = new ImageView(context);
        view.setLayoutParams(new GridView.LayoutParams(85, 85));
        view.setScaleType(ImageView.ScaleType.CENTER_CROP);
        view.setPadding(2, 2, 2, 2);
    }
    else {
        view = (ImageView)convertView;
    }

    view.setImageResource(imagess[position]);
    return view;
}
}

```

Класс, реализующий окно приложения, также останется практически без изменений (см. листинг 17.17) за исключением кода привязки данных в методе `onCreate()`:

```

adapter = new ImageAdapter(getApplicationContext());
grid.setAdapter(adapter);

```

Сделав необходимые изменения в коде программы, запустите проект на выполнение. При выборе элемента списка в окне текстового поля вверху экрана будет отображен идентификатор выбранного элемента (рис. 17.11).

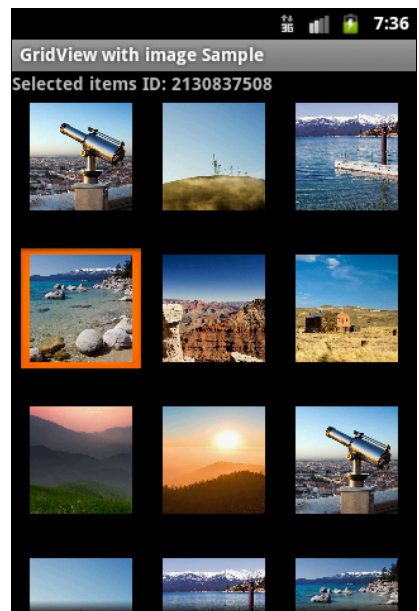


Рис. 17.11. Приложение с виджетом `GridView`, отображающим графику

Загрузка изображений и текста в список

Часто в списке необходимо отображать комбинированную информацию — текст вместе с картинкой или значком. Например, нужно отобразить список контактов с фотографиями и текстовой информацией для каждого контакта. Рассмотрим это, взяв за основу наш список контактов и добавив к нему изображения.

Создайте в Eclipse новый проект:

- ❑ **Project name** — `ListViewImageText`;
- ❑ **Application name** — `ListView with Image and Text`;
- ❑ **Package name** — `com.samples.data.listviewimagetext`;
- ❑ **Create Activity** — `ListIconsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_ListViewImageText`.

Чтобы создать такой список, нам понадобится нестандартная компоновка для строки списка. В файле компоновки `row.xml` мы используем виджет `ImageView` для отображения значка из ресурсов и `TextView` для текста, как представлено в листинге 17.19.

Листинг 17.19. Файл компоновки `row.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <ImageView
        android:id="@+id/option_icon"
        android:layout_width="48dp"
        android:layout_height="fill_parent"/>
    <TextView
        android:id="@+id/option_text"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="10dp"
        android:textSize="16dp" />
</LinearLayout>
```

Для комбинированного адаптера данных, так же как и в предыдущем примере, мы создадим отдельный класс, который будет наследоваться от `BaseAdapter`. Однако реализация этого класса будет несколько сложнее, чем в листинге 17.18 при создании адаптера для графических данных. В новом классе адаптера нам понадобится экземпляр `LayoutInflater` для получения контейнерного объекта `View`, представляющего строку списка, определенную в `row.xml`, и состоящего из элементов `ImageView` и `TextView` (класс `LayoutInflater` мы уже использовали в главах 9 и 10 при создании нестандартных уведомлений и диалоговых окон).

Поскольку мы находимся не в классе, реализующем Activity, нам будет недоступен метод `getLayoutInflater()`, но существует еще один способ получения объекта `LayoutInflater` — через вызов системной службы `Layout Inflater Service`. В классе `Context` есть метод `getSystemService()` для вызова системных служб, которому надо передать параметр со значением `Context.LAYOUT_INFLATER_SERVICE`.

```
LayoutInflater inflater = (LayoutInflater)context.getSystemService(  
    Context.LAYOUT_INFLATER_SERVICE);
```

Системные службы и способы получения доступа к ним подробно мы будем проходить позже, начиная с главы 24, а пока мы просто используем вызов службы `Layout Inflater Service` для получения объекта `LayoutInflater`. После создания экземпляра `LayoutInflater` мы получаем доступ к XML-компоновке строки для отображения данных (`row.xml`) и можем с помощью уже известного нам метода `findViewById()` добраться до полей `ImageView` и `TextView`. Код адаптера данных представлен в листинге 17.20.

Листинг 17.20. Файл класса адаптера данных `CustomAdapter.java`

```
package com.samples.data.listviewimagetext;  
  
import android.content.Context;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.AdapterView;  
import android.widget.BaseAdapter;  
import android.widget.ImageView;  
import android.widget.TextView;  
  
public class CustomAdapter extends BaseAdapter {  
    private LayoutInflater inflater;  
  
    private static final String[] contacts = {  
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",  
        "Emma Greenman", "Joshua Harrison", "Madison Johnson",  
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",  
        "Daniel Honeymen", "Isabella Jackson", "William Patterson",  
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};  
  
    private static final int[] images = {  
        R.drawable.a1, R.drawable.a2, R.drawable.a3,  
        R.drawable.a4, R.drawable.a5, R.drawable.a6,  
        R.drawable.a7, R.drawable.a8, R.drawable.a9,  
        R.drawable.a10, R.drawable.a11, R.drawable.a12,  
        R.drawable.a13, R.drawable.a14, R.drawable.a15 };  
  
    public CustomAdapter(Context context) {  
        super();
```

```

        inflater = (LayoutInflater) context.getSystemService(
            Context.LAYOUT_INFLATER_SERVICE);
    }

    @Override
    public int getCount() {
        return contacts.length;
    }

    @Override
    public String getItem(int position) {
        return contacts[position];
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    // Создание новой комбинированной строки
    // для каждой пары элементов данных
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // Получаем компоновку строки
        View view = inflater.inflate(R.layout.row, null);

        // Загружаем в строку данные из массивов
        ImageView image = (ImageView) view.findViewById(R.id.image);
        image.setImageResource(images[position]);

        TextView text = (TextView) view.findViewById(R.id.text);
        text.setText(contacts[position]);

        return view;
    }
}

```

Класс `ListIconsActivity` не имеет каких-либо новых особенностей, мы используем стандартный метод `setListAdapter()` для загрузки данных в список. Код класса `ListIconsActivity` показан в листинге 17.21.

Листинг 17.21. Файл класса окна приложения `ListIconsActivity.java`

```

package com.samples.data.listviewimagetext;

import android.app.ListActivity;
import android.os.Bundle;

public class ListIconsActivity extends ListActivity {
    @Override

```

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    //setContentView(R.layout.main);  
  
    setListAdapter(new CustomAdapter(getApplicationContext()));  
}  
}
```

Запустите проект на выполнение. Теперь наш однообразный список с текстовой информацией смотрится гораздо интереснее, чем раньше (рис. 17.12)!

Далее в этой главе мы покажем, какие виджеты предоставляет библиотека Android для отображения графических данных самого разнообразного типа и вида.

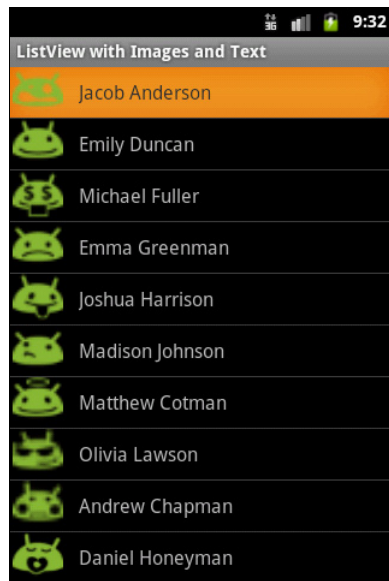


Рис. 17.12. Список, отображающий текст и значки

Gallery

Виджет Gallery — это окно списка с графическим наполнением, имеющее горизонтальную прокрутку и подсветку выбранного изображения. На мобильном устройстве список перемещают с помощью левых и правых кнопок со стрелками на D-pad (навигаторе) или прокручивая изображения пальцем. Чаще всего элемент Gallery используется как средство просмотра коллекции фотографий или значков.

Это простой в использовании виджет, и работа с ним в коде программы в целом аналогична работе с обычными списками. В качестве примера приложения с использованием элемента Gallery создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — GalleryApp;
- ☐ **Application name** — Gallery Sample;
- ☐ **Package name** — com.samples.ui.gallery;
- ☐ **Create Activity** — GalleryActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch17_Gallery.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 17.22.

Листинг 17.22. Файл компоновки main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <Gallery
        android:id="@+id/gallery"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="24px"/>

</LinearLayout>
```

Код класса-адаптера данных `ImageAdapter` представлен в листинге 17.23. Различие между классами адаптеров для `GridView` и `Gallery` будет только в методе `getView()`, загружающем графические ресурсы в виджет.

Листинг 17.23. Файл класса адаптера данных ImageAdapter.java

```
package com.samples.ui.gallery;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {

    private int mGalleryItemBackground;
    private Context context;

    private final Integer[] images = {
        R.drawable.photo1, R.drawable.photo2,
        R.drawable.photo3, R.drawable.photo4,
        R.drawable.photo5, R.drawable.photo6,
        R.drawable.photo7, R.drawable.photo8,
    };
};
```

```
public ImageAdapter(Context con) {
    context = con;
}

public View getView(
    int position, View convertView, ViewGroup parent) {
    ImageView view = new ImageView(context);

    view.setImageResource(images[position]);
    view.setPadding(20,20,20,20);
    view.setLayoutParams(new Gallery.LayoutParams(140, 190));
    view.setScaleType(ImageView.ScaleType.FIT_XY);
    view.setBackgroundResource(mGalleryItemBackground);
    return view;
}

public int getCount() {
    return images.length;
}

public Object getItem(int position) {
    return images[position];
}

public long getItemId(int position) {
    return images[position];
}
}
```

Полный код класса `GalleryActivity` представлен в листинге 17.24.

Листинг 17.24. Файл класса окна приложения `GalleryActivity.java`

```
package com.samples.ui.gallery;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Gallery;
import android.widget.TextView;
import android.widget.AdapterView.OnItemClickListener;

public class GalleryActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

```

final Gallery g = (Gallery) findViewById(R.id. gallery);
g.setAdapter(new ImageAdapter(this));

final TextView label = (TextView) findViewById(R.id.text);
label.setText("Slide 1 from " + g.getAdapter().getCount());

g.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent,
        View v, int pos, long id) {
        label.setText(
            "Slide " + ++pos + " from " + parent.getCount());
    }
});
}
}

```

Скомпилируйте проект и запустите на выполнение. При выборе картинки из галереи в окне текстового поля внизу картинки также будет отображаться индекс выбранного элемента и полное количество изображений в коллекции, как показано на рис. 17.13.

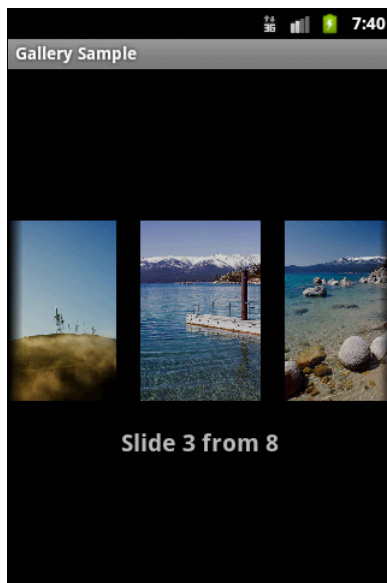


Рис. 17.13. Приложение с виджетом Gallery

SlidingDrawer

Виджет `SlidingDrawer` — это выдвижная панель с маркером. Данный виджет используется в некоторых моделях мобильных устройств с Android на панели **Application Launcher**, которая отображает список программ, установленных на устройстве.

В неактивном состоянии `SlidingDrawer` скрыт, и на экране виден только маркер. При нажатии маркера пользователем выдвигается информационная панель. `SlidingDrawer` может использоваться вертикально или горизонтально. Специальный графический фрагмент состоит из двух дочерних представлений: маркера, который может перемещаться пользователем, и информационного наполнения, прикрепленного к маркеру и перемещаемого вместе с маркером.

Размер `SlidingDrawer` определяет пространство, которое будет занимать его информационное наполнение при выдвинутой панели `SlidingDrawer`. Обычно для определе-

ния высоты и ширины используется значение `fill_parent`. В XML-компоновке `SlidingDrawer` необходимо определить ссылку на маркер (это отдельный графический ресурс) и контейнер для информационного наполнения.

Можно создать собственный `SlidingDrawer` и использовать его в своем приложении. Поскольку `SlidingDrawer` является контейнерным виджетом, информационное наполнение может быть любое — текст, графика или контейнер с дочерними виджетами. Если содержимое панели не помещается на экране, автоматически добавляется вертикальная полоса прокрутки.

При создании XML-компоновки для `SlidingDrawer` необходимо определить ресурс для маркера и информационного наполнения:

```
android:handle="@+id/handle"
android:content="@+id/content"
```

В качестве примера приложения с использованием виджета `SlidingDrawer` создайте в среде Eclipse новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `SlidingDrawerApp`;
- ☐ **Application name** — `SlidingDrawer Sample`;
- ☐ **Package name** — `com.samples.ui.slidingdrawer`;
- ☐ **Create Activity** — `SlidingDrawerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_SlidingDrawer`.

В нашем примере для информационного наполнения будет использоваться `GridView`, как в предыдущем примере. Иконки можно взять из каталога `Resources/Images/` архива книги или использовать свои собственные значки. Все изображения, использованные в примерах, взяты из Android SDK.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 17.25.

Листинг 17.25. Файл компоновки `main.xml`

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

    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    <SlidingDrawer
        android:id="@+id/drawer"
        android:layout_width="fill_parent"
```

```

    android:layout_height="fill_parent"
    android:handle="@+id/handle"
    android:content="@+id/content"
    android:bottomOffset="9px">

    <ImageView
        android:id="@id/handle"
        android:layout_width="320dip"
        android:layout_height="50dip"
        android:src="@drawable/handle"/>

    <LinearLayout
        android:id="@id/content"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <GridView
            android:id="@+id/grid"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:numColumns="auto_fit"
            android:verticalSpacing="10dp"
            android:horizontalSpacing="10dp"
            android:columnWidth="60dp"
            android:stretchMode="columnWidth"
            android:gravity="center"/>

    </LinearLayout>
</SlidingDrawer>
</LinearLayout>

```

Код класса `SlidingDrawerActivity` в целом похож на код класса для работы с элементом `GridView` (листинг 17.26).

Листинг 17.26. Файл класса окна приложения `SlidingDrawerActivity.java`

```

package com.samples.ui.slidingdrawer;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.GridView;
import android.widget.TextView;

public class SlidingDrawerActivity extends Activity
    implements AdapterView.OnItemClickListener,
        AdapterView.OnItemSelectedListener {

    private TextView selectText;
    private ImageAdapter adapter;

```

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

    selectText=(TextView)findViewById(R.id.label);

    GridView gridview = (GridView)findViewById(R.id.grid);
    adapter = new ImageAdapter(this);
    gridview.setAdapter(adapter);
    gridview.setOnItemClickListener(this);
}

@Override
public void onItemClick(AdapterView<?> parent,
    View v, int position, long id) {
    selectText.setText("Selected items ID: " +
        adapter.getItemId(position));
}

@Override
public void onNothingSelected(AdapterView<?> parent) {
    selectText.setText("Selected items: none");
}
}
```

В классе-оболочке данных `ImageAdapter` заполните массив данных ссылками на идентификаторы значков, которые вы будете использовать для наполнения виджета.

Полный код класса представлен в листинге 17.27.

Листинг 17.27. Файл класса адаптера данных `ImageAdapter.java`

```
package com.samples.ui.slidingdrawer;

import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {

    private Context context;

    // Массив элементов для наполнения виджета
    private Integer[] imagess = {
        R.drawable.ic_launcher_allhide,
        R.drawable.ic_launcher_android,
```

```
R.drawable.ic_launcher_browser,
R.drawable.ic_launcher_calculator,
R.drawable.ic_launcher_calendar,
R.drawable.ic_launcher_camera,
R.drawable.ic_launcher_contacts,
R.drawable.ic_launcher_email,
R.drawable.ic_launcher_email_generic,
R.drawable.ic_launcher_gallery,
R.drawable.ic_launcher_google_talk,
R.drawable.ic_launcher_home,
R.drawable.ic_launcher_im,
R.drawable.ic_launcher_maps,
R.drawable.ic_launcher_musicplayer_2,
R.drawable.ic_launcher_phone_dialer };

public ImageAdapter(Context con) {
    context = con;
}

public View getView(int position,
    View convertView, ViewGroup parent) {
    ImageView imageView;

    if (convertView == null) {
        imageView = new ImageView(context);
        imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setPadding(8, 8, 8, 8);
    }
    else {
        imageView = (ImageView)convertView;
    }

    imageView.setImageResource(images[position]);
    return imageView;
}

public int getCount() {
    return images.length;
}

public Object getItem(int position) {
    return null;
}

public long getItemId(int position) {
    return 0;
}
}
```

Запустите проект на выполнение. Нажав маркер, можно развернуть виджет на весь экран. При выделении иконки в SlidingDrawer в текстовом поле сверху экрана будет отображаться идентификатор выбранного элемента (рис. 17.14).

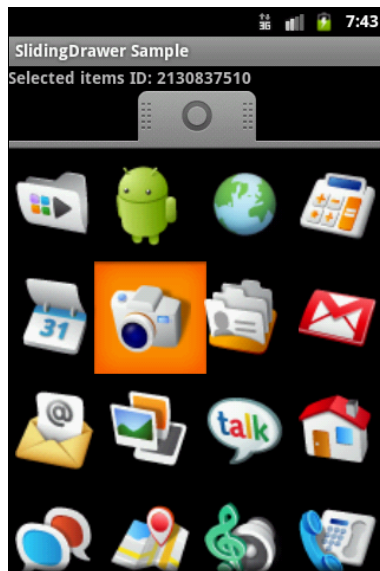


Рис. 17.14. Приложение с виджетом SlidingDrawer

Выпадающий список

Виджет Spinner — это аналог ComboBox (выпадающий список) для Android. При нажатии кнопки на элементе или центральной кнопки на D-клавиатуре устройства появляется список опций с переключателями, который, в отличие от элемента ListView, не занимает весь экран.

Основное событие элемента Spinner, которое реализуют в программе, использующей Spinner, — это событие выбора пункта списка. Для этого в коде программы необходимо реализовать методы обратного вызова `onItemSelected()` и `onNothingSelected()`, которые объявлены в интерфейсе `AdapterView.OnItemSelectedListener`.

Общая реализация создания и подключения адаптера данных для выпадающего списка аналогична реализации для ListView, рассмотренной ранее в этой главе, за исключением нескольких моментов. При создании адаптера для выпадающего списка мы используем компоновку `android.R.layout.simple_spinner_item`:

```
String[] data = {...};
ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
    this, android.R.layout.simple_spinner_item, data);
```

Затем мы должны дополнительно задать компоновку для выпадающего списка с помощью вызова метода `setDropDownViewResource()`:

```
arrayAdapter.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item);
```

Для примера приложения с использованием выпадающего списка создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — SpinnerApp;
- ☐ **Application name** — Spinner Sample;

- ❑ **Package name** — `com.samples.ui.spinner`;
- ❑ **Create Activity** — `SpinnerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_Spinner`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 17.28.

Листинг 17.28. Файл компоновки `main.xml`

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

    <TextView
        android:id="@+id/TextView01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    <Spinner
        android:id="@+id/Spinner01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="true"/>

</LinearLayout>
```

Для заполнения списка данными используйте все тот же массив строк с именами. Полный код класса `SpinnerActivity`, представляющего окно приложения, показан в листинге 17.29.

Листинг 17.29. Файл класса приложения `SpinnerActivity.java`

```
package com.samples.ui.spinner;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;

public class SpinnerActivity extends Activity
    implements AdapterView.OnItemClickListener {

    private TextView mLabel;

    // Массив данных для списка
    private final String[] contacts = {
        "Jacob Anderson", "Emily Duncan", "Michael Fuller",
```

```

        "Emma Greenman", "Joshua Harrison", "Madison Johnson",
        "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
        "Michael Honeyman", "Isabella Jackson", "William Patterson",
        "Joseph Godwin", "Samantha Bush", "Christopher Gateman");

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

    mLabel = (TextView) findViewById(R.id.TextView01);

    final Spinner spin = (Spinner) findViewById(R.id.Spinner01);
    spin.setOnItemSelectedListener(this);

    // Создаем адаптер данных
    ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(
        this, android.R.layout.simple_spinner_item, contacts);
    // Задаем компоновку для списка
    arrayAdapter.setDropDownViewResource(
        android.R.layout.simple_spinner_dropdown_item);
    spin.setAdapter(arrayAdapter);
}

public void onItemSelected(
    AdapterView<?> parent, View v, int position, long id) {
    mLabel.setText(contacts[position]);
}

public void onNothingSelected(AdapterView<?> parent) {
    mLabel.setText("");
}
}

```

Запустите проект на выполнение. Внешний вид приложения с виджетом Spinner и открытым выпадающим списком представлен на рис. 17.15.

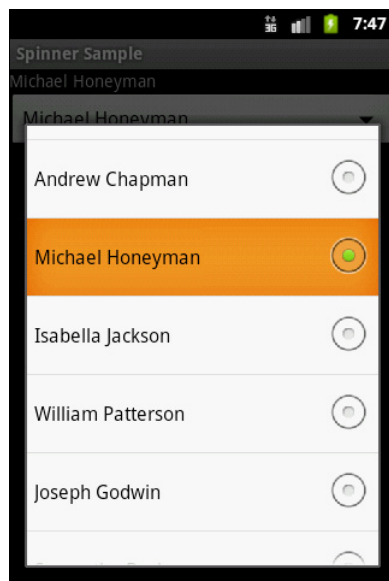


Рис. 17.15. Приложение с виджетом Spinner

Текстовые поля с автозаполнением

Текстовые поля с автозаполнением — это выпадающие списки, которые при наборе в них текста выдают подходящее слово (или набор из нескольких слов). Текстовые поля с автозаполнением довольно популярный элемент для построения пользовательского интерфейса в Android. Они представлены двумя классами:

- ❑ `AutoCompleteTextView`;
- ❑ `MultiAutoCompleteTextView`.

Эти классы наследуют все методы для работы с текстом от класса `TextView` и редактирования текста от класса `EditText`. Иерархия классов текстовых полей с автозаполнением представлена на рис. 17.16.

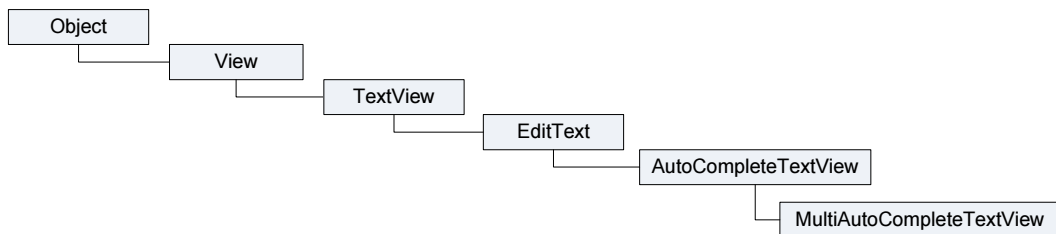


Рис. 17.16. Иерархия классов текстовых полей с автозаполнением

При использовании текстового поля с автозаполнением к нему необходимо подключать пользовательский словарь, из которого виджет будет извлекать подходящие слова или словосочетания. Таким словарем может быть любая структура данных — массив, список, но чаще всего используют для хранения словаря внешний файл (текстовый или XML) или базу данных SQLite.

AutoCompleteTextView

Виджет `AutoCompleteTextView` — это текстовое поле с автозаполнением и возможностью редактирования вводимого текста. Такие виджеты очень полезны в мобильных приложениях, когда вы хотите ускорить процесс ввода текста в приложении.

Поскольку `AutoCompleteTextView` является подклассом `EditText`, он позволяет использовать все возможности форматирования и редактирования текста. Кроме того, у `AutoCompleteTextView` есть свойство `android:completionThreshold` для указания минимального числа символов, которое должен ввести пользователь прежде, чем начинает работать функция автозаполнения списка. Для связывания с данными виджету `AutoCompleteTextView` необходим адаптер, содержащий список значений кандидата через `setAdapter()`.

Для практического примера приложения с элементом `AutoCompleteTextView` создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ❑ **Project name** — `AutoCompleteTextViewApp`;
- ❑ **Application name** — `AutoComplete Sample`;

- ❑ **Package name** — `com.samples.ui.autocompletetextview`;
- ❑ **Create Activity** — `AutoCompleteTextViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_AutoCompleteTextView`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 17.30.

Листинг 17.30. Файл компоновки `main.xml`

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/text"/>

    <AutoCompleteTextView
        android:id="@+id/auto_complete"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:completionThreshold="3"/>

</LinearLayout>
```

Визуальная компоновка адаптера данных для текстового поля с автозаполнением определяется значением `android.R.layout.simple_dropdown_item_1line`. Реализация класса окна приложения с элементом `AutoCompleteTextView` представлена в листинге 17.31. Набором данных для виджета в программе является наш массив строк с именами и фамилиями людей, которые связаны через адаптер с `AutoCompleteTextView`.

Листинг 17.31. Файл класса окна приложения `AutoCompleteTextViewActivity.java`

```
package com.samples.ui.autocompletetextview;

import android.app.Activity;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.widget.AdapterView;
import android.widget.AutoCompleteTextView;
import android.widget.TextView;

public class AutoCompleteTextViewActivity extends Activity
    implements TextWatcher {
    private TextView mText;
    private AutoCompleteTextView mAutoComplete;
```

```

final String[] contacts = {
    "Jacob Anderson", "Emily Duncan", "Michael Fuller",
    "Emma Greenman", "Joshua Harrison", "Madison Johnson",
    "Matthew Cotman", "Olivia Lawson", "Andrew Chapman",
    "Michael Honeyman", "Isabella Jackson", "William Patterson",
    "Joseph Godwin", "Samantha Bush", "Christopher Gateman"};

@Override
public void onCreate(Bundle icle) {
    super.onCreate(icle);
    setContentView(R.layout.main);

    mText = (TextView)findViewById(R.id.text);
    mAutoComplete=(AutoCompleteTextView)findViewById(
        R.id.auto_complete);
    mAutoComplete.addTextChangedListener(this);

    mAutoComplete.setAdapter(new ArrayAdapter<String>(this,
        android.R.layout.simple_dropdown_item_1line, contacts));
}

public void onTextChanged(
    CharSequence s, int start, int before, int count) {
    mText.setText(mAutoComplete.getText());
}

public void beforeTextChanged(
    CharSequence s, int start, int count, int after) {
}

public void afterTextChanged(Editable s) { }
}

```

Запустите проект на выполнение. При наборе текста в окне виджета будут отображаться варианты автозаполнения, которые будут выбираться из массива, созданного нами в классе `AutoCompleteTextViewActivity` (рис. 17.17).

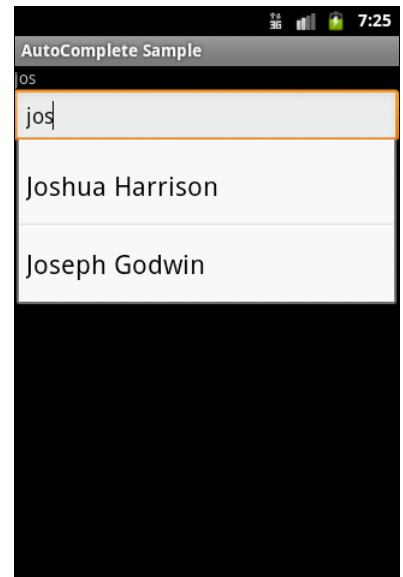


Рис. 17.17. Пример виджета `AutoCompleteTextView`

MultiAutoCompleteTextView

Виджет `MultiAutoCompleteTextView` — это текстовое поле с автозаполнением и возможностью редактирования текста, расширяющее функциональность виджета `AutoCompleteTextView` тем, что `MultiAutoCompleteTextView` может показывать автозаполнение для каждой из подстрок текста, разделенных знаком пунктуации. Разделитель задается явно вызовом метода `setTokenizer()`:

```
MultiAutoCompleteTextView textView =  
    (MultiAutoCompleteTextView) findViewById(  
        R.id.MultiAutoCompleteTextView01);  
textView.setAdapter(adapter);  
// Устанавливаем разделитель для подстрок текста  
textView.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
```

Покажем на примере работу `MultiAutoCompleteTextView`. Создайте новый проект и в диалоге **Create New Project** введите следующие значения:

- ☐ **Project name** — `MultiAutoCompleteTextViewApp`;
- ☐ **Application name** — `MultiAutoCompleteTextView Sample`;
- ☐ **Package name** — `com.samples.ui.multiautocompletetextView`;
- ☐ **Create Activity** — `MultiAutoCompleteTextViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch17_MultiAutoCompleteTextView`.

Откройте файл компоновки и создайте структуру компоновки подобно листингу 17.32.

Листинг 17.32. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent">  
  
    <MultiAutoCompleteTextView  
        android:id="@+id/MultiAutoCompleteTextView01"  
        android:layout_height="wrap_content"  
        android:layout_width="fill_parent"/>  
  
</LinearLayout>
```

Для того чтобы можно было различать подстроки в строке данных, необходимо вызвать метод `setTokenizer()`:

```
textView.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
```

Инициализация и привязка к данным для `MultiAutoCompleteTextView` в основном похожа на работу с `AutoCompleteTextView` из листинга 17.30. Претерпел изменения только набор текстовых данных, в каждую строку которых была добавлена запятая для разделения на подстроки (листинг 17.33).

Листинг 17.33. Файл класса окна приложения `MultiAutoCompleteTextViewActivity.java`

```
package com.samples.ui.autocompletetextview;

import android.app.Activity;
import android.os.Bundle;
import android.widget.MultiAutoCompleteTextView;
import android.widget.AdapterView;

public class MultiAutoCompleteTextViewActivity extends Activity {

    // Массив данных для отображения списка
    private final String[] contacts = {
        "Anderson, Jacob", "Duncan, Emily", "Fuller, Michael",
        "Greenman, Emma", "Harrison, Joshua", "Johnson, Madison",
        "Cotman, Matthew", "Lawson, Olivia", "Chapman, Andrew",
        "Honeyman, Michael", "Jackson, Isabella", "Patterson, William",
        "Godwin, Joseph", "Bush, Samantha", "Gateman, Christopher"};

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

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            this, android.R.layout.simple_dropdown_item_1line, contacts);

        MultiAutoCompleteTextView textView =
            (MultiAutoCompleteTextView) findViewById(
                R.id.MultiAutoCompleteTextView01);

        textView.setAdapter(adapter);
        textView.setTokenizer(
            new MultiAutoCompleteTextView.CommaTokenizer());
    }
}
```

Запустите проект на выполнение. Получившееся приложение представлено на рис. 17.18.

На рисунке видно, как виджет при вводе текста пользователем отображает варианты автозаполнения с учетом разбиения на подстроки элементов списка.

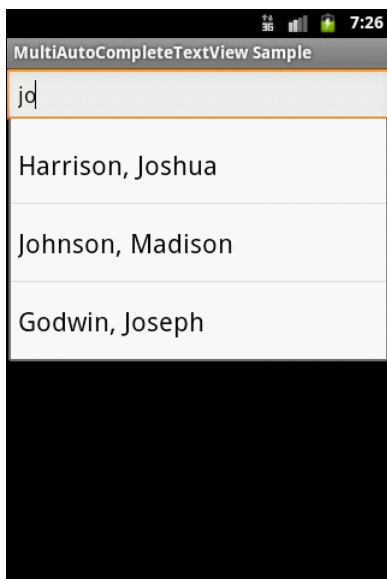


Рис. 17.18. Использование виджета `MultiAutoCompleteTextView`

Резюме

В этой главе мы рассмотрели использование виджетов для отображения данных, представленных в виде списков, причем эти данные могут содержать как текстовую, так и графическую информацию.

Полученные в этой главе навыки работы с адаптерами данных и виджетами для отображения данных нам пригодятся в следующей, где мы приступим к изучению важной темы — построению локальных хранилищ данных с использованием встроенной базы данных SQLite.



ГЛАВА 18

База данных SQLite

Платформа Android предоставляет функциональность для создания и управления локальными базами данных, которые позволяют сохранять на мобильном устройстве или планшетном ПК достаточно сложные и объемные коллекции данных. Для сохранения данных используется БД SQLite.

База данных SQLite — самая популярная БД в мире смартфонов и планшетных ПК. Эта база данных используется также и в популярных гаджетах от корпорации Apple. Широкое распространение этой базы данных на мобильных платформах объясняется эффективностью ее работы на устройствах с малой памятью и небольшой производительностью процессора.

Высокая эффективность и низкие требования к аппаратному обеспечению объясняются тем, что SQLite не является клиент-серверной базой данных, как, например, SQL Server, БД SQLite — это так называемая *встраиваемая* база данных, т. е. она не нуждается в постоянно работающем движке БД, как у SQL Server. БД SQLite предоставляет библиотеку с набором API-функций, которые может вызывать клиентское приложение.

В SQLite для хранения каждой базы данных создается отдельный файл. Доступ к этому файлу для чтения данных может быть предоставлен одновременно нескольким процессам, однако запись в файл в конкретный момент времени позволяет только одному процессу, который будет блокировать доступ для других процессов на время записи данных.

Встроенные базы данных в Android

В Android файлы баз данных SQLite сохраняются в файловой системе мобильного устройства в каталоге `/data/data/package_name/databases`, где *package_name* — имя пакета приложения, которое управляет этой базой данных. SQLite хранит всю базу данных, включая определения, таблицы, индексы и данные, в одном стандартном файле с расширением `db`.

Файлы баз данных можно увидеть, запустив **File Explorer** инструмента DDMS. Приложение, которое использует собственную базу данных, создает вложенный каталог со стандартным именем `databases`, в котором располагает файлы базы данных SQLite. Например, на рис. 18.1 в окне **File Explorer** показан файл базы данных `alarms.db` приложения `com.android.alarmclock` — это база данных стандартного будильника мобильного устройства Android.

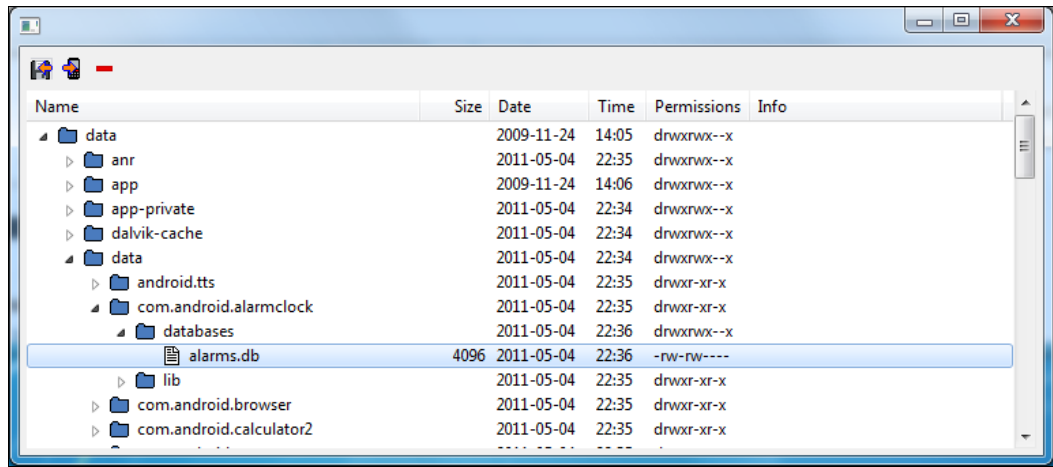


Рис. 18.1. Файл базы данных стандартного будильника мобильного устройства

Платформа Android для стандартного смартфона предоставляет несколько встроенных баз данных, доступных для всех приложений, находящихся на данном устройстве через ContentProvider (контент-провайдер) — компонент, которые предоставляют общедоступный интерфейс для работы с хранилищем данных.

Вот, например, стандартные базы данных, которые всегда присутствуют на мобильном телефоне Android, независимо от производителя и модели:

- ☐ *ContactsContract* — используется для сохранения данных пользовательских контактов в телефоне;
- ☐ *CallLog* — используется для сохранения всех вызовов: входящих и исходящих звонков, пропущенных вызовов и продолжительности звонка;
- ☐ *Browser* — используется для сохранения истории просмотренных страниц, созданных закладок и поиска;
- ☐ *Mediastore* — используется для сохранения и организации доступа к медиафайлам и коллекциям: музыки, видео и фотографиям;
- ☐ *Settings* — используется для сохранения пользовательских настроек мобильного телефона, например языка интерфейса, яркости экрана, уровня громкости звонка, конфигурации сети и т. д.;
- ☐ *UserDictionary* — используется для сохранения пользовательских словарей, которые применяются для автодополнения при вводе текста.

Существует также база данных для хранения SMS-сообщений. По какой-то причине документация по базе данных SMS отсутствует в официальной документации Android SDK, однако мы ее рассмотрим в главе 33, при изучении способов программного перехвата входящих и исходящих SMS-сообщений.

Кроме перечисленных, на мобильном телефоне также имеются различные узкоспециализированные базы данных, предназначенные для обеспечения работы различных приложений. Например, веб-браузер, помимо базы данных Browser, использует дополни-

тельную базу данных *WebviewCache* для кэширования, будильник использует собственную базу данных *Alarms*.

Каждой из перечисленных баз данных в системе Android соответствует одноименный контент-провайдер. Контент-провайдеры — это тема следующей главы, сначала нам надо изучить создание и работу приложения с собственной базой данных SQLite напрямую, без контент-провайдера.

Инструменты для работы с базами данных на Android-телефоне

Чтобы работать с базой данных SQLite — создавать таблицы и связи, просматривать и модифицировать структуру базы данных, нам понадобятся инструменты. Набор Android SDK поставляется с инструментом управления базой данных **sqlite3**. Кроме того, доступны различные инструменты для работы с SQLite сторонних разработчиков. Рассмотрим эти инструменты подробнее.

Инструмент **sqlite3**

Программа для управления базой данных **sqlite3** представляет собой инструмент командной строки, который дает возможность просматривать содержание таблиц, выполнять SQL-команды и исполнять другие полезные функции баз данных SQLite.

Работать с инструментом **sqlite3** можно следующим образом. Сначала надо запустить Android Debug Bridge (**adb**) и создать соединение с экземпляром эмулятора Android (эмулятор Android должен быть открыт и система загружена). Запустите командную строку Windows и введите в ней следующую команду:

```
adb -s emulator-5554 shell
```

Появится символ **#**, означающий приглашение к вводу следующих команд. Это значит, что **adb** нашел выполняющийся экземпляр AVD и успешно соединился с ним. Далее после символа **#** введите команду **sqlite3** и аргумент — полный путь до файла одной из баз данных SQLite, установленных на эмуляторе, например до файла базы данных SMS- и MMS-сообщений:

```
sqlite3 /data/data/com.android.providers.telephony/databases/mmssms.db
```

Должно появиться сообщение:

```
SQLite version 3.5.9
Enter ".help" for instructions
sqlite>
```

Теперь введите команду **.tables**. Будет выведен список всех таблиц, содержащихся в базе данных **mmssms.db**, как показано на рис. 18.2.

Вы можете использовать команду **.help**, чтобы посмотреть список доступных команд **sqlite3**. Более подробно с этим инструментом можно ознакомиться на сайте <http://www.sqlite.org>, где имеется большое количество документации по использованию команд **sqlite3** и примеры команд и запросов к базе данных SQLite.

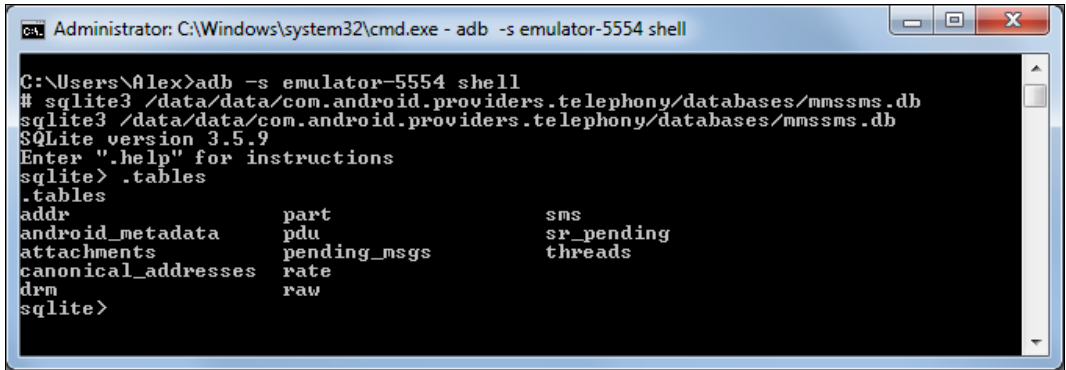


Рис. 18.2. Вывод списка таблиц базы данных SMS- и MMS-сообщений

Использование инструментов сторонних разработчиков для работы с SQLite

Сторонние разработчики также предоставляют различные графические инструменты для работы с базой данных SQLite в Android. Например, для Eclipse имеется плагин **Questoid SQLite Browser**, доступный по адресу <http://marketplace.eclipse.org/content/questoid-sqlite-browser>.

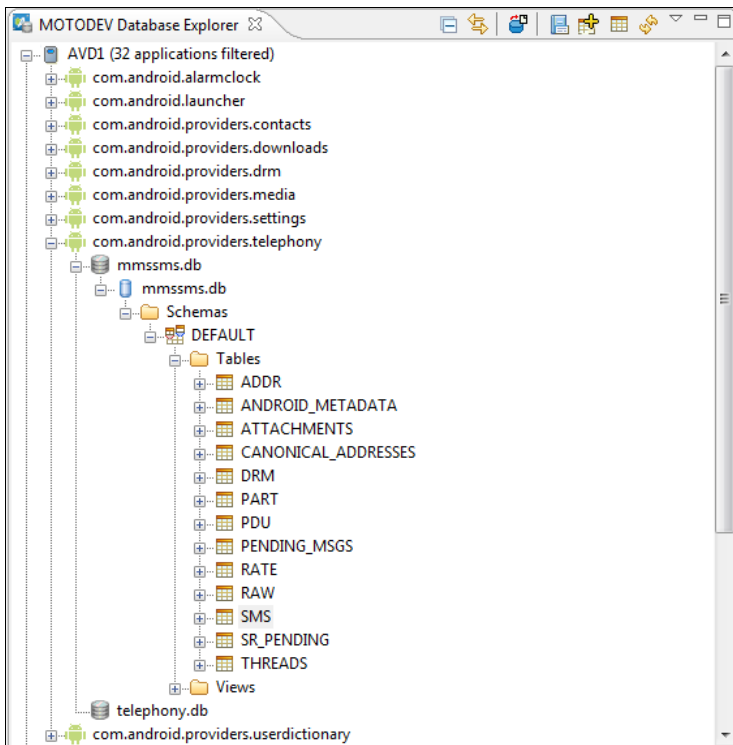
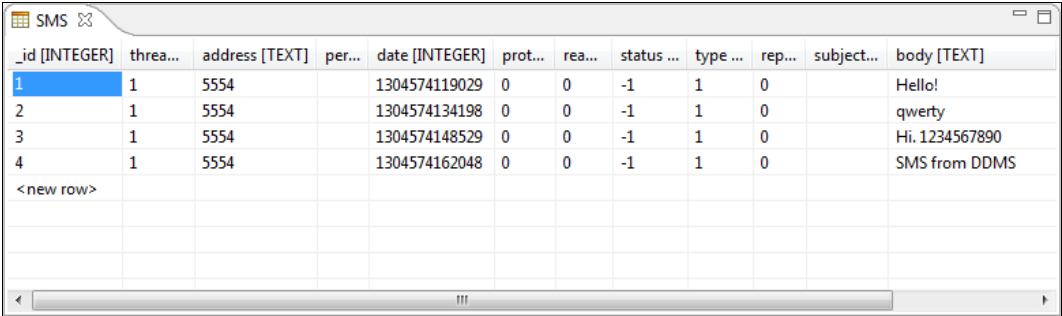


Рис. 18.3. Представление MOTODEV Database Explorer в IDE MOTODEV Studio for Android

Интересный инструмент предоставляет и Motorola — **MOTODEV Studio for Android**. Это интегрированная среда разработки на основе Eclipse (**MOTODEV Studio for Android** доступен также в виде плагина, который вы можете поставить на установленный ранее Eclipse). Познакомиться с этой средой разработки вы можете по адресу <http://developer.motorola.com/docstools/motodevstudio/>.

Среда разработки **MOTODEV Studio for Android** содержит несколько дополнительных функциональностей, которые отсутствуют в стандартном плагине ADT. В частности, в этой IDE есть хороший графический инструмент **MOTODEV Database Explorer**, с помощью которого можно работать с базами данных SQLite. Этот инструмент отображает список баз данных мобильного устройства и их структуру. Например, на рис. 18.3 показано отображение структуры базы данных SMS-сообщений в представлении **MOTODEV Database Explorer**.

MOTODEV Database Explorer также позволяет отображать и редактировать записи в таблицах. Например, на рис. 18.4 показана открытая таблица SMS с записями полученных SMS-сообщений.



_id [INTEGER]	threa...	address [TEXT]	per...	date [INTEGER]	prot...	rea...	status ...	type ...	rep...	subject...	body [TEXT]
1	1	5554		1304574119029	0	0	-1	1	0		Hello!
2	1	5554		1304574134198	0	0	-1	1	0		qwerty
3	1	5554		1304574148529	0	0	-1	1	0		Hi. 1234567890
4	1	5554		1304574162048	0	0	-1	1	0		SMS from DDMS
<new row>											

Рис. 18.4. Просмотр записей таблицы SMS базы данных SMS- и MMS-сообщений

Создание базы данных: класс SQLiteOpenHelper

Библиотеки Android обеспечивают набор классов для создания и управления базами данных SQLite.

В библиотеке Android есть класс `SQLiteOpenHelper` для создания базы данных. Класс `SQLiteOpenHelper` содержит два абстрактных метода:

- ❑ `onCreate()` — вызывается при первом создании базы данных;
- ❑ `onUpgrade()` — вызывается при модификации базы данных.

В приложении создается свой класс, наследуемый от `SQLiteOpenHelper`. В этом классе необходимо реализовать вышеперечисленные методы, описав в них логику создания и модификации базы.

В этом же классе принято объявлять открытые строковые константы для названия таблиц и полей создаваемой базы данных, которые клиенты могут использовать для определения столбцов при выполнении запросов к базе данных. Тщательно документируйте тип данных каждого столбца, т. к. клиенту требуется эта информация для обращения к данным.

Например, так можно объявить константы для таблицы `contact`:

```
public static final String TABLE_NAME = "contact";
public static final String NAME = "name";
public static final String PHONE = "phone";
```

Класс, являющийся расширением `SQLiteOpenHelper`, также неявно наследует интерфейс `BaseColumns`, в котором определена строковая константа `_ID`, представляющая имя поля для идентификаторов записей. В создаваемых таблицах базы данных поле `_ID` должно иметь тип `INTEGER PRIMARY KEY AUTOINCREMENT`. Описатель `AUTOINCREMENT` является необязательным.

В библиотеке Android для управления базой данных SQLite используется класс `SQLiteDatabase`. Этот класс имеет набор методов для создания, удаления базы данных, а также для выполнения различных операторов SQL для управления базой данных. Объект этого класса представляет нашу базу данных и передается в качестве входного параметра в методы `onCreate()` и `onUpgrade()` класса, расширяющего `SQLiteOpenHelper`.

В теле метода обратного вызова `onCreate()` необходимо реализовать логику создания таблиц и при необходимости заполнить их начальными данными, например:

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + TABLE_NAME
        + " (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        + COL_NAME + " TEXT, " + COL_PHONE + " TEXT);");
    ...
}
```

Метод `onUpdate()` вызывается при установке обновлений программы с измененной структурой таблиц. В методе обратного вызова `onUpgrade()` можно, например, реализовать запрос в базу данных на уничтожение таблицы (`DROP TABLE`), после чего вновь вызвать метод `onCreate()` для создания версии таблицы с обновленной структурой, например, так:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(db);
}
```

Давайте теперь разработаем приложение, создающее базу данных с заданной структурой и наполняющее эту базу некоторым количеством записей. Создайте в Eclipse новый проект:

- ☐ **Project name** — `DbContact`;
- ☐ **Application name** — `Database Contact creator`;
- ☐ **Package name** — `com.samples.app.dbcontact`;
- ☐ **Create Activity** — `DbCreateActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch18_DbContact`.

В файле компоновки для главного окна приложения будет единственная кнопка с надписью **Create Database** и идентификатором `button`. Код файла `main.xml` представлен в листинге 18.1.

Листинг 18.1. Файл компоновки `main.xml`

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

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Create Database"
        android:onClick="onClick"
        android:id="@+id/button"/>

</LinearLayout>
```

В предыдущих главах мы создавали учебные приложения с набором данных, представляющих собой имена и фамилии людей и их телефоны, — своего рода упрощенный аналог встроенной базы данных контактов, имеющейся на каждом мобильном телефоне. Теперь мы преобразуем этот набор данных в БД SQLite.

В проект добавим новый класс, который назовем `ContactDbHelper`. Класс `ContactDbHelper`, наследуемый от класса `SQLiteOpenHelper`, будет представлять базу данных `Contacts`. В этом классе будет создаваться база данных с единственной таблицей `people`. Таблица `people` будет содержать три столбца:

- `_id`;
- `first_name`;
- `phone`.

В методе `onCreate()` класса производится создание таблицы и заполнение ее текстовыми данными при первом запуске приложения на устройстве. Полный код класса `ContactDbHelper` приведен в листинге 18.2.

Листинг 18.2. Файл класса для создания базы данных `ContactDbHelper.java`

```
package com.samples.app.dbcontact;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.provider.BaseColumns;
```

```
public class ContactDbHelper extends SQLiteOpenHelper
    implements BaseColumns {

    public static final String DB_CONTACTS = "contacts.db";
    public static final String TABLE_NAME = "people";
    public static final String NAME = "first_name";
    public static final String PHONE = "phone";

    public ContactDbHelper(Context context) {
        super(context, DB_CONTACTS, null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + TABLE_NAME
            + " ( id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + NAME + " TEXT, " + PHONE + " TEXT);");

        ContentValues values = new ContentValues();

        values.put(NAME, "Jacob Anderson");
        values.put(PHONE, "412412411");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Emily Duncan");
        values.put(PHONE, "161863187");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Michael Fuller");
        values.put(PHONE, "896443658");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Emma Greenman");
        values.put(PHONE, "964990543");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Joshua Harrison");
        values.put(PHONE, "759285086");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Madison Johnson");
        values.put(PHONE, "950285777");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Matthew Cotman");
        values.put(PHONE, "687699999");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Olivia Lawson");
        values.put(PHONE, "161863187");
        db.insert(TABLE_NAME, NAME, values);
    }
}
```

```

        values.put(NAME, "Daniel Honeyman");
        values.put(PHONE, "876545644");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Isabella Jackson");
        values.put(PHONE, "907868756");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "William Patterson");
        values.put(PHONE, "687699693");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Joseph Godwin");
        values.put(PHONE, "965467575");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Andrew Chapman");
        values.put(PHONE, "896874556");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Samantha Bush");
        values.put(PHONE, "907865645");
        db.insert(TABLE_NAME, NAME, values);

        values.put(NAME, "Christopher Gateman");
        values.put(PHONE, "896874556");
        db.insert(TABLE_NAME, NAME, values);
    }

    @Override
    public void onUpgrade(
        SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}

```

В классе главного окна приложения `DbCreateActivity` создадим обработчик события нажатия кнопки **Create Database**. В теле этого обработчика будет формироваться наша база данных. Для создания базы данных вызывается конструктор класса `ContactDbHelper`. При формировании базы данных всегда необходимо проверять возвращаемое значение. Если результатом вызова конструктора будет `null`, значит, при создании базы данных произошла ошибка.

Код класса `DbCreateActivity` представлен в листинге 18.3.

Листинг 18.3. Файл класса главного окна приложения `DbCreateActivity.java`

```

package com.samples.app.dbcontact;

import android.app.Activity;
import android.database.sqlite.SQLiteDatabase;

```

```

import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class DbCreateActivity extends Activity
    implements View.OnClickListener{

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

    @Override
    public void onClick(View arg0) {
        SQLiteDatabase db = new ContactDbHelper(
            getApplicationContext()).getWritableDatabase();

        if (db != null) {
            Toast.makeText(getApplicationContext(),
                "DB Contacts is created", Toast.LENGTH_LONG).show();
        }
        else {
            Toast.makeText(getApplicationContext(),
                "Error create database!", Toast.LENGTH_LONG).show();
        }
    }
}

```

Выполните компиляцию и запустите приложение на эмуляторе Android. После запуска приложения нажмите кнопку **Create Database**. В случае успешного создания базы данных на экране должно отобразиться уведомление "DB Contacts is created". Внешний вид нашего приложения для создания и наполнения базы данных Contacts представлен на рис. 18.5.

Чтобы убедиться, что база данных была успешно создана, можно запустить DDMS и открыть окно **File Explorer**. В каталоге `data/data/com.samples.app.dbcontact/databases` будет находиться файл `contacts.db` (рис. 18.6).

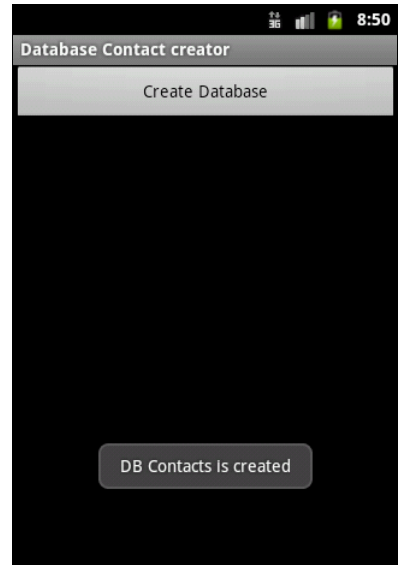


Рис. 18.5. Приложение для создания базы данных

Можно посмотреть содержимое базы данных через **sqlite3** или, если вы поставили **MOTODEV Studio for Android**, через представление **Database Explorer**. Открыв таблицу `people` в **MOTODEV Database Explorer**, можно убедиться, что все данные, которые мы определили для таблицы, были созданы (рис. 18.7).

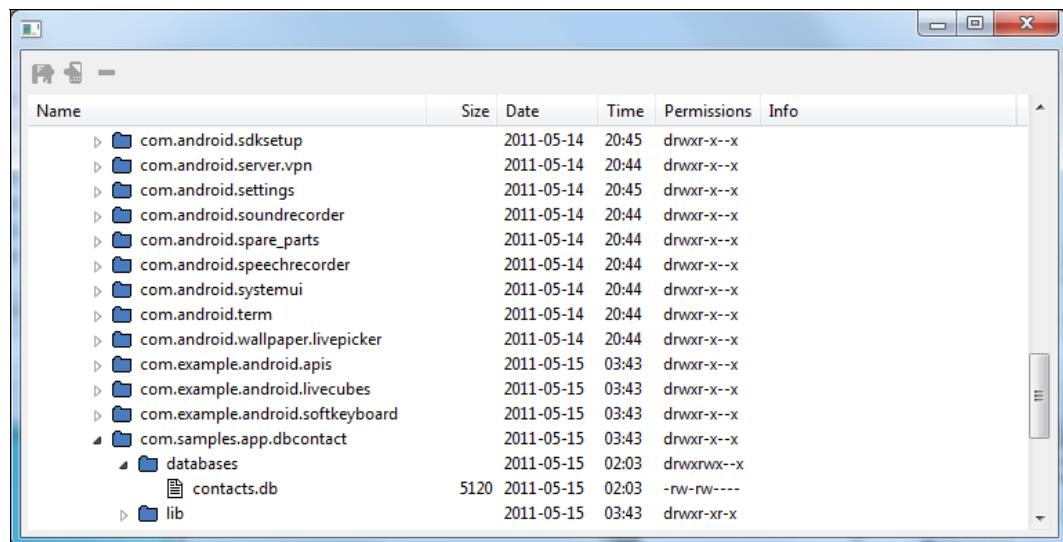


Рис. 18.6. Расположение файла базы данных в каталоге приложения на мобильном устройстве

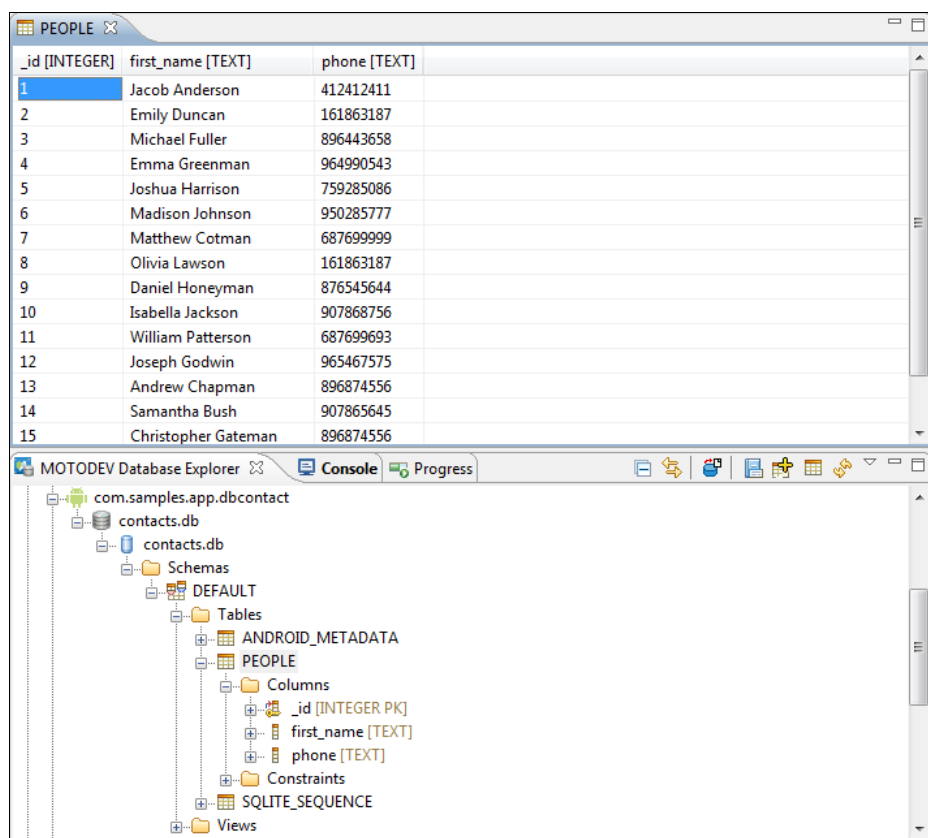


Рис. 18.7. Просмотр содержимого таблицы contacts в MOTODEV Database Explorer

Таким образом, мы создали базу данных SQLite, но нам необходимо теперь научиться пользоваться этой базой данных — получать к ней доступ из других приложений.

Резюме

В этой главе мы изучили базу данных SQLite, основные принципы ее создания и обеспечение доступа к информации, хранящейся в такой базе данных.

Если вы не собираетесь открывать данные другим приложениям, можете работать с данными непосредственно через классы, реализующие интерфейс для взаимодействия с базой данных SQLite. Однако, если нужно создать общедоступную базу данных, которой могут пользоваться остальные приложения, находящиеся на мобильном устройстве, необходимо использовать контент-провайдеры, создание и применение которых мы будем рассматривать в следующей главе.



ГЛАВА 19

Content Provider

В этой главе рассматривается организация доступа к данным с помощью компонента Content Provider (контент-провайдер). Content Provider — это один из четырех фундаментальных компонентов Android-приложения, который обеспечивает информационное наполнение приложения. Контент-провайдер нужен, если есть необходимость совместного использования данных приложениями.

Создание компонента Content Provider

Компонент Content Provider — это единственный способ совместного использования данных Android-приложениями. Если нужно предоставить доступ к своим данным для других приложений, необходимо предпринять следующие шаги:

1. Создать собственный Content Provider, как подкласс класса `ContentProvider`.
2. Объявить Content Provider в файле манифеста приложения.
3. Реализовать в клиентском приложении функциональность для доступа к Content Provider.

Все контент-провайдеры предоставляют общий интерфейс для запросов данных клиентскими приложениями — на чтение, добавление, изменение и удаление данных.

Content Provider представляют данные в виде плоской таблицы. Каждая запись в таблице обязательно включает числовое поле `_ID`, которое уникально идентифицирует запись в пределах таблицы. Идентификаторы могут использоваться для соответствия строк в связанных таблицах — например, чтобы находить номер телефона человека в одной таблице и фотографию того же человека в другой.

Расширение класса *ContentProvider*

Библиотека Android SDK предоставляет класс `ContentProvider`, который расположен в пакете `android.content`. Этот класс является абстрактным и напрямую в коде приложения не используется, необходимо создавать собственную реализацию класса, расширяющего `ContentProvider`. В классе, наследуемом от `ContentProvider`, в зависимости от того, как будет использоваться база данных, потребуется реализовать следующие методы:

- ☐ `query()` — для возвращения данных вызывающей программе;
- ☐ `insert()` — для вставки новых данных в Content Provider;

- ❑ `update()` — для обновления существующих данных в Content Provider;
- ❑ `delete()` — для удаления данных в Content Provider;
- ❑ `getType()` — для возвращения типа MIME данных в Content Provider. Этот метод реализовывать необязательно, если в нем нет необходимости.

В предыдущей главе мы создали проект `DbContact`, предназначенный для работы с базой данных. Давайте теперь добавим в этот проект класс провайдера данных, расширяющий класс `ContentProvider`. Назовем его `ContactProvider` и объявим его расширением класса `ContentProvider`. Среда Eclipse автоматически сгенерирует для нашего класса методы-заглушки, которые мы должны будем реализовать. Код созданного класса `ContactProvider` показан в листинге 19.1.

Листинг 19.1. Файл класса провайдера данных `ContactProvider.java`

```
package com.samples.app.dbcontact;

import android.content.ContentProvider;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;

public class ContactProvider extends ContentProvider {

    @Override
    public boolean onCreate() {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Cursor query(Uri arg0, String[] arg1, String arg2,
        String[] arg3, String arg4) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int delete(Uri arg0, String arg1, String[] arg2) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public Uri insert(Uri arg0, ContentValues arg1) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public int update(Uri arg0, ContentValues arg1, String arg2,
```

```
        String[] arg3) {  
        // TODO Auto-generated method stub  
        return 0;  
    }  
  
    @Override  
    public String getType(Uri arg0) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
}
```

Методы `query()`, `insert()`, `update()`, `delete()`, которые требуется реализовать в классе, производном от `ContentProvider`, предоставляют клиенту провайдера данных тонкую оболочку над одноименными методами класса `SQLiteDatabase`. Фактически `Content Provider` инкапсулирует от приложения-клиента саму базу данных. Далее мы будем постепенно заполнять класс `ContactProvider`, реализуя эти методы.

URI

Каждый контент-провайдер предоставляет открытый URI (обернутый как объект `URI`), что уникально идентифицирует его набор данных. Контент-провайдер, который управляет множественными наборами данных (множественными таблицами), должен предоставлять отдельные URI для каждого набора данных. Все URI для провайдеров начинаются со строки `"content://"`.

При создании контент-провайдера, как правило, определяют константу для URI. Android определяет константы URI для всех стандартных провайдеров, которые идут с платформой. Константа URI состоит из четырех частей, которые показаны на рис. 19.1.

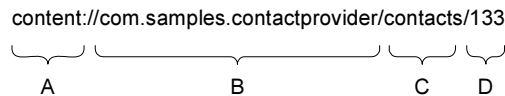


Рис. 19.1. Части константы URI

Эти части URI предоставляют следующие сведения:

- ❑ A — стандартный префикс, указывающий, что данные предоставляются Content Provider. Этот префикс никогда не изменяется;
- ❑ B — часть URI, который идентифицирует Content Provider. Для сторонних приложений он должен быть именем пакета и класса (в нижнем регистре), чтобы гарантировать свою уникальность среди других URI;
- ❑ C — путь, который Content Provider использует, чтобы определить требуемые наборы данных. Если Content Provider предоставляет только один тип данных, эта часть URI может отсутствовать. Если провайдер предоставляет данные нескольких ти-

пов, включая и подтипы, то эта часть URI будет, например, выглядеть так: `contacts/photos` или `contacts/birthday`;

- **D** — идентификатор конкретной записи. Это и есть `_ID` записи, представляющий собой строку в таблице базы данных. Если запрос не ограничивается одной записью, эта часть URI пропускается:

```
content://com.samples.app.contactprovider/contacts
```

При запросе система анализирует URI и передает запрос этому Content Provider. Сам объект `ContentProvider` не используется напрямую. Клиентские приложения вызывают Content Provider через объект класса `ContentResolver`. Получить экземпляр `ContentResolver` можно через вызов метода `getContentResolver()` в классе `Activity` или другого компонента Android-приложения:

```
ContentResolver resolver = getContentResolver();
```

Этот класс содержит большое количество методов для работы с содержимым базы данных. В клиентском приложении можно использовать методы `ContentResolver` для взаимодействия с любыми Content Provider, которые доступны и требуются для работы приложения с данными.

Константа URI используется во всех взаимодействиях с Content Provider. Каждый метод класса `ContentResolver` берет URI как первый параметр. URI идентифицирует Content Provider, к которому объект `ContentResolver` будет обращаться. После инициализации запроса система Android идентифицирует Content Provider, который является адресатом запроса и его реализацией. Система сама инициализирует все объекты `ContentProvider`, и нет необходимости делать это самостоятельно.

Если провайдер имеет связанные таблицы, необходимо определить константы URI для каждой из таблиц. Все эти URI должны иметь одинаковые полномочия (т. к. они идентифицируют Content Provider) и отличаться друг от друга только их путями.

В классе, реализующем Content Provider, также необходимо определить константу URI. Например, в нашем случае для базы данных `Contacts` и таблицы `people` вы можете добавить в класс `ContactProvider` из листинга 19.1 следующую константу:

```
static final Uri CONTENT_URI = Uri.parse(
    "content://com.samples.app.dbcontact.contactprovider/people");
```

Управление базой данных из приложения

Теперь нам надо в нашем классе `ContactProvider` определить методы для управления нашей базой данных. В теле метода `onCreate()`, который вызывается системой при создании экземпляра Content Provider, инициализируется объект `SQLiteDatabase`. Этот код мы уже использовали в классе `DbCreateActivity` из листинга 18.3 (в обработчике события кнопки `onCreate()`). Добавим этот код в метод `onCreate()`, а также создадим в классе новую переменную `db` типа `SQLiteDatabase`, т. к. объект класса `SQLiteDatabase` нам потребуется при реализации остальных методов управления базой данных. Код, который надо добавить в класс `ContactProvider`, представлен в листинге 19.2.

Листинг 19.2. Добавление в класс `ContactProvider` реализации метода `onCreate()`

```
public class ContactProvider extends ContentProvider {  
    private static final Uri CONTENT_URI = Uri.parse(  
        "content://com.samples.app.dbcontact.contactprovider/people");  
    private SQLiteDatabase db;  
  
    @Override  
    public boolean onCreate() {  
        db = new ContactDbHelper(getContext()).getWritableDatabase();  
        return (db == null) ? false : true;  
    }  
    ...  
}
```

После этого нам осталось определить стандартные методы управления базой данных для чтения, вставки, модификации и удаления данных. Это сделать довольно просто: в каждом из этих методов надо будет вызывать соответствующие методы класса `SQLiteDatabase`, поскольку созданный нами класс `ContactProvider` является оболочкой над классом `SQLiteDatabase`.

Чтение данных

Для чтения данных используют вызов метода `query()`, который объявляется так:

```
Cursor query (String table, String[] columns,  
             String selection, String[] selectionArgs,  
             String groupBy, String having, String sortOrder)
```

В метод `query()` передают семь параметров:

- ☐ `table` — имя таблицы, к которой передается запрос;
- ☐ `columns` — список имен возвращаемых полей. При передаче `null` возвращаются все столбцы;
- ☐ `selection` — параметр, формирующий выражение `WHERE` (исключая сам оператор `WHERE`). Значение `null` возвращает все строки;
- ☐ `selectionArgs` — значения аргументов фильтра для оператора `WHERE`;
- ☐ `groupBy` — параметр, формирующий выражение `GROUP BY` (исключая сам оператор `GROUP BY`). Если `GROUP BY` не нужен, передается `null`;
- ☐ `having` — параметр, формирующий выражение `HAVING` (исключая сам оператор `HAVING`). Если не нужен, передается `null`;
- ☐ `sortOrder` — параметр, формирующий выражение `ORDER BY` (исключая сам оператор `ORDER BY`). При сортировке по умолчанию передается `null`.

Объект `Cursor`, возвращаемый методом `query()`, обеспечивает доступ к набору записей результирующей выборки. Для обработки возвращаемых данных объект `Cursor`

имеет набор методов для чтения каждого типа данных — `getString()`, `getInt()` и `getFloat()`.

Реализация метода `query()` для нашего класса `ContactProvider` представлена в листинге 19.3.

Листинг 19.3. Добавление в класс `ContactProvider` реализации метода `query()`

```
@Override
public Cursor query(Uri url, String[] columns,
    String selection, String[] selectionArgs, String sort) {
    String orderBy;

    if (TextUtils.isEmpty(sort)) {
        orderBy = ContactDbHelper.NAME;
    }
    else {
        orderBy = sort;
    }

    Cursor c = db.query(ContactDbHelper.TABLE_NAME,
        projection, selection, selectionArgs, null, null, orderBy);
    c.setNotificationUri(getContext().getContentResolver(), url);

    return c;
}
```

Добавление записей

Для вставки новой записи в базу данных SQLite используется метод `insert()`. Этот метод объявлен следующим образом:

```
long insert (String table, String nullColumnHack, ContentValues values)
```

В метод `insert()` необходимо передать три параметра:

- ❑ `table` — имя таблицы, в которую будет вставлена запись;
- ❑ `nullColumnHack` — в базе данных SQLite не разрешается вставлять полностью пустую строку, и если строка, полученная от клиента Content Provider, будет пустой, то только этому столбцу явно будет назначено значение `null`;
- ❑ `values` — карта отображений (класс `Map` и его наследники), передаваемая клиентом Content Provider, которая содержит пары `ключ-значение`. Ключи в карте должны быть названиями столбцов таблицы, значения — вставляемыми данными.

Метод `insert()` возвращает идентификатор `_ID` вставленной строки или `-1` в случае ошибки. Метод `insert()` возвращает клиенту Content Provider URI вставляемой строки (с присвоенным ей идентификатором в конце).

Реализация метода `insert()` для вставки записей в нашу базу данных представлена в листинге 19.4.

Листинг 19.4. Добавление в класс `ContactProvider` реализации метода `insert()`

```
@Override
public Uri insert(Uri url, ContentValues inValues) {
    ContentValues values = new ContentValues(inValues);
    long rowId = db.insert(
        ContactDbHelper.TABLE_NAME, ContactDbHelper.NAME, values);

    if (rowId > 0) {
        Uri uri = ContentUris.withAppendedId(CONTENT_URI, rowId);
        getContext().getContentResolver().notifyChange(uri, null);

        return uri;
    }
    else {
        throw new SQLException("Failed to insert row into " + url);
    }
}
```

Обновление записей

Для обновления записей в базе данных применяют метод `update()` класса `SQLiteDatabase`:

```
int update (String table, ContentValues values,
           String whereClause, String[] whereArgs)
```

В этих методах два последних параметра формируют SQL-выражение `WHERE` аналогично рассмотренному методу `query()` для чтения данных. Эти методы возвращают в клиентское приложение количество модифицированных или удаленных строк.

Реализация метода `update()` для модификации данных в классе `ContactProvider` представлена в листинге 19.5.

Листинг 19.5. Добавление в класс `ContactProvider` реализации метода `update()`

```
@Override
public int update(Uri url, ContentValues values,
                 String where, String[] whereArgs) {
    int retVal = db.update(ContactDbHelper.TABLE_NAME, values,
                          where, whereArgs);
    getContext().getContentResolver().notifyChange(url, null);

    return retVal;
}
```

Удаление записей

Для удаления записей в базе данных используют метод `delete()`:

```
int delete (String table, String whereClause, String[] whereArgs)
```

Два последних параметра, так же как и в рассмотренных ранее методах, формируют SQL-выражение `WHERE` аналогично рассмотренному методу `query()` для чтения данных. Метод `delete()`, аналогично методу `update()`, возвращает количество модифицированных или удаленных строк.

Реализация нашего варианта метода `delete()` для класса `ContentProvider` представлена в листинге 19.6.

Листинг 19.6. Добавление в класс `ContactProvider` реализации метода `delete()`

```
@Override
public int delete(Uri url, String where, String[] whereArgs) {
    int retVal = db.delete(ContactDbHelper.TABLE_NAME, where, whereArgs);
    getContext().getContentResolver().notifyChange(url, null);

    return retVal;
}
```

Декларирование компонента Content Provider в файле манифеста приложения

Чтобы система Android могла узнать о Content Provider, который вы разработали, необходимо задекларировать его в элементе `<provider>` в файле `AndroidManifest.xml`. Content Provider, которые не объявлены в декларации, не видимы в системе Android, и обращение к ним сгенерирует исключение во время выполнения программы.

При объявлении Content Provider необходимо в атрибуте `android:name` указать имя класса, реализующего Content Provider, в атрибуте `android:authorities` — URI, по которому будут обращаться приложения для доступа к базе данных.

Код файла манифеста для приложения с добавленным элементом `<provider>` приведен в листинге 19.7.

Листинг 19.7. Файл `AndroidManifest.xml` приложения `DbContact`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.dbcontact">
    <application android:label="@string/app_name">
        <provider
            android:name=".ContactProvider"
            android:authorities="com.samples.app.dbcontact.contactprovider">
        </provider>
        <activity
            android:name=".DbCreateActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

```
        </intent-filter>
    </activity>
</application>
</manifest>
```

Атрибут `android:name` — это полное имя подкласса `ContentProvider`. Атрибут `android:authorities` — часть `authority` константы `URI`, которая идентифицирует данный `Content Provider`.

Другие атрибуты элемента `<provider>` могут устанавливать разрешения для чтения и записи данных, устанавливать значок и текст, который отображен пользователю, включать и отключать провайдер и т. д.

Запросы к Content Provider

Чтобы сделать запрос к `Content Provider` из клиентского приложения, необходимы три обязательных параметра: `URI`, который идентифицирует провайдера, имена запрашиваемых полей данных и типы данных для этих полей.

Если вы запрашиваете отдельную запись, также необходим идентификатор этой записи. Рассмотрим теперь процесс создания запросов к `Content Provider`.

Чтение возвращаемых значений

Чтобы сделать запрос к `Content Provider`, в клиентском приложении используют метод `Activity.managedQuery()`:

```
Cursor managedQuery (Uri uri, String[] projection,
                    String selection, String[] selectionArgs, String sortOrder)
```

Оба метода принимают один и тот же набор параметров и возвращают объект `Cursor`. Однако метод `managedQuery()` заставляет `Activity` управлять циклом жизни объекта `Cursor` аналогично жизненному циклу самой деятельности.

Эти методы имеют одинаковый набор параметров. Первые два параметра являются обязательными:

- `URI` провайдера — это константа `CONTENT_URI`, которая идентифицирует конкретный объект `ContentProvider` и набор данных. Существуют некоторые вспомогательные методы для формирования `URI`, в частности `ContentUris.withAppendedId()` и `Uri.withAppendedPath()`, которые добавляют в конец `URI` идентификатор записи. Это статические методы класса `ContentUris`, которые возвращают объект класса `Uri` с добавленным в конце идентификатором записи;

- имена полей данных, которые вы хотите получить из БД.

Следующие два параметра в методе — это детализация фильтра для запроса, отформатированная как SQL-предложение `WHERE` (исключая сам оператор `WHERE` непосредственно). Запрос возвращает набор записей из базы данных. Имена столбцов, их типы и заданный по умолчанию порядок сортировки данных являются специфическими для каждого `Content Provider`. Но каждый `Content Provider` имеет обязательное поле `_ID`, который содержит уникальный числовой идентификатор для каждой записи.

Значение `null` в этих параметрах возвращает все строки, если только URI сам не ограничивает запрос единственной записью. Чтобы ограничить запрос только одной конкретной записью, вы можете добавить значение `_ID` записи. Каждый провайдер может также сообщить клиентской программе о количестве возвращаемых записей через поле `_COUNT`.

Последний параметр определяет порядок сортировки записей, как в SQL-операторе `ORDER BY`. Если в параметр передать значение `null`, выборка будет отсортирована по умолчанию. Порядок сортировки по умолчанию определяют заранее, в реализации класса `SQLiteOpenHelper`. Пример фрагмента кода для получения выборки данных:

```
final String ID = "_id";
final String NAME = "name";
final String PHONE = "phone";
// Массив имен полей, которые мы хотим получить
private static final String[] mContent = new String[] {ID, NAME, PHONE};
Cursor cursor
...
Cursor cursor = managedQuery(CONTENT_URI, mContent, null, null, null);
```

Позиционирование курсора

Объект `Cursor` может использоваться для перемещений назад или вперед по выборке данных. У экземпляров типа `Cursor` есть встроенное понятие позиции, похожей на Java-интерфейс `Iterator`. Чтобы получить из выборки требуемые записи, можно использовать несколько методов для позиционирования курсора:

- ☐ `moveToFirst()` — перемещает курсор в первую запись в выборке;
- ☐ `moveToLast()` — перемещает курсор в последнюю запись в выборке;
- ☐ `moveToNext()` — перемещает курсор в следующую запись и одновременно определяет, существует ли эта запись. Метод `moveToNext()` возвращает `true`, если курсор указывает на другую строку после перемещения, и `false`, если текущая запись была последней в выборке;
- ☐ `moveToPrevious()` — перемещает курсор в предыдущую запись;
- ☐ `moveToPosition()` — перемещает курсор в указанную позицию;
- ☐ `getPosition()` — возвращает текущий индекс позиции курсора.

Кроме вышеперечисленных методов у курсора есть еще и набор условных методов для определения позиции курсора в выборке:

- ☐ `isFirst()` — курсор указывает на первую запись;
- ☐ `isLast()` — курсор указывает на последнюю запись;
- ☐ `isBeforeFirst()` — курсор установлен *перед* первой записью;
- ☐ `isAfterLast()` — курсор установлен *после* последней записи.

Эти методы могут использоваться в программном коде для проверки местонахождения позиции курсора.

Добавление записей

Чтобы добавить новую запись в Content Provider, сначала надо создать объект Map — карту с парами ключ-значение в объекте ContentValues, где каждый ключ соответствует имени столбца в Content Provider, а значение — конкретным данным для новой записи в этом столбце. После этого вызывается метод ContentResolver.insert(), которому надо передать URI провайдера и созданный объект ContentValues. Этот метод возвращает полный URI новой записи — т. е. URI Content Provider с добавленным в конце строки идентификатором для новой записи.

Вы можете использовать возвращаемый URI, чтобы сделать новый запрос и получить уже обновленную выборку данных. Например, код для добавления новой записи для нашей базы данных может выглядеть так:

```
// Создаем объект ContentValues с двумя столбцами для NAME и PHONE
ContentValues values = new ContentValues (2);

// Формируем запись
values.put (NAME, textName.getText().toString());
values.put (PHONE, textPhone.getText().toString());

// Вставляем новую запись в базу данных
getContentResolver().insert(CONTENT_URI, values);

// Обновляем выборку
cursor.requery();
```

Изменение записи

Содержащуюся в записи информацию можно изменить, используя вызов метода update(). Например, так:

```
// Создаем объект ContentValues с двумя столбцами для NAME и PHONE
ContentValues values = new ContentValues (2);

// Формируем запись
values.put (NAME, textName.getText().toString());
values.put (PHONE, textPhone.getText().toString());

// Передаем модифицируемую запись и ее ID в базу данных
getContentResolver().update(CONTENT_URI, values, "_ID=" + id, null);

// Обновляем выборку
mCursor.requery();
```

Удаление записей

Чтобы удалить единственную запись, вызовите метод ContentResolver.delete() с идентификатором этой записи, например, таким образом:

```
// Передаем запрос на удаление записи с нужным ID в базу данных
getContentResolver().delete(CONTENT_URI, "_ID=" + id, null);
```

```
// Обновляем выборку
cursor.requery();
```

Если требуется одновременно удалить множество записей в базе данных, необходимо вызвать метод `ContentResolver.delete()` с SQL-определением `WHERE`, в котором следует указать условие для удаления выбранных строк.

Клиентское приложение для работы с базой данных

Теперь, когда мы познакомились с особенностями подключения и работы с базой данных через контент-провайдер, разработаем полноценное клиентское приложение для работы с базой данных.

Создайте в Eclipse новый проект и в мастере **Create New Project** введите следующие значения:

- ☐ **Project name** — `DbContactClient`;
- ☐ **Application name** — `Contacts application`;
- ☐ **Package name** — `com.samples.app.dbcontactclient`;
- ☐ **Create Activity** — `ContactActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch19_DbContactClient`.

Файл компоновки `row.xml`, который нужен для формирования структуры списка в главном окне приложения, будет состоять из двух текстовых полей, как показано в листинге 19.8.

Листинг 19.8. Файл компоновки строки списка `row.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:textSize="18sp"/>

    <TextView
        android:id="@+id/phone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
android:layout_alignParentRight="true"
android:textSize="18sp"
android:paddingRight="10px"/>
```

```
</RelativeLayout>
```

Диалоговые окна для добавления и модификации контакта представляют собой нестандартные диалоги, для которых требуется создание собственной компоновки. В диалогах кроме кнопок **ОК** и **Cancel** добавлены текстовые поля для имени и телефона. Код файла компоновки для диалоговых окон представлен в листинге 19.9.

Листинг 19.9. Файл компоновки диалогового окна dialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView
            android:text="@string/field_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"/>
        <EditText
            android:id="@+id/name"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"/>
    </LinearLayout>
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    <TextView
        android:text="@string/field_phone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"/>
    <EditText
        android:id="@+id/phone"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"/>
    </LinearLayout>
</LinearLayout>
```

В файл `strings.xml` вынесены все надписи для текстовых полей и кнопок, имеющихся в приложении (это приложение нам понадобится позднее, в *главе 21* при создании локализованного приложения с русским языком интерфейса). Код файла `strings.xml` показан в листинге 19.10.

Листинг 19.10. Файл строковых ресурсов `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Contacts application</string>
    <string name="btn_ok">OK</string>
    <string name="btn_cancel">Cancel</string>
    <string name="field_name">Name:</string>
    <string name="field_phone">Phone:</string>
    <string name="title_add">Add new Contact</string>
    <string name="title_edit">Edit Contact</string>
    <string name="title_delete">Delete this Contact?</string>
    <string name="menu_add">Add</string>
    <string name="menu_edit">Edit</string>
    <string name="menu_delete">Delete</string>
    <string name="toast_notify">Please select Contact!</string>
</resources>
```

Класс `ContactActivity` предназначен для отображения содержимого и работы с нашей базой данных и имеет меню из трех пунктов — **Add**, **Edit** и **Delete** — для добавления, модификации и удаления данных соответственно. С помощью меню пользователь сможет вызывать диалоговые окна для добавления, редактирования и удаления контакта.

Код класса `ContactActivity` представлен в листинге 19.11.

Листинг 19.11. Файл класса окна приложения `ContactActivity.java`

```
package com.samples.app.dbcontactclient;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.ContentResolver;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.ListAdapter;
import android.widget.SimpleCursorAdapter;
import android.widget.Toast;
```



```
public class ContactActivity extends ListActivity {
    // Идентификаторы пунктов меню
    private static final int IDM_ADD = 101;
    private static final int IDM_EDIT = 102;
    private static final int IDM_DELETE = 103;

    // Имена полей таблицы Contact
    private static final String ID = "_id";
    private static final String NAME = "first_name";
    private static final String PHONE = "phone";

    private static final Uri CONTENT_URI = Uri.parse(
        "content://com.samples.app.dbcontact.contactprovider/people");

    private ContentResolver resolver;
    private Cursor cursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Получаем объект ContentResolver
        resolver = this.getContentResolver();

        // Массив, описывающий структуру таблицы people
        final String[] projection = new String[] {ID, NAME, PHONE};

        // Создаем курсор
        cursor = resolver.query(
            CONTENT_URI, projection, null, null, null);

        // Создаем адаптер данных и привязываем его к списку
        final ListAdapter mAdapter = new SimpleCursorAdapter(this,
            R.layout.main, cursor,
            new String[] {NAME, PHONE},
            new int[] {R.id.name, R.id.phone});
        setListAdapter(mAdapter);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_ADD, Menu.NONE, R.string.menu_add)
            .setIcon(R.drawable.ic_menu_add)
            .setAlphabeticShortcut('a');
        menu.add(Menu.NONE, IDM_EDIT, Menu.NONE, R.string.menu_edit)
            .setIcon(R.drawable.ic_menu_edit)
            .setAlphabeticShortcut('e');
```

```

        menu.add(Menu.NONE, IDM_DELETE, Menu.NONE, R.string.menu_delete)
            .setIcon(R.drawable.ic_menu_delete)
            .setAlphabeticShortcut('d');

        return(super.onCreateOptionsMenu(menu));
    }

    // Обработчик события выбора пункта меню
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        final long id = this.getSelectedItemId();

        switch (item.getItemId()) {
            case IDM_ADD: {
                CallAddContactDialog();
            }
            break;
            case IDM_EDIT:
                if (id > 0) {
                    CallEditContactDialog(id);
                }
                else {
                    Toast.makeText(this, R.string.toast_notify,
                        Toast.LENGTH_SHORT).show();
                }
                break;
            case IDM_DELETE:
                if (id > 0) {
                    CallDeleteContactDialog(id);
                }
                else {
                    Toast.makeText(this, R.string.toast_notify,
                        Toast.LENGTH_SHORT).show();
                }
                break;
        }
        return(super.onOptionsItemSelected(item));
    }

    // Создание диалогового окна добавления нового контакта
    private void CallAddContactDialog() {
        LayoutInflater inflater = LayoutInflater.from(this);
        View root = inflater.inflate(R.layout.dialog, null);

        final EditText textName = (EditText)root.findViewById(R.id.name);
        final EditText textPhone = (EditText)root.findViewById(
            R.id.phone);

        AlertDialog.Builder b = new AlertDialog.Builder(this);
        b.setView(root);
    }

```

```
b.setTitle(R.string.title_add);
b.setPositiveButton(
    R.string.btn_ok, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {
            ContentValues values = new ContentValues(2);

            values.put(NAME, textName.getText().toString());
            values.put(PHONE, textPhone.getText().toString());

            resolver.insert(CONTENT_URI, values);
            cursor.requery();
        }
    });
b.setNegativeButton(
    R.string.btn_cancel, new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int whichButton) {}
    });
b.show();
}

// Создание диалогового окна редактирования контакта
private void CallEditContactDialog(final long id) {
    LayoutInflater inflater = LayoutInflater.from(this);
    View root = inflater.inflate(R.layout.dialog, null);

    final EditText textName = (EditText)root.findViewById(R.id.name);
    final EditText textPhone = (EditText)root.findViewById(
        R.id.phone);

    cursor.moveToPosition(this.getSelectedItemPosition());
    textName.setText(cursor.getString(1));
    textPhone.setText(cursor.getString(2));

    AlertDialog.Builder b = new AlertDialog.Builder(this);
    b.setView(root);
    b.setTitle(R.string.title_edit);

    b.setPositiveButton(
        R.string.btn_ok, new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                ContentValues values = new ContentValues(2);

                values.put(NAME, textName.getText().toString());
                values.put(PHONE, textPhone.getText().toString());

                resolver.update(CONTENT_URI, values, "_ID=" + id, null);
                cursor.requery();
            }
        });
}
```

```
b.setNegativeButton(  
    R.string.btn_cancel, new DialogInterface.OnClickListener() {  
        public void onClick(DialogInterface dialog, int whichButton) {}  
    });  
  
b.show();  
}  
  
// Создание диалогового окна удаления контакта  
private void CallDeleteContactDialog(final long id) {  
    AlertDialog.Builder b = new AlertDialog.Builder(this);  
    b.setTitle(R.string.title_delete);  
    b.setMessage(cursor.getString(1) + "\nPhone: " +  
        cursor.getString(2));  
  
    b.setPositiveButton(  
        R.string.btn_ok, new DialogInterface.OnClickListener() {  
            public void onClick(DialogInterface dialog, int whichButton) {  
                resolver.delete(CONTENT_URI, "_ID=" + id, null);  
                cursor.requery();  
            }  
        });  
  
    b.setNegativeButton(  
        R.string.btn_cancel, new DialogInterface.OnClickListener() {  
            public void onClick(DialogInterface dialog, int whichButton) {}  
        });  
  
    b.show();  
}  
}
```

Скомпилируйте проект и запустите его в эмуляторе мобильного устройства. Если вы все сделали правильно, наше приложение должно соединиться с БД.

При первом запуске приложения сработает метод обратного вызова onCreate() в классе ContactDbHelper, который создаст базу данных Contacts с одной таблицей и заполнит ее текстовыми данными, как показано на рис. 19.2.



Contacts application	
Andrew Chapman	896874556
Christopher Gateman	896874556
Daniel Honeyman	876545644
Emily Duncan	161863187
Emma Greenman	964990543
Isabella Jackson	907868756
Jacob Anderson	412412411
Joseph Godwin	965467575
Joshua Harrison	759285086
Madison Johnson	950285777
Matthew Cotman	687699999
Michael Fuller	896443658
Olivia Lawson	161863187
Samantha Bush	907865645
William Patterson	687699693

Рис. 19.2. Клиентское приложение для работы с базой данных

Приложение содержит меню, при выборе одного из пунктов которого отображаются диалоговые окна для добавления, модификации и удаления контакта, представленные на рис. 19.3.

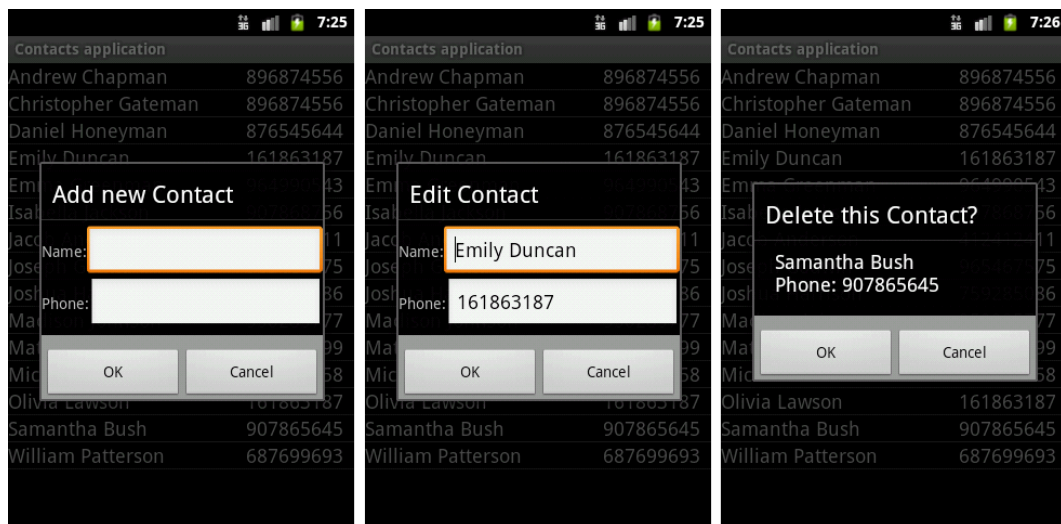


Рис. 19.3. Диалоговые окна для добавления, модификации и удаления контакта

Резюме

В этой главе мы рассмотрели последний из четырех фундаментальных компонентов Android-приложения — Content Provider, а также базу данных SQLite, применяемую в системе Android в качестве хранилища данных. Использование компонента Content Provider дает приложениям возможность взаимодействовать с хранилищами данных в системе Android. Компонент Content Provider позволяет с помощью запросов к источнику данных получать и отображать содержимое этого источника в приложении, а также управлять данными — создавать новые, модифицировать существующие и удалять записи в хранилище данных.

Далее мы переходим к главе, где будем рассматривать сохранение и управление пользовательскими настройками приложения.



ГЛАВА 20

Сохранение пользовательских настроек

Платформа Android обеспечивает механизмы для изменения и сохранения пользовательских настроек — Preferences.

Пользовательские настройки обычно используются для сохранения установок приложения, например размеров и стилей шрифта, звукового сопровождения, которые будут загружены при запуске приложения.

Пользовательские настройки в Android

В любом устройстве Android обязательно присутствует интерфейс для установки пользовательских настроек (на панели **Application Launcher** — значок **Settings**). Пример установки настроек и внешний вид окон показан на рис. 20.1.

Android предоставляет в распоряжение разработчиков библиотеку Preferences Framework, с помощью которой можно создавать индивидуальный набор настроек и

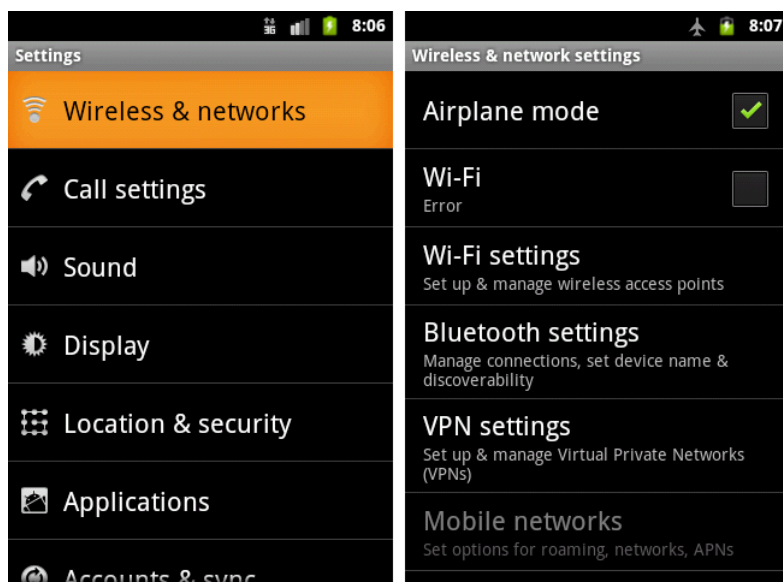


Рис. 20.1. Установка настроек для мобильного устройства

встраивать их в приложения. Preferences Framework представляет собой набор классов для разработки. Иерархия этих классов показана на рис. 20.2.

Пользовательские настройки — это отдельный экран в приложении, вызываемый из Activity. Они определяются в отдельном XML-файле. Корнем настроек в XML является элемент <PreferenceScreen>, который представляет собой контейнер и также может содержать дочерние элементы <PreferenceScreen>. Элемент <PreferenceCategory> также является контейнерным элементом и предназначен для объединения настроек в группы.

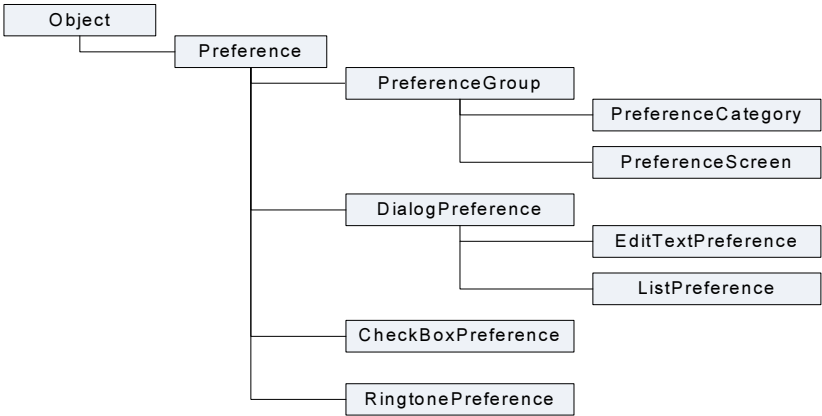


Рис. 20.2. Иерархия классов настроек

Для сохранения настроек с разным типом данных используются следующие классы:

- ☐ `CheckBoxPreference` — флажок для данных булевого типа;
- ☐ `EditTextPreference` — диалоговое окно с редактируемым текстовым полем;
- ☐ `ListPreference` — диалоговое окно со списком;
- ☐ `RingtonePreference` — диалоговое окно для установки режима звонка мобильного устройства;

Работа с этими пользовательскими настройками и использование их в приложениях будет подробно рассмотрена далее в этой главе.

Если набор настроек будет совместно использоваться разными компонентами данного приложения (например, в приложении несколько Activity), необходимо присвоить имя этому набору. Если набор настроек будет использоваться только в одном Activity, имя можно не присваивать и применять вызов метода `Activity.getPreferences()`. Нельзя использовать пользовательские настройки для обмена данными между приложениями.

Доступ к настройкам

Система Android позволяет приложениям сохранять пользовательские настройки в виде пары ключ-значение для примитивных типов данных. В пользовательских настройках можно сохранять любые данные, если они имеют тип `String` или являются примитивными типами данных (`boolean`, `int` и т. д.).

Пользовательские настройки могут быть доступны для одного Activity, а могут быть общими для всех Activity приложения. Пользовательские настройки возможно также сделать общими для всех приложений, установленных в системе.

Для получения доступа к настройкам в коде приложения внутри Activity используются следующие методы:

- ❑ `getPreferences()` — для обращения к локальным настройкам, предназначенным только для этого Activity;
- ❑ `getSharedPreferences()` — для обращения к общим настройкам приложения.

Для управления настройками, общими для всех приложений, предназначен класс `PreferenceManager`, который содержит группу методов `getDefaultSharedPreferences()` из объекта `PreferencesManager`. С их помощью можно получить общедоступные для всех приложений пользовательские настройки, предоставляемые на уровне системы.

Все эти методы возвращают экземпляр класса `SharedPreferences`, из которого можно получить значения соответствующих настроек с помощью ряда методов:

- ❑ `getBoolean(String key, boolean defValue);`
- ❑ `getFloat(String key, float defValue);`
- ❑ `getInt(String key, int defValue);`
- ❑ `getLong(String key, long defValue);`
- ❑ `getString(String key, String defValue).`

Второй параметр в методах `get...()` — это значение по умолчанию, возвращаемое при невозможности прочитать значение для данной настройки.

Например, таким образом можно получить значение для пользовательской настройки с типом `boolean`:

```
SharedPreferences prefs =
    PreferenceManager.getDefaultSharedPreferences(this);
boolean val = prefs.getBoolean(getString(R.string.pref_item), false)
```

Создание настроек для приложения мы опробуем на практике, взяв за основу проект текстового редактора для чтения-записи файла, который мы создали ранее в *разд. "Сохранение и чтение файлов с SD-карты" главы 16*, и постепенно будем совершенствовать его, добавляя различные виды настроек.

CheckBoxPreference

Настройка `CheckBoxPreference` сохраняет `boolean`-переменную в `SharedPreferences`. Эта самая простая пользовательская настройка. Для примера использования `CheckBoxPreference` и остальных настроек, которые мы будем рассматривать на протяжении этой главы, создадим новый проект и в окне **Create New Project** введем следующие значения:

- ❑ **Project name** — `Preferences`;
- ❑ **Application name** — `Preferences Sample`;

- ❑ **Package name** — `com.samples.app.preferences`;
- ❑ **Create Activity** — `EditorActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch20_Preferences`.

Файл компоновки `main.xml` будет таким же, как и в приложении с текстовым редактором из главы 16 (см. листинг 16.6). Просто скопируйте его в новый проект.

Так как у пользовательских настроек есть имена, к которым будут обращаться из файла настроек и программного кода, эти имена целесообразно хранить в файле `res/values/strings.xml`. Нашу первую пользовательскую настройку назовем `OpenMode`. Поскольку в это приложение мы будем постепенно добавлять пользовательские настройки других типов, можете сразу записать их имена в файл. Содержимое файла `strings.xml` для полного набора настроек, которые будут добавлены в этой главе, показано в листинге 20.1.

Листинг 20.1. Файл `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Preferences Sample</string>
    <string name="pr_openmode">OpenMode</string>
    <string name="pr_color">Color</string>
    <string name="pr_color_black">ColorBlack</string>
    <string name="pr_color_red">ColorRed</string>
    <string name="pr_color_green">ColorGreen</string>
    <string name="pr_color_blue">ColorBlue</string>
    <string name="pr_size">Size</string>
    <string name="pr_style">Style</string>
    <string name="pr_style_regular">StyleRegular</string>
    <string name="pr_style_bold">StyleBold</string>
    <string name="pr_style_italic">StyleItalic</string>
    <string name="pr_tone">Tone</string>
</resources>
```

Наша настройка `OpenMode` будет или сразу загружать текстовый файл в поле редактирования, если установлен флажок, или открывать пустое поле, если флажок не установлен. Код файла `preferences.xml` показан в листинге 20.2.

Листинг 20.2. Файл `preferences.xml`

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="@string/pr_openmode"
        android:title="Open file"
        android:summary="To open a file at start application"/>
</PreferenceScreen>
```

Учитывая, что пользовательские настройки объявляются в XML-файле, для их отображения необходимо использовать встроенный Activity. В классе Activity внутри метода обратного вызова `onCreate()` нужно только вызвать метод `addPreferencesFromResource()` и загрузить XML-ресурс из файла (в нашем примере — это файл `preferences.xml`), содержащий пользовательские настройки. Код класса `PreferencesActivity` для вызова представлен в листинге 20.3.

Листинг 20.3. Файл класса окна приложения `PreferencesActivity.java`

```
package com.samples.res.preferences;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class PreferencesActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Загружаем пользовательские настройки из ресурсов
        addPreferencesFromResource(R.xml.preferences);
    }
}
```

Не забудьте добавить объявление `PreferencesActivity` в файл манифеста приложения, как в листинге 20.4.

Листинг 20.4. Файл `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.preferences">
    <application android:label="@string/app_name">
        <activity
            android:name=".EditorActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".PreferencesActivity"
            android:label="@string/app_name">
        </activity>
    </application>
</manifest>
```

За основу класса главного Activity возьмем класс `EditorActivity` из главы 16 (см. листинг 16.8). В класс добавим новый пункт меню — **Settings**, который будет открывать

окно настроек. В метод обратного вызова `onOptionsItemSelected()` добавим в структуру `switch` новый элемент `case` для вызова окна настроек:

```
case IDM_PREF:
    Intent intent = new Intent();
    intent.setClass(this, PreferencesActivity.class);
    startActivity(intent);
    break;
```

Чтение установок настроек будет производиться в методе `onResume()`. Этот метод вызывается системой Android как во время запуска приложения, так и после закрытия окна настроек и возврата главного Activity на передний план. Код для получения значения настройки в методе `onResume()` должен выглядеть так:

```
SharedPreferences prefs =
    PreferenceManager.getDefaultSharedPreferences(this);
// Читаем установленное значение из CheckBoxPreference
if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
    openFile(FILENAME);
}
```

В методе `getBoolean()` второй параметр (`false`) означает значение по умолчанию для возвращаемого значения пользовательских настроек, если запрос на чтение установленного значения закончится неудачей.

Полный код класса главного Activity приложения `EditorActivity` представлен в листинге 20.5.

Листинг 20.5. Файл класса окна приложения `EditorActivity.java`

```
package com.samples.res.preferences;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.graphics.Color;
import android.graphics.Typeface;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.text.method.NumberKeyListener;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class EditorActivity extends Activity {

    static final int IDM_OPEN = 101;
    static final int IDM_SAVE = 102;
    static final int IDM_PREF = 103;
    static final int IDM_EXIT = 104;

    private final static String FILENAME = "file.txt";
    private EditText mEdit;

    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.main);

        mEdit = (EditText)findViewById(R.id.edit);
    }

    @Override
    public void onResume() {
        super.onResume();
        SharedPreferences prefs =
            PreferenceManager.getDefaultSharedPreferences(this);

        // Читаем установленное значение из CheckBoxPreference
        if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
            openFile(FILENAME);
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open")
            .setIcon(R.drawable.ic_menu_open)
            .setAlphabeticShortcut('o');
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save")
            .setIcon(R.drawable.ic_menu_save)
            .setAlphabeticShortcut('s');
        menu.add(Menu.NONE, IDM_PREF, Menu.NONE, "Settings")
            .setIcon(R.drawable.ic_menu_preferences)
            .setAlphabeticShortcut('t');
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit")
            .setIcon(R.drawable.ic_menu_exit)
            .setAlphabeticShortcut('x');
        return (super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
```

```

switch (item.getItemId()) {
    case IDM_OPEN:
        openFile(FILENAME);
        break;
    case IDM_SAVE:
        saveFile(FILENAME);
        break;
    case IDM_PREF:
        Intent intent = new Intent();
        intent.setClass(this, PreferencesActivity.class);
        startActivity(intent);
        break;
    case IDM_EXIT:
        finish();
        break;
    default:
        return false;
}
return true;
}

private void openFile(String fileName) {
    try {
        InputStream inStream = openFileInput(FILENAME);

        if (inStream != null) {
            InputStreamReader tmp =
                new InputStreamReader(inStream);
            BufferedReader reader = new BufferedReader(tmp);
            String str;
            StringBuffer buffer = new StringBuffer();

            while ((str = reader.readLine()) != null) {
                buffer.append(str + "\n");
            }

            inStream.close();
            mEdit.setText(buffer.toString());
        }
    }
    catch (Throwable t) {
        Toast.makeText(getApplicationContext(),
            "Exception: " + t.toString(), Toast.LENGTH_LONG)
            .show();
    }
}

private void saveFile(String FileName) {
    try {
        OutputStreamWriter outStream =
            new OutputStreamWriter(openFileOutput(FILENAME, 0));
    }
}

```

```
        outputStream.write(mEdit.getText().toString());
        outputStream.close();
    }
    catch (Throwable t) {
        Toast.makeText(getApplicationContext(),
            "Exception: " + t.toString(), Toast.LENGTH_LONG)
            .show();
    }
}
```

Скомпилируйте и запустите проект. В результате у вас получилась записная книжка с пока единственной опцией установки настройки (рис. 20.3).

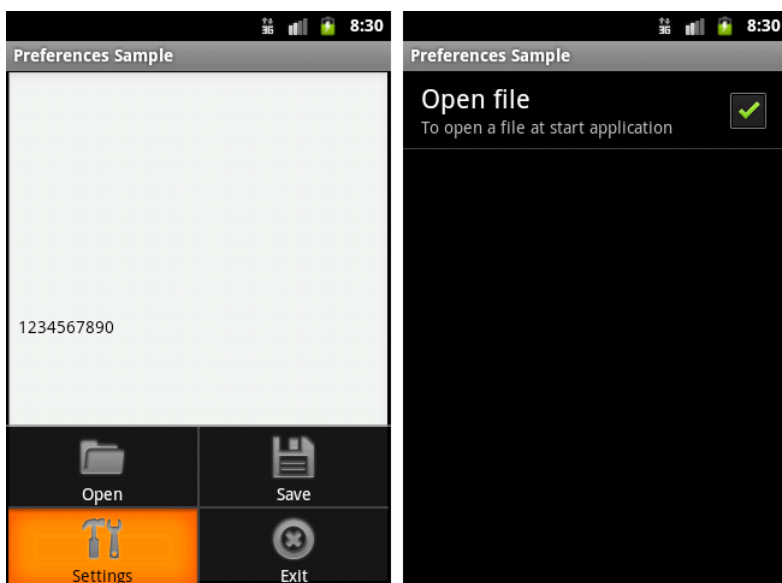


Рис. 20.3. Установка режима открытия редактора с помощью флажка `CheckBoxPreference`

EditTextPreference

Каркас настроек в Android предоставляет также пользовательские настройки с текстовым полем — `EditTextPreference`. Эти настройки позволяют фиксировать текст, вводимый пользователем в свободной форме. Чтобы продемонстрировать работу этого элемента, добавим в наше приложение дополнительные возможности — в текстовом поле пользователь будет устанавливать размер шрифта для редактора.

Файл `preferences.xml` для нашего приложения с дополнительным элементом `<EditTextPreference>` представлен в листинге 20.6.

Листинг 20.6. Файл preferences.xml

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="@string/pr_openmode"
        android:title="Open file"
        android:summary="To open a file at start application"/>
    <EditTextPreference
        android:key="@string/pr_size"
        android:title="Text Size"
        android:summary="Set text size"
        android:defaultValue="14"
        android:dialogTitle="Enter text size (from 10 to 32)"/>
</PreferenceScreen>
```

В метод `onResume()` добавим код для чтения установленного значения размера шрифта (листинг 20.7).

Листинг 20.7. Метод обратного вызова `onResume()` в классе `EditorActivity`

```
@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);

    // Читаем режим открытия файла из CheckBoxPreference
    if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
        openFile(FILENAME);
    }

    // Читаем размер текста из EditTextPreference
    float fSize = Float.parseFloat(
        prefs.getString(getString(R.string.pr_size), "20"));

    // Меняем настройки в EditText
    mEdit.setTextSize(fSize);
}
```

Скомпилируйте и запустите проект. Теперь в нашем редакторе появилась опция установки размера текста в виде диалогового окна с текстовым полем ввода (рис. 20.4).

Поскольку в текстовое поле `EditTextPreference` разрешен свободный ввод любого текста, желательно проверять пользовательский ввод. Попробуйте усовершенствовать приложение, добавив проверку правильности вводимых данных.

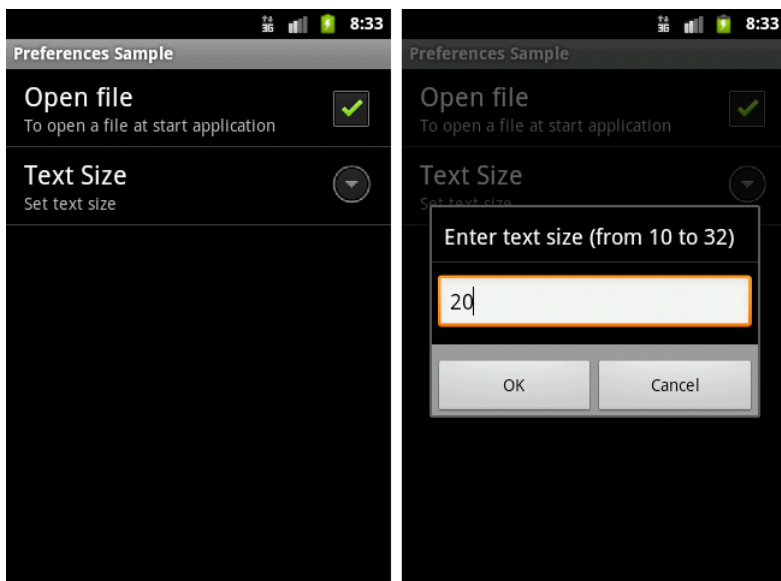


Рис. 20.4. Вызов окна `EditTextPreference` для установки размера текста

ListPreference

`ListPreference` представляет собой диалоговое окно со списком. Для его формирования требуется строковый ресурс для заголовка диалогового окна и массив строк для списка значений. Индекс выбранной строки списка определяет, какое значение сохранено как пользовательские настройки в `SharedPreferences`.

Для нашего приложения сделаем список для выбора стиля текста. В списке будет четыре опции: **Regular**, **Bold**, **Italic**, **Bold+Italic**. Для массива значений списка `ListPreference` необходимо создать в каталоге `res/values/` файл `arrays.xml`. Содержимое файла представлено в листинге 20.8.

Листинг 20.8. Файл `arrays.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="text_style">
        <item>Regular</item>
        <item>Bold</item>
        <item>Italic</item>
        <item>Bold+Italic</item>
    </string-array>
</resources>
```

В файл `preferences.xml` добавим дополнительный элемент `<ListPreference>`, в котором определим атрибуты заголовка окна, привязку к массиву значений и значение по умолчанию (листинг 20.9).

Листинг 20.9. Файл preferences.xml

```

<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="@string/pr_openmode"
        android:title="Open file"
        android:summary="To open a file at start application"/>
    <EditTextPreference
        android:key="@string/pr_size"
        android:title="Text Size"
        android:summary="Set text size"
        android:defaultValue="14"
        android:dialogTitle="Enter text size (from 10 to 32)"/>
    <ListPreference
        android:key="@string/pr_style"
        android:title="Text Style"
        android:summary="Set text style"
        android:defaultValue="1"
        android:entries="@array/text_style"
        android:entryValues="@array/text_style"
        android:dialogTitle="Choose text style"/>
</PreferenceScreen>

```

В листинге 20.10 приводится код для метода `onResume()`.

Листинг 20.10. Код метода `onResume()` в классе `EditorActivity`

```

@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);

    // Читаем открытия файла из CheckBoxPreference
    if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
        openFile(FILENAME);
    }

    // Читаем размер текста из EditTextPreference
    float fSize = Float.parseFloat(
        prefs.getString(getString(R.string.pr_size), "20"));

    // Читаем стили текста из ListPreference
    String regular = prefs.getString(getString(R.string.pr_style), "");
    int typeface = Typeface.NORMAL;

    if (regular.contains("Bold")) {
        typeface += Typeface.BOLD;
    }
}

```

```

if (regular.contains("Italic")) {
    typeface += Typeface.ITALIC;
}
// Меняем настройки в EditText
mEdit.setTextSize(fSize);
mEdit.setTypeface(null, typeface);
}

```

Теперь при запуске приложения и выборе опции **TextStyle** появляется диалог выбора пользовательских настроек для стиля текста (рис. 20.5).

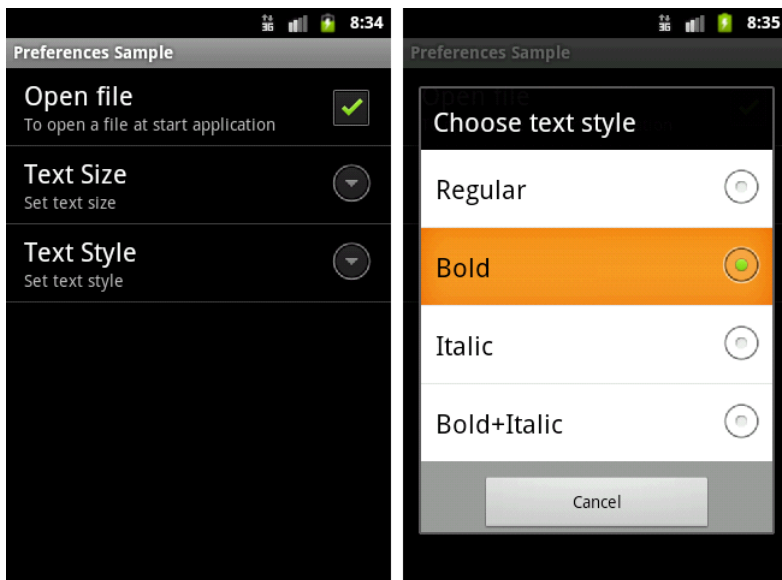


Рис. 20.5. Использование `ListPreference` для выбора стиля текста

Обратите внимание, что в диалоговом окне нет кнопки сохранения, только кнопка **Cancel**. Изменения сохраняются сразу при выборе опции списка.

RingtonePreference

`RingtonePreference` — это специализированная настройка для установки режима мелодии звонка мобильного устройства. В нашем приложении в использовании `RingtonePreference` особой необходимости нет, мы его добавим только для примера (листинг 20.11).

Листинг 20.11. Файл `preferences.xml`

```

<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <CheckBoxPreference
        android:key="@string/pr_openmode"

```

```

android:title="Open file"
android:summary="To open a file at start application"/>

<EditTextPreference
    android:key="@string/pr_size"
    android:title="Text Size"
    android:summary="Set text size"
    android:defaultValue="14"
    android:dialogTitle="Enter text size (from 10 to 32)"/>
<ListPreference
    android:key="@string/pr_style"
    android:title="Text Style"
    android:summary="Set text style"
    android:defaultValue="1"
    android:entries="@array/text_style"
    android:entryValues="@array/text_style"
    android:dialogTitle="Choose text style"/>
<RingtonePreference
    android:key="@string/pr_tone"
    android:title="Tone"
    android:showDefault="true"
    android:showSilent="true"
    android:summary="Set tone (on or off)"/>
</PreferenceScreen>

```

Настройка `<RingtonePreference>` предоставляет диалоговое окно выбора мелодии звонка со списком опций (рис. 20.6).

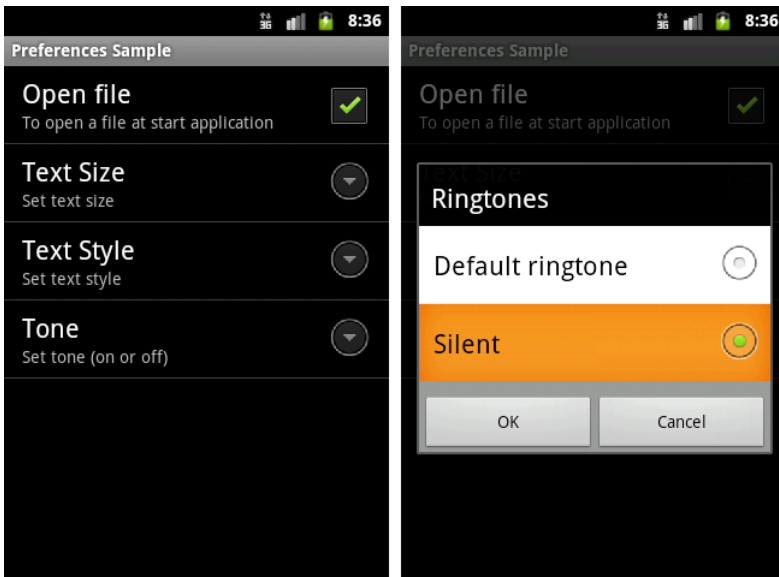


Рис. 20.6. Список `RingtonePreference` для выбора мелодии звонка мобильного устройства

Список в диалоговом окне отображает доступные на мобильном устройстве мелодии для звонка, уведомлений, тонового набора. При необходимости добавить тихий режим в элементе `<RingtonePreference>` предусмотрен атрибут `android:showSilent`: если его значение выставить в `true`, в списке мелодий появится дополнительная опция **Silent**.

PreferenceCategory

Если приложение содержит много пользовательских настроек, их расположение в одном большом списке может стать неудобным для восприятия. Preferences Framework позволяет группировать пользовательские настройки, распределяя их по категориям.

Категории добавляются через элемент `<PreferenceCategory>` в файле `preferences.xml` и используются для группировки связанных настроек. Вместо того чтобы иметь все пользовательские настройки как дочерние записи корневого элемента `<PreferenceScreen>`, можно поместить в XML-файл дополнительные элементы `<PreferenceCategory>` под корневым элементом `<PreferenceScreen>`, а затем установить пользовательские настройки в соответствующие категории.

Код файла `preferences.xml`, сгруппированный по категориям настроек, представлен в листинге 20.12.

Листинг 20.12. Файл `preferences.xml`

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Open file mode preferences">
        <CheckBoxPreference
            android:key="@string/pr_openmode"
            android:title="Open file"
            android:summary="To open a file at start application"/>
    </PreferenceCategory>
    <PreferenceCategory android:title="Text preferences">
        <EditTextPreference
            android:key="@string/pr_size"
            android:title="Text Size"
            android:summary="Set text size"
            android:defaultValue="14"
            android:dialogTitle="Enter text size (from 10 to 32)"/>
        <ListPreference
            android:key="@string/pr_style"
            android:title="Text Style"
            android:summary="Set text style"
            android:defaultValue="1"
            android:entries="@array/text_style"
            android:entryValues="@array/text_style"
            android:dialogTitle="Choose text style"/>
    </PreferenceCategory>
    <PreferenceCategory android:title="Other Preferences">
        <RingtonePreference
            android:key="@string/pr_tone"
```

```

        android:title="Tone"
        android:showDefault="true"
        android:showSilent="true"
        android:summary="Set tone (on or off)"/>
    </PreferenceCategory>
</PreferenceScreen>

```

Результатом применения элемента `<PreferenceCategory>` является список элементов настроек, сгруппированных по категориям. Визуально это добавляет разделитель с заголовком категории между группами настроек, как на рис. 20.7.

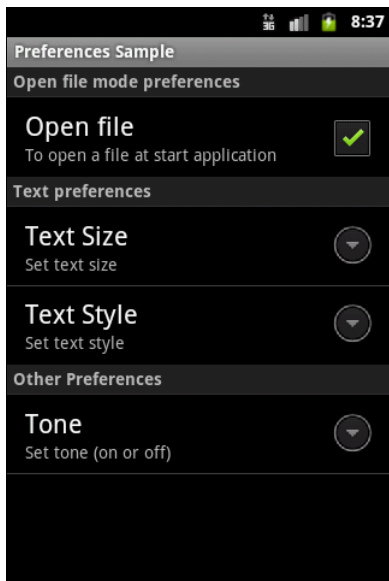


Рис. 20.7. Группировка настроек с помощью элемента `<PreferenceCategory>`

PreferenceScreen

В корневой элемент `<PreferenceScreen>` можно вложить дочерние элементы `<PreferenceScreen>`. Любые дочерние записи вложенного элемента `<PreferenceScreen>` будут отображаться на отдельном экране. Родительский экран `<PreferenceScreen>` в этом случае будет отображать в своем списке настроек вход для запуска дочернего экрана настроек.

Дополненный файл `preferences.xml` с дочерним окном настроек выбора цвета текста представлен в листинге 20.13.

Листинг 20.13. Файл `preferences.xml`

```

<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Open file mode preferences">
        <CheckBoxPreference
            android:key="@string/pr_openmode"
            android:title="Open file"
            android:summary="To open a file at start application"/>
    </PreferenceCategory>

```

```

<PreferenceCategory android:title="Text preferences">
    <PreferenceScreen
        android:key="@string/pr_color"
        android:title="Text Color"
        android:summary="Set text color">
            <CheckBoxPreference
                android:key="@string/pr_color_black"
                android:title="Black"
                android:defaultValue="true"
                android:summary="Set black color"/>
            <CheckBoxPreference
                android:key="@string/pr_color_red"
                android:title="Red"
                android:summary="Set red color"/>
            <CheckBoxPreference
                android:key="@string/pr_color_green"
                android:title="Green"
                android:summary="Set green color"/>
            <CheckBoxPreference
                android:key="@string/pr_color_blue"
                android:title="Blue"
                android:summary="Set blue color"/>
        </PreferenceScreen>
    <EditTextPreference
        android:key="@string/pr_size"
        android:title="Text Size"
        android:summary="Set text size"
        android:defaultValue="14"
        android:dialogTitle="Enter text size (from 10 to 32)"/>
    <ListPreference
        android:key="@string/pr_style"
        android:title="Text Style"
        android:summary="Set text style"
        android:defaultValue="1"
        android:entries="@array/text_style"
        android:entryValues="@array/text_style"
        android:dialogTitle="Choose text style"/>
</PreferenceCategory>
<PreferenceCategory android:title="Other Preferences">
    <RingtonePreference
        android:key="@string/pr_tone"
        android:title="Tone"
        android:showDefault="true"
        android:showSilent="true"
        android:summary="Set tone (on or off)"/>
    </PreferenceCategory>
</PreferenceScreen>

```

Для окна выбора цвета текста необходимо добавить код обработки выбора цвета в метод `onResume()` класса `EditorActivity`. Выбор нескольких цветов из группы сумми-

рует значения каждого выбранного цвета, позволяя получить дополнительные цвета текста.

Измененный метод `onResume()` класса `EditorActivity` представлен в листинге 20.14.

Листинг 20.14. Метод обратного вызова `onResume()` в классе `EditorActivity`

```
@Override
public void onResume() {
    super.onResume();
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);

    // Читаем режим открытия файла из CheckBoxPreference
    if (prefs.getBoolean(getString(R.string.pr_openmode), false)) {
        openFile(FILENAME);
    }

    // Читаем размер текста из EditTextPreference
    float fSize = Float.parseFloat(
        prefs.getString(getString(R.string.pr_size), "20"));

    // Читаем цвет текста из CheckBoxPreference
    // и суммируем значения для получения дополнительных цветов текста
    int color = Color.BLACK;
    if (prefs.getBoolean(getString(R.string.pr_color_red), false)) {
        color += Color.RED;
    }
    if (prefs.getBoolean(getString(R.string.pr_color_green), false)) {
        color += Color.GREEN;
    }
    if (prefs.getBoolean(getString(R.string.pr_color_blue), false)) {
        color += Color.BLUE;
    }
    float fSize = Float.parseFloat(
        prefs.getString(getString(R.string.pr_size), "20"));

    // Читаем стили текста из ListPreference
    String regular = prefs.getString(getString(R.string.pr_style), "");
    int typeface = Typeface.NORMAL;

    if (regular.contains("Bold")) {
        typeface += Typeface.BOLD;
    }
    if (regular.contains("Italic")) {
        typeface += Typeface.ITALIC;
    }

    // Меняем настройки в EditText
    mEdit.setTextSize(fSize);
}
```



```
mEdit.setTextColor(color);  
mEdit.setTypeface(null, typeface);  
}
```

Окончательный вид приложения со всеми созданными в этой главе пользовательскими настройками и отдельным входом **Text Color** для дочернего окна настроек показан на рис. 20.8.

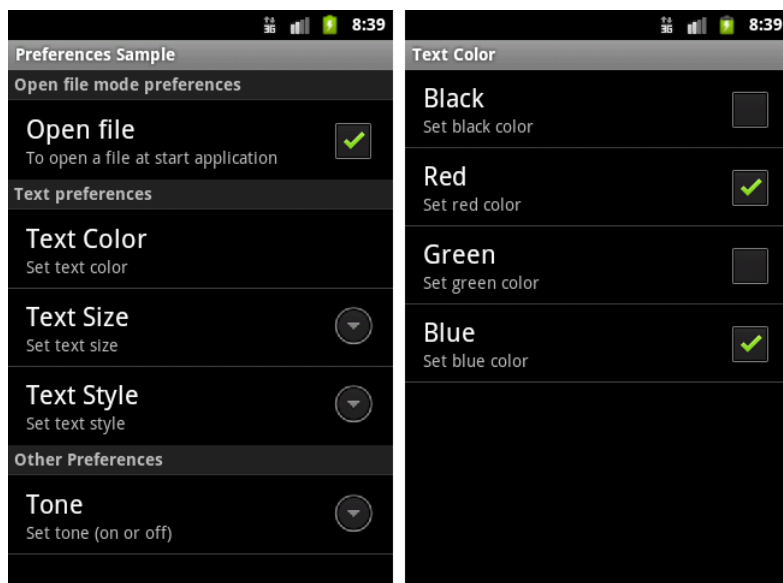


Рис. 20.8. Добавление дочернего контейнера `PreferenceScreen` для выбора цвета текста

Резюме

В этой главе мы изучили возможности, предоставляемые системой Android для сохранения пользовательских настроек приложения. Платформа Android имеет `Preferences Framework` — специальный каркас, с помощью которого в Android-приложении можно создать функциональность для сохранения и изменения пользовательских настроек данного приложения.

В следующей главе мы рассмотрим принципы создания локализованных приложений для Android.

ГЛАВА 21



Локализация приложений

Приложения для платформы Android могут работать на устройствах во многих регионах. Чтобы привлечь больше пользователей, ваше приложение должно обрабатывать текстовые, аудиофайлы, числа, валюту и графику способами, соответствующими настройкам или языкам тех регионов, где ваше приложение будет использоваться.

Ресурсы, заданные по умолчанию

Всякий раз, когда приложение запускается с настройками региона (или языка), для которого вы не создали локализованные строковые ресурсы, система Android загружает заданные по умолчанию строки из файла `res/values/strings.xml`. Если заданный по умолчанию ресурс отсутствует или в нем отсутствует требуемая строка, то приложение не запустится и сгенерирует исключение.

Чтобы предотвратить запуск приложения с ошибкой, удостоверьтесь, что файл `res/values/strings.xml` существует и определяет каждую необходимую строку. Данное требование применимо ко всем типам ресурсов, а не только к строкам: вы должны создать набор заданных по умолчанию файлов ресурса, содержащих все ресурсы, которые ваше приложение загружает, — компоновки, изображения, анимацию и т. д.

При создании локализованного приложения исследуйте в программном коде каждую ссылку на ресурсы. Удостоверьтесь, что заданный по умолчанию ресурс определен для каждого. Также удостоверьтесь, что заданный по умолчанию строковый файл содержит все необходимые строки. Локализованный строковый файл может содержать подмножество строк, но заданный по умолчанию строковый файл должен содержать их все — если какая-либо строка отсутствует в локализованном файле, загрузится версия строки, определенная в ресурсах по умолчанию.

Создание локализованных ресурсов

Хорошая практика программирования под Android заключается в использовании файлов ресурсов, чтобы отделить содержание приложения от логики работы приложения. Обычно локализуют строковые свойства элементов пользовательского интерфейса, но в принципе можно локализовать и другие компоненты приложения, например XML-компоновку для окон приложения, создав отдельную компоновку для каждого региона или языка.

Чтобы создать дополнительный ресурс для конкретного региона или языка, необходимо использовать спецификатор, который определяет комбинацию региона и языка. Пример спецификаторов приведен в табл. 21.1.

Таблица 21.1. Спецификаторы локализованных ресурсов

Локальный код	Язык / Страна	Размещение файла strings.xml
Default	English / United Kingdom	res/values/
ru-rRU	Russian / Russia	res/values-ru/
de-rDE	German / Germany	res/values-de/
ja-rJP	Japanese / Japan	res/values-ja/
fr-rFR	French / France	res/values-fr/
fr-rCA	French / Canada	res/values-fr/
en-rCA	English / Canada	res/values/
en-rUS	English / United States	res/values/

Как видно из таблицы, английский язык является заданным по умолчанию и для него нет смысла определять отдельный ресурс, создавая каталог `res/values-en/`. Давайте теперь создадим локализованное приложение, переводя на русский язык интерфейс приложения для работы с базой данных контактов из главы 18. При создании проекта мастер создает в каталоге `res/` папки по умолчанию.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch21_LocalizeDbContact`.

В каталоге `res/` создадим дополнительную папку `values-ru/` — каталог для русской локализованной версии. В этот каталог мы поместим локализованную версию файла строковых ресурсов, как показано на рис. 21.1.

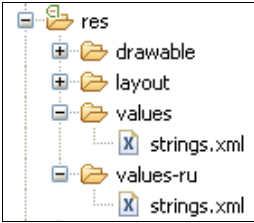


Рис. 21.1. Каталоги с локализованными строковыми ресурсами

На основе английского файла создайте файл для русской версии приложения, переводя все строки, находящиеся в нем, на русский язык, как показано в листинге 21.1. Сохраните файл в каталоге `res/values-ru/`.

Листинг 21.1. Файл строковых ресурсов strings.xml для русского языка

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Загрузка контактов из базы данных</string>
```

```
<string name="btn_ok">Сохранить</string>
<string name="btn_cancel">Отмена</string>
<string name="field_name">Имя:</string>
<string name="field_phone">Тел.:</string>
<string name="title_add">Добавить новый контакт</string>
<string name="title_edit">Изменить контакт</string>
<string name="title_delete">Удалить контакт?</string>
<string name="menu_add">Добавить</string>
<string name="menu_edit">Изменить</string>
<string name="menu_delete">Удалить</string>
<string name="toast_notify">выберите контакт!</string>
</resources>
```

Создав необходимые файлы, выполните компиляцию проекта и запустите его в эмуляторе Android. Если у вас в эмуляторе остались настройки по умолчанию, отобразится английская версия приложения.

Теперь поменяйте язык. Для этого откройте панель **Application Launcher** и выберите последовательно **Settings | Language & keyboard | Select language | Русский**, как показано на рис. 21.2.

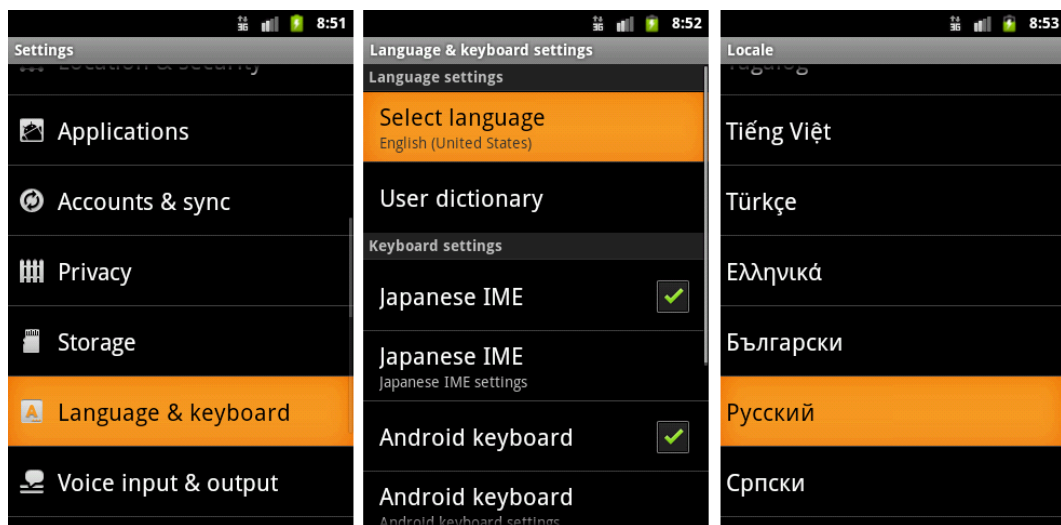


Рис. 21.2. Установка языковых настроек в эмуляторе

Снова зайдите в **Application Launcher**. Теперь наше приложение в **Application Launcher** называется **Загрузка контактов из базы данных**. Запустите его. Приложение и все его диалоговые окна будут на русском языке. Внешний вид локализованного приложения с русским языком интерфейса, запущенного в эмуляторе мобильного устройства, показан на рис. 21.3.

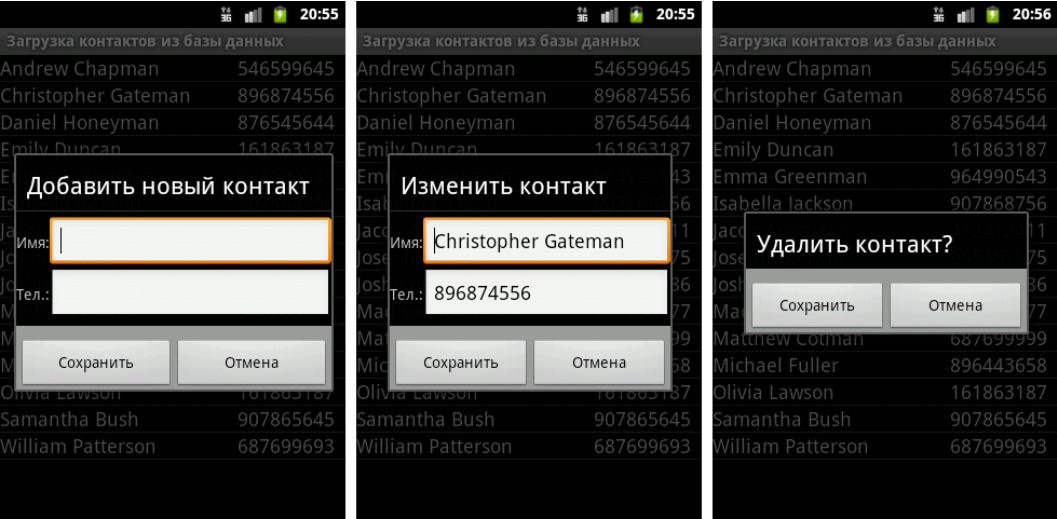


Рис. 21.3. Локализованное приложение для редактирования контактов

Резюме

В этой главе мы рассмотрели общие принципы создания приложений для платформы Android с поддержкой разных языков интерфейса и для примера произвели локализацию ранее созданного англоязычного приложения на русский язык.

В следующей главе мы приступим к изучению библиотек, предоставляемых платформой Android для работы с графикой, — рассмотрим рисование графических примитивов и загрузку графики из ресурсов и XML-документов.



ГЛАВА 22

Графика

В этой главе мы обсудим варианты применения графики в Android-приложениях. Также будут рассмотрены основы использования объектов `Drawable` для работы с графикой.

Графику в приложении можно создавать двумя способами:

- ☐ нарисовав графику в объекте `View` из компоновки;
- ☐ нарисовав графику непосредственно на канве.

Рисование графики в объекте `View` является лучшим выбором, если требуется нарисовать простую графику, которая не будет динамически изменяться в процессе работы приложения и не является реализацией сложной графической игры.

Объект *Drawable*

Android SDK предлагает двумерную графическую библиотеку рисования на формах и изображениях — `android.graphics.drawable`.

Класс `Drawable` является базовым классом для всех классов работы с графикой. Это общая абстракция для рисуемого объекта. Класс `Drawable` определяет разнообразные виды графики, включая `BitmapDrawable`, `ShapeDrawable`, `PictureDrawable`, `LayerDrawable` и др. Диаграмма графических классов Android представлена на рис. 22.1.

Есть два способа определить и инициализировать объект `Drawable`:

- ☐ использовать изображения, сохраненные в каталоге `res/drawable/`;
- ☐ использовать XML-файл, в котором будут определены свойства объекта `Drawable`.

Самый простой способ добавить графику в приложение — это создать ссылку на загрузочный модуль проектных ресурсов. При этом поддерживаются следующие типы файлов:

- ☐ PNG — наиболее предпочтительный тип;
- ☐ JPEG — приемлемый тип;
- ☐ GIF — нежелательный тип, лучше конвертировать в PNG.

Загрузка через ссылку на ресурсы предпочтительна для значков, картинок или другой графики в приложении. Этот способ загрузки графических ресурсов мы уже интенсивно использовали в создаваемых приложениях на протяжении всей книги.

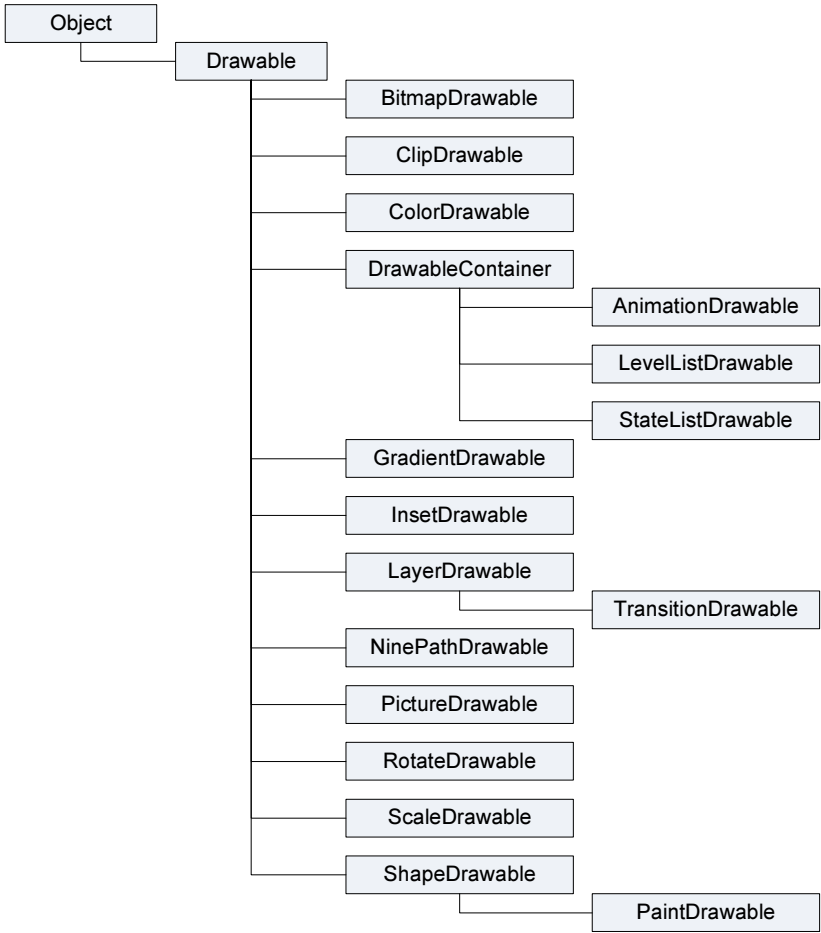


Рис. 22.1. Иерархия графических классов

Ресурсы изображений, помещенные в каталог `res/drawable/` во время компиляции проекта, могут быть автоматически оптимизированы со сжатием изображения утилитой `aapt`. Качество изображения после сжатия остается практически таким же, но при этом требуется меньший объем памяти на мобильном устройстве. Если требуется загружать растровые изображения (BMP, которые утилита `aapt` обязательно оптимизирует) без сжатия, поместите их в каталог `res/raw/`, где они не будут оптимизироваться утилитой `aapt`. Однако потребуются загрузка с использованием методов потокового ввода-вывода и последующая конвертация графики в растровый рисунок.

Создание объектов *Drawable* в коде программы

В других случаях вы можете захотеть обработать ваш ресурс изображения как объект `Drawable`. Чтобы это сделать, создайте `Drawable`, загрузив изображение из ресурсов примерно так:

```
Resources res = mContext.getResources();
Drawable myImage = res.getDrawable(R.drawable.my_image);
```


Каждый уникальный ресурс в вашем проекте может поддерживать только одно состояние независимо от того, сколько различных объектов вы можете инициализировать в программном коде для этого ресурса. Например, если вы инициализируете два объекта `Drawable` от одного ресурса изображения, а затем измените свойство, например `alpha` (прозрачность), для одного из объектов `Drawable`, другой объект также изменит это свойство.

К настоящему времени вы должны быть знакомы с принципами разработки интерфейса пользователя. Следовательно, вы понимаете силу и гибкость, свойственную определению объектов в XML. Если есть объект `Drawable`, который вы хотели бы создать и который первоначально не зависит от переменных, определенных вашим программным кодом или пользовательским взаимодействием, то определение `Drawable` в XML — наилучшее решение. Даже если вы ожидаете, что ваш объект `Drawable` изменит свои свойства в течение работы пользователя с приложением, вы должны задать начальные свойства объекта `Drawable` в XML, поскольку вы можете всегда изменять свойства после создания этого объекта в коде программы.

Как только вы определили ваш `Drawable` в XML-файле, сохраните этот файл в каталоге `res/drawable/` вашего проекта. Затем загрузите и инициализируйте объект, вызвав метод `Resources.getDrawable()` и передав ему в качестве параметра идентификатор ресурса вашего XML-файла. Любой подкласс, производный от класса `Drawable`, который поддерживает метод `inflate()`, может быть определен в XML и инициализирован в коде приложения.

В следующих разделах мы подробно рассмотрим использование объектов `Drawable` в коде программы.

Класс *TransitionDrawable*

Одна из возможностей использования объектов `Drawable` — загрузка картинок с плавными переходами. Создание переходов между изображениями с помощью класса `TransitionDrawable` лучше рассмотреть на примере. Создайте в среде Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — `Transition`;
- ☐ **Application name** — `Transition Sample`;
- ☐ **Package name** — `com.samples.res.transition`;
- ☐ **Create Activity** — `TransitionActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в электронном архиве в каталоге `Ch22_Transition`.

В файле компоновки для Activity создайте один элемент `ImageView`, как показано в листинге 22.1.

Листинг 22.1. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```

android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:gravity="center">

<ImageView
    android:id="@+id/image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/photo1"/>

</LinearLayout>

```

В каталоге `res/drawable/` проекта создадим два графических файла: `photo1.jpg` и `photo2.jpg`. Их можно взять из папки `Resources/` архива книги или в вашем каталоге `Android SDK — графика, используемая в примерах`, взята из каталога `android-sdk-windows/platforms/android-2/samples/ApiDemos/res/drawable/`.

В каталоге `res/drawable/` проекта также создадим файл `transition.xml`, в котором определим переход. Код файла `transition.xml` представлен в листинге 22.2.

Листинг 22.2. Файл `transition.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/photo1"></item>
    <item android:drawable="@drawable/photo2"></item>
</transition>

```

В классе окна приложения `LoadImageActivity` создадим объект `TransitionDrawable` и установим его как содержание `ImageView`:

```

Resources res = this.getResources();
mTransition = (TransitionDrawable) res.getDrawable(R.drawable.transition);

```

Также в классе `LoadImageActivity` создадим обработчик `onClick()`, при вызове которого будет происходить смена картинок с плавным переходом в течение 1 секунды:

```

transition.startTransition(1000);

```

Полный код класса `TransitionActivity` представлен в листинге 22.3.

Листинг 22.3. Файл класса окна приложения `TransitionActivity.java`

```

package com.samples.res.transition;

import android.app.Activity;
import android.content.res.Resources;
import android.graphics.drawable.TransitionDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

```

```
public class TransitionActivity extends Activity
    implements View.OnClickListener {

    private ImageView image;
    private TransitionDrawable trans;

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

        image = (ImageView) findViewById(R.id.image);

        Resources res = this.getResources();
        trans = (TransitionDrawable) res.getDrawable(R.drawable.transition);
    }

    @Override
    public void onClick(View v) {
        // Переключаем картинки с эффектом плавного
        // перехода в течение 1 секунды
        image.setImageDrawable(trans);
        trans.startTransition(1000);
    }
}
```

Выполните компиляцию проекта и запустите проект на выполнение. При касании экрана (нажатии правой кнопки мыши в области экрана эмулятора) будет происходить плавное переключение фотографий (рис. 22.2).

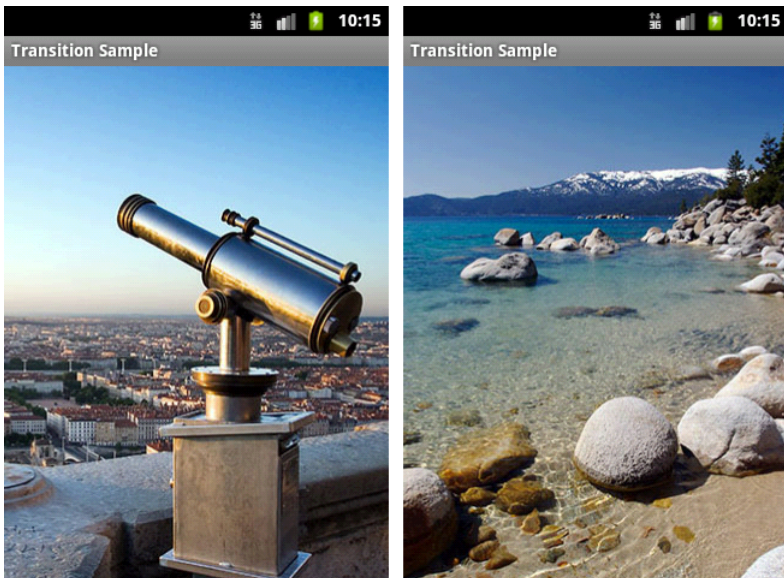


Рис. 22.2. Переходы между изображениями

Класс *ShapeDrawable*

Если вы хотите динамически рисовать различные двумерные фигуры, класс *ShapeDrawable* вполне удовлетворит ваши потребности. С объектами *ShapeDrawable* вы можете программно создавать любые примитивные формы и их стили.

Для создания графических примитивов в библиотеке Android есть набор классов, производных от базового класса *Shape*:

- ☐ *PathShape*;
- ☐ *RectShape*;
- ☐ *ArcShape*;
- ☐ *OvalShape*;
- ☐ *RoundRectShape*.

Иерархия классов графических примитивов представлена на рис. 22.3.

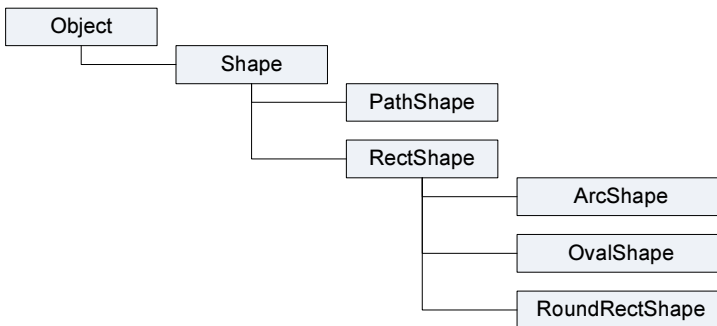


Рис. 22.3. Иерархия классов графических примитивов

Класс *ShapeDrawable* — расширение класса *Drawable*, вы можете использовать его так же, как и любой другой объект *Drawable*. В конструкторе *ShapeDrawable* рисуемый графический примитив определяется как *RectShape*, т. е. прямоугольник. Также для объекта *ShapeDrawable* необходимо установить цвет и границы фигуры. Если вы не установите границы, то графический примитив не будет рисоваться. Если вы не установите цвет, фигура будет черной — значение по умолчанию.

Класс *RectShape* также можно использовать для рисования горизонтальных или вертикальных линий. Для этого надо задать высоту (или ширину) прямоугольника в 1—2 пиксела, например:

```

ShapeDrawable d = new ShapeDrawable(new RectShape());
d.setIntrinsicHeight(2);
d.setIntrinsicWidth(150);
d.getPaint().setColor(Color.MAGENTA);
  
```

Аналогично прямоугольнику прорисовывается объект *OvalShape* (эллипс):

```

ShapeDrawable d = new ShapeDrawable(new OvalShape());
d.setIntrinsicHeight(100);
d.setIntrinsicWidth(150);
d.getPaint().setColor(Color.RED);
  
```

Прорисовка прямоугольника с закругленными сторонами (`RoundRect`) несколько сложнее. Конструктор класса `RoundRect` для рисования прямоугольника с закругленными сторонами принимает несколько параметров:

```
RoundRectShape(float[] outerRadii, RectF inset, float[] innerRadii)
```

Параметры конструктора `RoundRectShape`:

- ❑ `outerRadii` — массив из восьми значений радиуса закругленных сторон внешнего прямоугольника. Первые два значения для верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внешнем прямоугольнике закруглений не будет, передается `null`;
- ❑ `inset` — объект класса `RectF`, который определяет расстояние от внутреннего прямоугольника до каждой стороны внешнего прямоугольника. Конструктор `RectF` принимает четыре параметра: пары `X` и `Y` координат левого верхнего и правого нижнего углов внутреннего прямоугольника. Если внутреннего прямоугольника нет, передается `null`;
- ❑ `innerRadii` — массив из восьми значений радиуса закруглений углов для внутреннего прямоугольника. Первые два значения — координаты верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внутреннем прямоугольнике закруглений не будет, передается `null`.

Например, создание прямоугольника с закругленными сторонами может выглядеть следующим образом:

```
float[] outR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };
RectF    rectF = new RectF(8, 8, 8, 8);
float[] inR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };
```

```
ShapeDrawable d = new ShapeDrawable(
    new RoundRectShape(outR, rectF, inR));
d.setIntrinsicHeight(100);
d.setIntrinsicWidth(150);
d.getPaint().setColor(Color.WHITE);
```

Класс `Path` формирует множественный контур геометрических путей, состоящих из прямых линейных сегментов, квадратичных и кубических кривых. Для установки точек и перемещения линий (или кривых) используются открытые методы `moveTo()` и `lineTo()`. Например, так с помощью класса `Path` можно нарисовать пятиконечную звезду:

```
Path p = new Path();
p.moveTo(50, 0);
p.lineTo(25, 100);
p.lineTo(100, 50);
p.lineTo(0, 50);
p.lineTo(75, 100);
p.lineTo(50, 0);
```

```
ShapeDrawable d = new ShapeDrawable(new PathShape(p, 100, 100));
d.setIntrinsicHeight(100);
d.setIntrinsicWidth(100);
```

```
d.getPaint().setColor(Color.YELLOW);
d.getPaint().setStyle(Paint.Style.STROKE);
```

Класс `ArcShape` создает графический примитив в форме дуги. Конструктор класса принимает два параметра:

```
ArcShape(float startAngle, float sweepAngle)
```

Первый параметр в конструкторе — угол начала прорисовки дуги в градусах, второй — угловой размер дуги в градусах.

Класс `ShapeDrawable`, подобно многим другим типам `Drawable`, входящим в пакет `android.graphics.drawable`, позволяет определять различные свойства рисунка с помощью набора открытых методов, например `setAlpha()` — для установки прозрачности, `setColorFilter()` — для установки цветового фильтра и т. д.

Продemonстрируем работу с классом `ShapeDrawable` и подклассами `Shape` на примере приложения, которое будет прорисовывать в объекте `ImageView` различные графические примитивы. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — `ShapeDrawable`;
- ☐ **Application name** — `ShapeDrawable Sample`;
- ☐ **Package name** — `com.samples.res.shapedrawable`;
- ☐ **Create Activity** — `ShapeDrawableActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch22_ShapeDrawable`.

Создайте файл компоновки с единственным виджетом `ImageView`, как показано в листинге 22.4.

Листинг 22.4. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/root"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal"
    android:layout_gravity="center_vertical|center_horizontal">

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minHeight="90px"
        android:minWidth="90px"
        android:layout_margin="115px"/>

</LinearLayout>
```

В классе `Activity` `DrawCanvasActivity` приложения создадим меню из шести опций:

- ❑ **Line** — прорисовка линии;
- ❑ **Oval** — прорисовка эллипса;
- ❑ **Rectangle** — прорисовка прямоугольника;
- ❑ **Round Rect. Fill** — прорисовка прямоугольника с закругленными углами;
- ❑ **Path** — прорисовка пятиконечной звезды с помощью линий;
- ❑ **Arc** — прорисовка дуги.

При выборе пункта меню в элементе `ImageView` будет прорисовываться соответствующий графический примитив. В методе обратного вызова `onOptionsItemSelected()` реализованы приведенные ранее фрагменты кода прорисовки фигур. Полный код класса `ShapeDrawableActivity` представлен в листинге 22.5.

Листинг 22.5. Файл класса окна приложения `ShapeDrawableActivity.java`

```
package com.samples.res.shapedrawable;

import android.app.Activity;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.RectF;
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.ArcShape;
import android.graphics.drawable.shapes.OvalShape;
import android.graphics.drawable.shapes.PathShape;
import android.graphics.drawable.shapes.RectShape;
import android.graphics.drawable.shapes.RoundRectShape;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;

public class ShapeDrawableActivity extends Activity {
    // Идентификаторы опций меню
    private static final int IDM_LINE = 101;
    private static final int IDM_OVAL = 102;
    private static final int IDM_RECT = 103;
    private static final int IDM_ROUNDRECT_FILL = 104;
    private static final int IDM_STAR = 105;
    private static final int IDM_ARC = 106;

    private ImageView image;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```

        image = (ImageView) findViewById(R.id.image);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        menu.add(Menu.NONE, IDM_LINE, Menu.NONE, "Line");
        menu.add(Menu.NONE, IDM_OVAL, Menu.NONE, "Oval");
        menu.add(Menu.NONE, IDM_RECT, Menu.NONE, "Rectangle");
        menu.add(Menu.NONE, IDM_ROUNDRECT_FILL, Menu.NONE, "Round Rect. Fill");
        menu.add(Menu.NONE, IDM_STAR, Menu.NONE, "Path");
        menu.add(Menu.NONE, IDM_ARC, Menu.NONE, "Arc");
        return (super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ShapeDrawable d = null;

        switch (item.getItemId()) {
            case IDM_LINE:
                d = new ShapeDrawable(new RectShape());
                d.setIntrinsicHeight(2);
                d.setIntrinsicWidth(150);
                d.getPaint().setColor(Color.MAGENTA);
                break;
            case IDM_OVAL:
                d = new ShapeDrawable(new OvalShape());
                d.setIntrinsicHeight(100);
                d.setIntrinsicWidth(150);
                d.getPaint().setColor(Color.RED);
                break;
            case IDM_RECT:
                d = new ShapeDrawable(new RectShape());
                d.setIntrinsicHeight(100);
                d.setIntrinsicWidth(150);
                d.getPaint().setColor(Color.BLUE);
                break;
            case IDM_ROUNDRECT_FILL:
                float[] outR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };
                RectF rectF = new RectF(8, 8, 8, 8);
                float[] inR = new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };

                d = new ShapeDrawable(new RoundRectShape(outR, rectF, inR));
                d.setIntrinsicHeight(100);
                d.setIntrinsicWidth(150);
                d.getPaint().setColor(Color.WHITE);
                break;
        }
    }

```



```

case IDM_STAR:
    Path p = new Path();
    p.moveTo(50, 0);
    p.lineTo(25,100);
    p.lineTo(100,50);
    p.lineTo(0,50);
    p.lineTo(75,100);
    p.lineTo(50,0);

    d = new ShapeDrawable(new PathShape(p, 100, 100));
    d.setIntrinsicHeight(100);
    d.setIntrinsicWidth(100);
    d.getPaint().setColor(Color.YELLOW);
    d.getPaint().setStyle(Paint.Style.STROKE);
    break;
case IDM_ARC:
    d = new ShapeDrawable(new ArcShape(0, 255));
    d.setIntrinsicHeight(100);
    d.setIntrinsicWidth(100);
    d.getPaint().setColor(Color.YELLOW);
    break;
}
image.setBackgroundDrawable(d);
return true;
}
}

```

Скомпилируйте и запустите проект на выполнение. При выборе пункта меню на поверхности виджета `ImageView` будет прорисовываться соответствующий графический примитив (рис. 22.4).

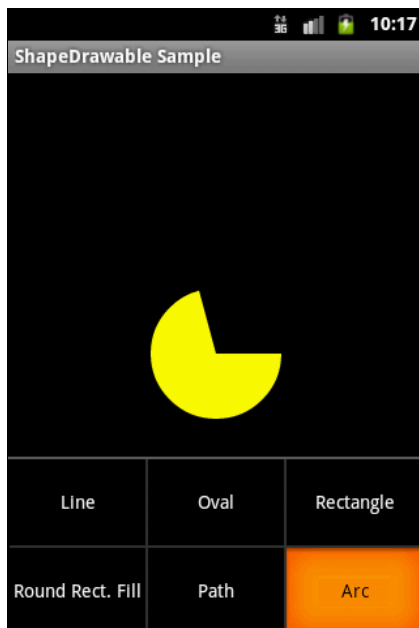


Рис. 22.4. Создание графических примитивов

Рисование на канве

Рисование на канве лучше всего использовать, когда окно приложения должно регулярно себя перерисовывать во время работы приложения. Например, при разработке игр необходимо создавать постоянно меняющуюся графику. Динамическое рисование на канве — процесс довольно медленный. Всего существует два способа реализации рисования на канве:

- ❑ в основном потоке программы, в котором запускается Activity, вы создаете собственный компонент View, затем вызываете метод `invalidate()` и обрабатываете создание графики в методе обратного вызова `onDraw()`;
- ❑ в отдельном потоке — через объект `SurfaceView`.

Класс `Canvas` имеет собственный набор методов для рисования, которые вы можете использовать, например `drawBitmap()`, `drawRect()`, `drawText()`. Другие классы, которые вы могли бы использовать, также имеют методы `draw()`. Например, можно создать объекты `Drawable` и передать их для прорисовки на канву. Класс `Drawable` имеет собственный метод `draw()`, который принимает объект `Canvas` как параметр.

Канва фактически является поверхностью, на которой ваша графика будет рисоваться. Когда вы выполняете прорисовку в пределах метода обратного вызова `View.onDraw()`, система передает в качестве параметра объект `Canvas`. Вы можете также получить объект `Canvas` вызовом метода `SurfaceHolder.lockCanvas()`, если имеете дело с объектом `SurfaceView`.

Система Android вызывает метод `onDraw()` по мере необходимости. Каждый раз, когда ваше изображение на канве представления требует перерисовки, необходимо вызывать метод `invalidate()`. Он требует от системы обновления представления, и система Android тогда вызовет ваш метод `onDraw()`.

Поскольку `ShapeDrawable` имеет свой собственный метод `draw()`, вы можете создать подкласс `View`, который рисует `ShapeDrawable` в коде метода обратного вызова `View.onDraw()`, например:

```
class CustomView extends View {
    private ShapeDrawable mDrawable;
    ...

    @Override
    protected void onDraw(Canvas canvas) {
        // Создаем прямоугольник
        mDrawable = new ShapeDrawable(new RectShape());
        mDrawable.setIntrinsicHeight(2);
        mDrawable.setIntrinsicWidth(150);
        mDrawable.getPaint().setColor(Color.MAGENTA);

        // Выводим прямоугольник на канву
        mDrawable.draw(canvas);
    }
}
```

Взяв за основу предыдущий пример, где мы загружали графические примитивы в фон виджета `ImageView`, создадим приложение, которое будет прорисовывать те же примитивы на канве представления. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- ❑ **Project name** — `Canvas`;
- ❑ **Application name** — `Draw on Canvas Sample`;
- ❑ **Package name** — `com.samples.res.canvas`;
- ❑ **Create Activity** — `ShapeDrawableActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch22_DrawCanvas`.

В приложении кроме класса `ShapeDrawableActivity`, представляющего окно приложения, потребуется дополнительный класс `DrawCanvasView`, производный от класса `View`, который и будет исполнять роль поверхности для рисования наших фигур. Объект `DrawCanvasView` будет основным представлением для окна приложения.

После создания графического примитива объект `ShapeDrawable` передается в объект `DrawCanvasView` для прорисовки на канве. Изменения в классе `ShapeDrawableActivity` представлены в листинге 22.6.

Листинг 22.6. Изменения в коде класса окна приложения `ShapeDrawableActivity.java`

```
public class ShapeDrawableActivity extends Activity {
    ...
    private DrawCanvasView view;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        view = new DrawCanvasView(this);
        setContentView(view);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ShapeDrawable d = new ShapeDrawable();
        switch (item.getItemId()) {
            ...
        }

        // Передаем созданный объект ShapeDrawable в View
        view.setDrawable(d);
        return true;
    }
}
```

В классе `DrawCanvasView` через открытый метод `setDrawable(ShapeDrawable)` передается объект `ShapeDrawable`, который будет прорисовываться на канве. В этом же методе вызывается `invalidate()`, требующий от системы перерисовки экрана. После выполнения система Android вызовет вашу реализацию метода `onDraw()`, который производит перерисовку канвы.

В реализации метода обратного вызова `onDraw()` компонента `View` используется объект `Canvas`, предоставляемый системой для всего вашего рисунка. Как только метод `onDraw()` будет выполнен, система Android будет использовать ваш объект `Canvas`, чтобы обновить графику на экране.

Полный код класса `DrawCanvasView` приведен в листинге 22.7.

Листинг 22.7. Файл класса `DrawCanvasView.java`

```
package com.samples.res.canvas;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.drawable.ShapeDrawable;
import android.view.View;

class DrawCanvasView extends View {
    // Константы, определяющие начальную координату объекта
    private static final int START_X = 10;
    private static final int START_Y = 10;

    private ShapeDrawable drawable;

    public DrawCanvasView(Context context) {
        super(context);
        setFocusable(true);
        drawable = new ShapeDrawable();
    }

    // Метод, загружающий объект ShapeDrawable для рисования
    public void setDrawable(ShapeDrawable shape) {
        drawable = shape;

        // Привязываем объект ShapeDrawable
        drawable.setBounds(START_X, START_Y,
            START_X + drawable.getIntrinsicWidth(),
            START_Y + drawable.getIntrinsicHeight() );
        this.getHeight();

        // Требуем перерисовки графики
        invalidate();
    }
}
```

```
// Перерисовка графического объекта
@Override
protected void onDraw(Canvas canvas) {
    drawable.draw(canvas);
}
```

Скомпилируйте и запустите проект на выполнение. Внешний вид приложения похож на предыдущий пример. При выборе пункта меню на канве будет прорисовываться соответствующий графический примитив (рис. 22.5).

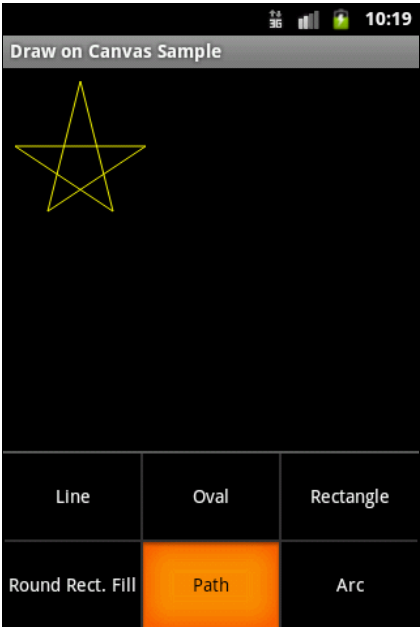


Рис. 22.5. Рисование графических объектов на канве

Резюме

В этой главе мы изучили возможности, предоставляемые платформой Android для работы с двумерной графикой. Библиотеки Android предлагают богатый набор классов для рисования графических примитивов и создания различных графических эффектов, с помощью которых вы можете украсить свое приложение.

В следующей главе мы изучим основы создания анимации для Android-приложений.



ГЛАВА 23

Создание анимации

В этой главе мы обсудим создание анимации для разработки визуально привлекательных приложений средствами Android SDK, который предоставляет двумерную графическую библиотеку анимации на канве и объектах `View` `android.view.animation` пакетов.

Анимация в Android представлена двумя видами:

- ❑ Tween Animation — анимация, выполняемая в виде простых преобразований объектов;
- ❑ Frame Animation — кадровая анимация.

Tween Animation

Анимация может выполняться в виде ряда простых преобразований — позиции, размера, вращения и прозрачности на поверхности объекта `View`. Так, например, для объекта `TextView` возможно перемещать, вращать, уменьшать или увеличивать текст. Если объект `TextView` имеет фоновое изображение, оно будет преобразовано наряду с текстом. Пакет `android.view.animation` включает все классы, используемые в анимации с промежуточными кадрами. Иерархия классов анимации представлена на рис. 23.1.

Основные классы анимации:

- ❑ `AnimationSet` — класс, представляющий группу анимаций, которые должны запускаться вместе. Если класс `AnimationSet` устанавливает какие-либо свойства, эти свойства наследуют и объекты, входящие в группу;

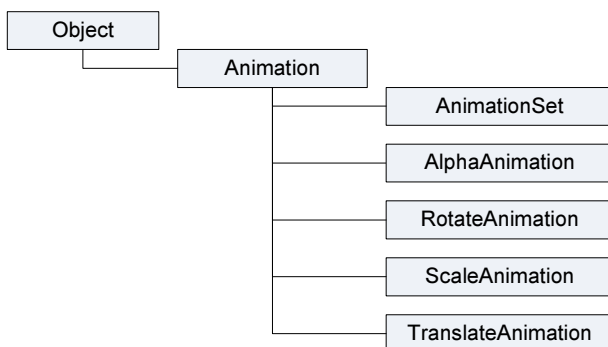


Рис. 23.1. Классы анимации в Android

- ❑ `AlphaAnimation` — класс анимации, который управляет прозрачностью объекта;
- ❑ `RotateAnimation` — класс анимации, который управляет вращением объекта;
- ❑ `ScaleAnimation` — класс анимации, который управляет масштабированием объекта;
- ❑ `TranslateAnimation` — класс анимации, который управляет позиционированием объекта.

Команды анимации определяют преобразования, которые необходимо произвести над объектом. Преобразования могут быть последовательными или одновременными. Каждое преобразование принимает набор параметров, определенных для этого преобразования (начальный размер, конечный размер при изменении размера, стартовый угол и конечный угол при вращении объекта и т. д.), а также набор общих параметров (например, начального времени и продолжительности). Если требуется сделать несколько преобразований одновременно, им задают одинаковое начальное время. Если требуется сделать последовательные преобразования, задается их время старта плюс продолжительность предыдущего преобразования.

Последовательность команд анимации определяется в XML-файле или в программном коде. В принципе для создания анимации предпочтителен XML-файл (аналогично определению компоновки в XML-файле), по причине его возможности многократного использования и большей гибкости, чем при жестком программном кодировании анимации.

Создание анимации в XML-файле

XML-файл анимации располагают в каталоге `res/anim/` Android-проекта. Файл должен иметь единственный корневой элемент, это может быть любой из элементов:

- ❑ `<set>`;
- ❑ `<alpha>`;
- ❑ `<scale>`;
- ❑ `<translate>`;
- ❑ `<rotate>`.

Элемент `<set>` является контейнером для остальных четырех компонентов и может, в свою очередь, включать в себя другой контейнер `<set>`.

Пример возможной иерархии элементов XML-файла анимации показан на рис. 23.2.

По умолчанию все элементы применяются одновременно. Чтобы запускать элементы последовательно, необходимо определить атрибут `startOffset` и указать значение в миллисекундах, например:

```
android:startOffset="3000"
```

Перечисленные ранее элементы управления анимацией содержат множество атрибутов. У элементов `<alpha>`, `<scale>`, `<translate>`, `<rotate>` и `<set>` поддерживаются общие атрибуты, унаследованные от базового класса `Animation`:

- ❑ `duration` — продолжительность эффекта в миллисекундах;
- ❑ `startOffset` — начальное время смещения для этого эффекта, в миллисекундах;

- ❑ `fillBefore` — когда установлен в `true`, то преобразование анимации применяется перед началом анимации;
- ❑ `fillAfter` — когда установлен в `true`, то преобразование применяется после конца анимации;
- ❑ `repeatCount` — определяет число повторений анимации;
- ❑ `repeatMode` — определяет поведение анимации при ее окончании. Возможные варианты: перезапустить без изменений или полностью изменить анимацию;
- ❑ `zAdjustment` — определяет режим упорядочения оси Z, чтобы использовать при выполнении анимации (нормаль, вершина или основание);
- ❑ `interpolator` — определяет постоянную скорости, которая описывает динамику визуального Activity в зависимости от времени, или, говоря простым языком, определяет скорость изменения анимации. Можно использовать любой из элементов подкласса интерполятора, определенных в `R.styleable`, например:

```
android:interpolator="@android:anim/decelerate_interpolator"
```

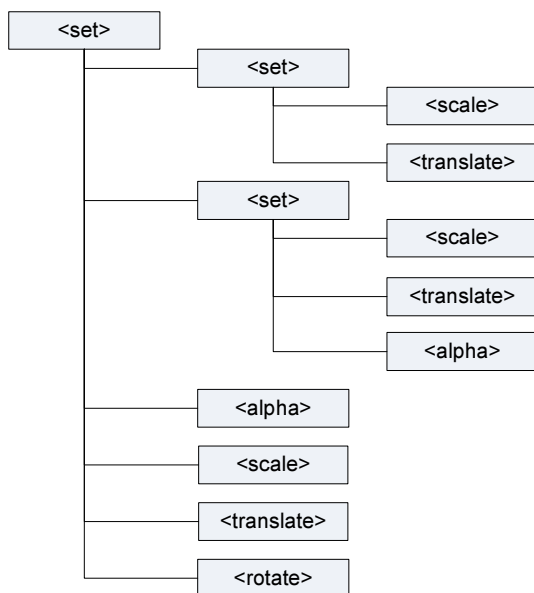


Рис. 23.2. Пример структуры XML-файла анимации

Элемент `<set>`

Элемент `<set>` — контейнер, который может содержать другие элементы. Представляет класс `AnimationSet`. Поддерживает атрибут `shareInterpolator`, который указывает на возможность совместного использования этого интерполятора для всех дочерних элементов.

Элемент **<alpha>**

Постепенно изменяющаяся анимация. Представляет `AlphaAnimation`. Поддерживает следующие атрибуты:

- ❑ `fromAlpha` — начальное значение прозрачности объекта;
- ❑ `toAlpha` — конечное значение прозрачности объекта.

Атрибуты содержат значение прозрачности от 0 до 1, где 0 означает полную прозрачность объекта.

Элемент **<scale>**

Элемент `<scale>` управляет анимацией изменения размеров объекта и представляет класс `ScaleAnimation`. Вы можете определить центральную точку изображения (закрепить центр анимации изображения), относительно которой будет изменяться масштабирование объекта. Элемент `<scale>` поддерживает следующие атрибуты:

- ❑ `fromXScale` — начальный масштаб по X;
- ❑ `toXScale` — конечный масштаб по X;
- ❑ `fromYScale` — начальный масштаб по Y;
- ❑ `toYScale` — конечный масштаб по Y;
- ❑ `pivotX` — X-координата закрепленного центра;
- ❑ `pivotY` — Y-координата закрепленного центра.

Элемент **<translate>**

Элемент `<translate>` — создает вертикальную или горизонтальную анимацию движения. Представляет класс `TranslateAnimation` и поддерживает следующие атрибуты:

- ❑ `fromXDelta` — начальное положение по X;
- ❑ `toXDelta` — конечное положение по X;
- ❑ `fromYDelta` — начальное положение по Y;
- ❑ `toYDelta` — конечное положение по Y.

Атрибуты должны быть в любом из следующих двух форматов:

- ❑ абсолютное значение;
- ❑ значения в процентах от -100 до 100%.

Элемент **<rotate>**

Элемент `<rotate>` предназначен для анимации вращения и представляет класс `RotateAnimation`. Поддерживает следующие атрибуты:

- ❑ `fromDegrees` — начальный угол вращения в градусах;
- ❑ `toDegrees` — конечный угол вращения в градусах;
- ❑ `pivotX` — координата X центра вращения в пикселах;
- ❑ `pivotY` — координата Y центра вращения в пикселах.

Анимация для графических примитивов

В приложение можно подключать несколько файлов анимации, используя их для одного объекта. Сначала мы рассмотрим работу с элементами анимации на примере графического примитива — прямоугольника, для которого используем все виды анимации, описанные в предыдущем разделе. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — TweenAnimationShapes;
- ☐ **Application name** — Tween Animation sample;
- ☐ **Package name** — com.samples.res.tweenanimationshapes;
- ☐ **Create Activity** — TweenAnimationActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch23_TweenAnimationShapes.

В каталоге `res/anim/` проекта создадим пять файлов с XML-анимацией: `alpha.xml`, `rotate.xml`, `scale.xml`, `translate.xml`, в каждом из которых продемонстрируем работу с соответствующим элементом, и файл `total.xml`, в котором продемонстрируем комбинацию элементов `<alpha>`, `<scale>`, `<translate>`, `<rotate>` для создания смешанной анимации.

Код файлов `alpha.xml`, `rotate.xml`, `scale.xml`, `translate.xml` и `total.xml` представлен в листингах 23.1—23.5.

Листинг 23.1. Файл `alpha.xml`

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:startOffset="0"
        android:duration="5000"/>

</set>
```

Листинг 23.2. Файл `rotate.xml`

```
<?xml version="1.0" encoding="utf-8" ?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
```

```
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
</set>
```

Листинг 23.3. Файл scale.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <scale
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="2.0"
        android:toYScale="2.0"
        android:duration="2500" />

    <scale
        android:startOffset="2500"
        android:duration="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5" />

</set>
```

Листинг 23.4. Файл translate.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <translate
        android:toYDelta="-100"
        android:fillAfter="true"
        android:duration="2500"/>

    <translate
        android:toYDelta="100"
        android:fillAfter="true"
        android:duration="2500"
        android:startOffset="2500"/>

</set>
```

Листинг 23.5. Файл total.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:startOffset="0"
        android:duration="5000"/>

    <scale
        android:duration="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="2.0"
        android:toYScale="2.0" />

    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />

    <translate
        android:toYDelta="-100"
        android:fillAfter="false"
        android:duration="2500" />

    <scale
        android:duration="2500"
        android:startOffset="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="0.5"
        android:toYScale="0.5" />

    <translate
        android:toYDelta="100"
        android:fillAfter="false"
        android:duration="2500"
        android:startOffset="2500" />

</set>
```

Графический примитив, изображение которого будет анимировано, также определим в отдельном XML-файле, который назовем `shape.xml` и сохраним в каталоге проекта `res/drawable/`. Это будет объект подкласса `ShapeDrawable` — прямоугольник красного цвета. Код файла `shape.xml` представлен в листинге 23.6.

Листинг 23.6. Файл shape.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <solid android:color="#F00"/>
</shape>
```

Файл компоновки для Activity приложения будет состоять из единственного элемента `ImageView`, как показано в листинге 23.7.

Листинг 23.7. Файл компоновки main.xml

```
<?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="horizontal"
    android:layout_gravity="center_vertical|center_horizontal">

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minHeight="100px"
        android:minWidth="100px"
        android:layout_margin="100px"/>

</LinearLayout>
```

Запустить анимацию в коде программы очень просто: надо создать объект `Animation` вызовом метода `AnimationUtils.loadAnimation()`, которому в качестве параметров следует передать контекст `Activity` и ссылку на XML-файл анимации. Затем запустить анимацию вызовом метода `View.startAnimation()`, передав в него объект `Animation`:

```
ImageView image = (ImageView) findViewById(R.id.image);
...
Animation animation =
    AnimationUtils.loadAnimation(this, R.anim.alpha);
image.startAnimation(animation);
```

В классе `Animation` есть вложенный интерфейс `AnimationListener`, в котором объявлены три метода обратного вызова:

- ☐ `onAnimationEnd()`;
- ☐ `onAnimationRepeat()`;
- ☐ `onAnimationStart()`.

В этих методах можно реализовать код обработки события запуска, окончания и перезапуска анимации. Например, по окончании анимации можно сделать объект анимации невидимым, а при запуске снова отобразить на экране:

```
@Override
public void onAnimationStart(Animation animation) {
    image.setVisibility(View.VISIBLE);
}

@Override
public void onAnimationEnd(Animation animation) {
    image.setVisibility(View.INVISIBLE);
}

@Override
public void onAnimationRepeat(Animation animation) {
    image.setVisibility(View.VISIBLE);
}
```

В классе Activity создадим меню из пяти пунктов, соответствующих каждому типу запускаемой анимации, — **Alpha**, **Scale**, **Translate**, **Rotate** и **Total**. В качестве идентификаторов пунктов меню используем идентификаторы ресурсов XML-файлов анимации, упростив тем самым структуру метода `onOptionsItemSelected()`, вызываемого при выборе пункта меню.

Полный код класса Activity `TweenAnimationActivity` представлен в листинге 23.8.

Листинг 23.8. Файл класса окна приложения `TweenAnimationActivity.java`

```
package com.samples.res.tweenanimaionshapes;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.Animation.AnimationListener;
import android.widget.ImageView;
import android.widget.Toast;

public class TweenAnimationActivity extends Activity
    implements AnimationListener{

    private ImageView image;
    private Animation animation;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        image = (ImageView)findViewById(R.id.image);
        image.setImageResource(R.drawable.shape);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(Menu.NONE, R.anim.alpha, Menu.NONE, "Alpha")
            .setAlphabeticShortcut('a');
        menu.add(Menu.NONE, R.anim.scale, Menu.NONE, "Scale")
            .setAlphabeticShortcut('s');
        menu.add(Menu.NONE, R.anim.translate, Menu.NONE, "Translate")
            .setAlphabeticShortcut('t');
        menu.add(Menu.NONE, R.anim.rotate, Menu.NONE, "Rotate")
            .setAlphabeticShortcut('r');
        menu.add(Menu.NONE, R.anim.total, Menu.NONE, "Total")
            .setAlphabeticShortcut('o');

        return(super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Загружаем анимацию из выбранного XML-файла
        animation = AnimationUtils.loadAnimation(
            this, item.getItemId());
        animation.setAnimationListener(this);

        image.startAnimation(animation);
        return true;
    }

    // Реализация интерфейса AnimationListener
    @Override
    public void onAnimationEnd(Animation animation) {
        image.setVisibility(View.INVISIBLE);
    }

    @Override
    public void onAnimationRepeat(Animation animation) {
        image.setVisibility(View.VISIBLE);
    }

    @Override
    public void onAnimationStart(Animation animation) {
        image.setVisibility(View.VISIBLE);
    }
}

```


Запустите проект на выполнение. Поочередно выбирая опции меню, просмотрите создаваемую анимацию для фигур. Внешний вид приложения показан на рис. 23.3.

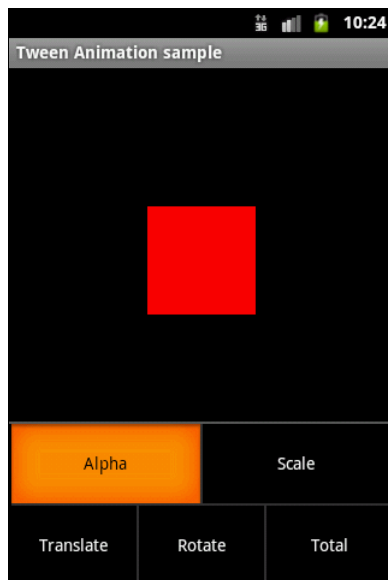


Рис. 23.3. Приложение с XML-анимацией

Анимация для графических файлов

Анимация для графических файлов ничем особым не отличается от анимации для графических примитивов. Рассмотрим на примере анимацию графического объекта, отображаемого в `ImageView`. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — `TweenAnimationView`;
- ☐ **Application name** — `Tween Animation Sample`;
- ☐ **Package name** — `com.samples.res.tweenanimationview`;
- ☐ **Create Activity** — `TweenAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch23_TweenAnimationView`.

Для изображения возьмите из каталога `Resources/Images/` архива книги файл `android_3d.png` (это фигурка андроида, которую мы уже не раз использовали в предыдущих главах).

В XML-файле анимации создадим более сложную структуру по сравнению с предыдущим примером: используем элемент `<scale>` для растягивания изображения и вложенный контейнер `<set>`. В контейнере `<set>` определим два дочерних элемента, `<scale>` и `<rotate>`, для одновременного вращения и изменения объекта `View`, и сохраним этот файл в каталоге `res/anim/` под именем `anim_android.xml`.

Полный код файла `anim_android.xml` представлен в листинге 23.9.

Листинг 23.9. Файл anim_android.xml

```

<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    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="3000" />

    <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="3000"
            android:duration="2000"
            android:fillBefore="false" />

        <rotate
            android:fromDegrees="0"
            android:toDegrees="-45"
            android:toYScale="0.0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="3000"
            android:duration="2000" />

    </set>
</set>

```

В файле компоновки для Activity создайте кнопку для запуска анимации и один элемент ImageView, как показано в листинге 23.10.

Листинг 23.10. Файл компоновки main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"

```

```
android:layout_width="fill_parent"
android:layout_height="fill_parent">

<LinearLayout
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_width="fill_parent">

    <Button
        android:id="@+id/btn_start"
        android:layout_height="wrap_content"
        android:text="Start"
        android:onClick="onClick"
        android:layout_width="fill_parent"
        android:layout_weight="1"/>

</LinearLayout>

<LinearLayout
    android:id="@+id/layout_anim"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:layout_width="fill_parent">

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minHeight="100px"
        android:minWidth="100px"
        android:layout_margin="100px"
        android:src="@drawable/android3d"/>

</LinearLayout>
</LinearLayout>
```

Полный код класса Activity TweenAnimationActivity представлен в листинге 23.11.

Листинг 23.11. Файл класса окна приложения TweenAnimationActivity.java

```
package com.samples.res.tweenanimationview;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;

public class TweenAnimationActivity extends Activity
    implements View.OnClickListener {
```

```

private ImageView image;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    image = (ImageView) findViewById(R.id.image);
}

@Override
public void onClick(View v) {
    // Запуск анимации
    Animation anim = AnimationUtils.loadAnimation(
        getApplicationContext(), R.anim.interpolator);
    image.startAnimation(anim);
}
}

```

Скомпилируйте проект и запустите его на выполнение. При нажатии кнопки **Start** фигурка андроида сначала плавно растянется по горизонтали, затем одновременно повернется и уменьшится в размерах, после чего вернется в исходное состояние. Внешний вид приложения представлен на рис. 23.4.

Можете в качестве упражнения поэкспериментировать с атрибутами элементов в XML-файле анимации и посмотреть получившиеся результаты.

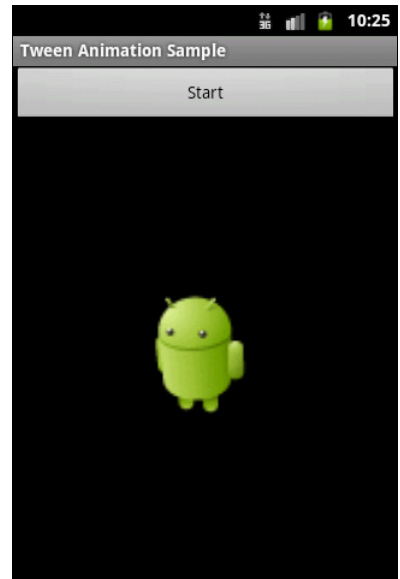


Рис. 23.4. Анимация отдельного объекта `View`

Независимо от того, как анимация изменяет размеры объекта или перемещает его на плоскости, границы элемента `View`, в который загружено изображение, останутся неизменными: ваша анимация не будет автоматически корректировать размеры для размещения объекта. Если анимация выйдет за границы родительского элемента `View`, произойдет отсечение объекта анимации.

Анимация для группы объектов

Анимацию можно сделать и для нескольких объектов, объединив их в группу. Например, поместив элементы в контейнер `LinearLayout`, причем можно использовать не только графические, но и текстовые объекты. Рассмотрим на примере анимацию двух

элементов, `ImageView` и `TextView`, объединенных в группу. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — `TweenAnimationLayout`;
- ☐ **Application name** — `Tween Animation on Layout`;
- ☐ **Package name** — `com.samples.res.tweenanimationview`;
- ☐ **Create Activity** — `TweenAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch23_TweenAnimationLayout`.

Усложним XML-файл анимации, добавив элементы для создания более интересных эффектов — используем элемент `<alpha>` для изменения прозрачности объектов и последовательность элемента `<rotate>` и нескольких элементов `<scale>` и `<rotate>` для вращения и масштабирования. Сохраним этот файл в каталоге `res/anim/` проекта под именем `circle.xml`. Полный код файла `circle.xml` представлен в листинге 23.12.

Листинг 23.12. Файл анимации `circle.xml`

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:shareInterpolator="false">

    <alpha
        android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:startOffset="0"
        android:duration="5000"/>

    <scale
        android:duration="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:toXScale="2.0"
        android:toYScale="2.0" />

    <rotate
        android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />

    <scale
        android:duration="2500"
        android:startOffset="2500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
```

```

        android:toXScale="0.5"
        android:toYScale="0.5" />
<scale
    android:duration="2500"
    android:startOffset="5000"
    android:pivotX="50%"
    android:pivotY="50%"
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:toXScale="1.25"
    android:toYScale="1.25" />
<scale
    android:duration="2500"
    android:startOffset="7500"
    android:pivotX="50%"
    android:pivotY="50%"
    android:fromXScale="1.0"
    android:fromYScale="1.0"
    android:toXScale="0.80"
    android:toYScale="0.80" />
</set>

```

В файле компоновки для Activity приложения создайте кнопку и дочерний контейнер `LinearLayout`, в котором разместите виджет `ImageView` с фигуркой андроида из предыдущего приложения и текстовое поле `TextView` с надписью "Hello, Android!". Для дочернего контейнера обязательно присвойте идентификатор:

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

по которому вы сможете загрузить его в код программы и использовать для анимации. Код файла компоновки показан в листинге 23.13.

Листинг 23.13. Файл компоновки `main.xml`

```

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

    <LinearLayout
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_width="fill_parent">
        <Button
            android:id="@+id/btn_start"
            android:layout_height="wrap_content"
            android:text="Start"
            android:onClick="onClick"
            android:layout_width="fill_parent"

```

```

        android:layout_weight="1"/>
    </LinearLayout>

    <LinearLayout
        android:id="@+id/layout_anim"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:layout_width="fill_parent">

        <ImageView
            android:id="@+id/image"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/android3d"/>

        <TextView
            android:id="@+id/text"
            android:text="@string/hello"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
    </LinearLayout>
</LinearLayout>

```

Внешний вид компоновки должен получиться таким, как на рис. 23.5.

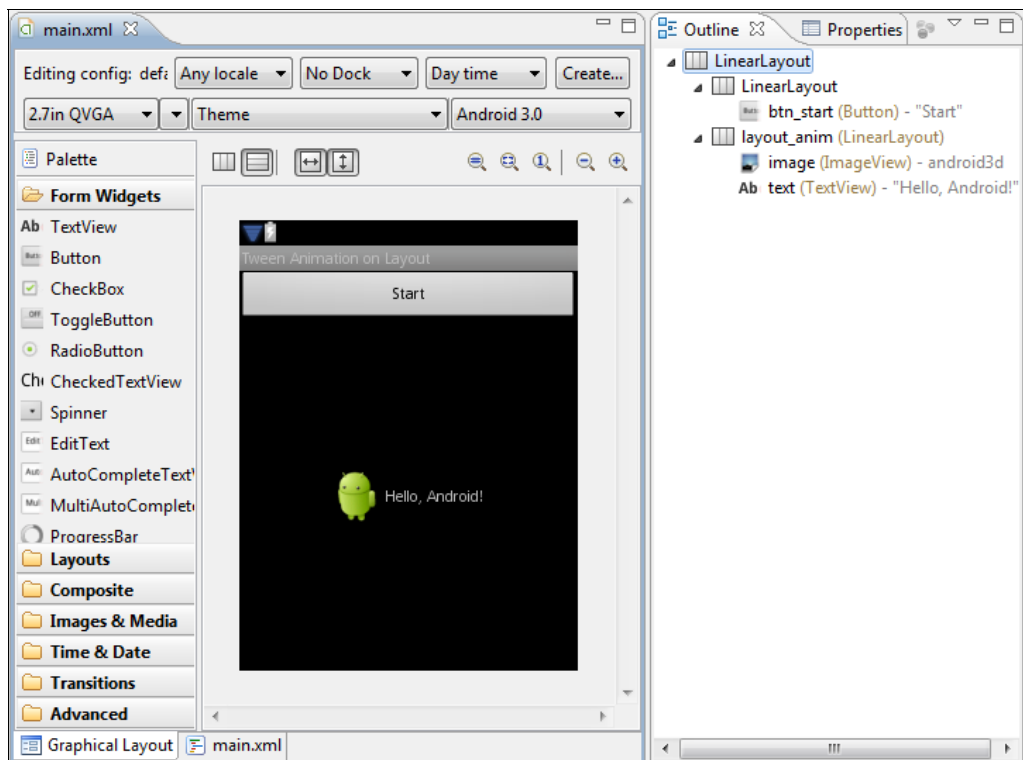


Рис. 23.5. Компоновка для приложения

Код класса Activity почти не отличается от предыдущего примера за исключением того, что мы работаем с анимацией не отдельного элемента, а с анимацией группы элементов:

```
layout = (LinearLayout) findViewById(R.id.layout_anim);
animation = AnimationUtils.loadAnimation(this, R.anim.circle);
```

Полный код класса TweenAnimationActivity представлен в листинге 23.14.

Листинг 23.14. Файл класса окна приложения TweenAnimationActivity.java

```
package com.samples.res.tweenanimationlayout;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.LinearLayout;

public class TweenAnimationAcctivity extends Activity
    implements View.OnClickListener {
    private LinearLayout layout;
    private Animation animation;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        layout = (LinearLayout) findViewById(R.id.layout_anim);
        animation = AnimationUtils.loadAnimation(this, R.anim.circle);
    }

    @Override
    public void onClick(View v) {
        layout.startAnimation(animation);
    }
}
```

Скомпилируйте и запустите проект на выполнение. Внешний вид приложения показан на рис. 23.6.

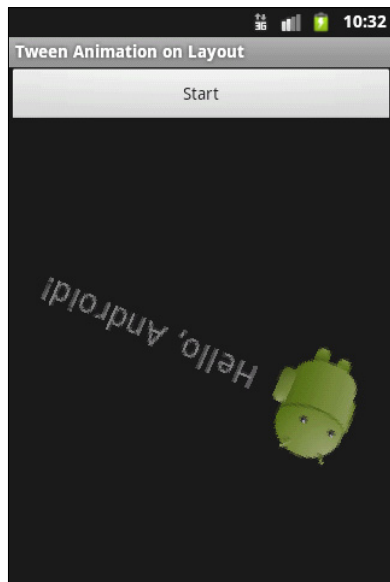


Рис. 23.6. Анимация компоновки с изображением и текстом

Frame Animation

Кадровая (фреймовая) анимация — традиционная анимация, которая создается последовательностью различных изображений подобно рулону пленки. Основой для кадровой анимации является класс `AnimationDrawable`.

Подобно анимации преобразований, о которой было рассказано ранее, XML-файлы для этого вида анимации располагают в каталоге `res/anim/` Android-проекта.

Создание анимации в XML

Для кадровой анимации XML-файл состоит из корневого элемента `<animation-list>` и дочерних узлов `<item>`, каждый из которых определяет кадр с двумя составляющими:

- ☐ графическим ресурсом для кадра;
- ☐ продолжительностью кадра.

Вот примерный XML-файл для анимации фрейма:

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">

    <item android:drawable="@drawable/file1" android:duration="200" />
    <item android:drawable="@drawable/file2" android:duration="200" />
    <item android:drawable="@drawable/file3" android:duration="200" />

</animation-list>
```

Эта анимация выполняется только для трех кадров. При установке атрибута `android:oneshot` списка в `true` анимация повторится только один раз и после остановки будет содержать последний кадр. Если же атрибут `android:oneshot` установлен в `false`, то анимация будет циклической. Этот XML-файл, сохраненный в каталоге `res/anim/` проекта, можно добавить как фоновое изображение и затем запустить анимацию.

Давайте рассмотрим создание кадровой анимации на примере. Создайте в Eclipse новый проект и заполните поля в окне **New Android Project**:

- ☐ **Project name** — `FrameAnimationXML`;
- ☐ **Application name** — `Frame Animation sample`;
- ☐ **Package name** — `com.samples.res.frameanimationxml`;
- ☐ **Create Activity** — `FrameAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch23_FrameAnimationXML`.

В файле компоновки `main.xml` создадим две кнопки для запуска и остановки анимации и виджет `ImageView` для отображения анимированного рисунка. Код файла компоновки показан в листинге 23.15.

Листинг 23.15. Файл компоновки main.xml

```

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

    <LinearLayout
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_width="fill_parent">

        <Button
            android:id="@+id/btn_start"
            android:layout_height="wrap_content"
            android:text="Start"
            android:onClick="onClick"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>

        <Button
            android:id="@+id/btn_stop"
            android:layout_height="wrap_content"
            android:text="Stop"
            android:onClick="onClick"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>

    </LinearLayout>

    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"/>

</LinearLayout>

```

Для анимации используем файлы android1.png, android2.png, android3.png, находящиеся в каталоге Resources/Animation/ архива книги. Таким образом, наша анимация будет состоять из трех кадров. Время показа каждого кадра установим в 300 миллисекунд. Все это запишем в XML-файл, который сохраним под именем android_anim.xml в каталоге res/values/. Полный код файла приведен в листинге 23.16.

Листинг 23.16. Файл анимации android_anim.xml

```

<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">

```

```

<item
    android:drawable="@drawable/android1"
    android:duration="300"/>
<item
    android:drawable="@drawable/android2"
    android:duration="300"/>
<item
    android:drawable="@drawable/android3"
    android:duration="300"/>

</animation-list>

```

Получить объект `AnimationDrawable` в коде программы можно следующим способом:

```

ImageView image = (ImageView)findViewById(R.id.image);
image.setBackgroundResource(R.anim.android_anim);
AnimationDrawable animation = (AnimationDrawable)image.getBackground();

```

Управлять объектом `AnimationDrawable` можно, используя его открытые методы `start()` и `stop()`.

Полный код класса `Activity` `FrameAnimationActivity` приведен в листинге 23.17.

Листинг 23.17. Файл класса окна приложения `FrameAnimationActivity.java`

```

package com.samples.res.frameanimationxml;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class FrameAnimationActivity extends Activity
    implements View.OnClickListener {

    private AnimationDrawable animation;

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

        ImageView image = (ImageView)findViewById(R.id.image);
        image.setBackgroundResource(R.anim.android_anim);

        animation = (AnimationDrawable)image.getBackground();
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {

```

```
        case R.id.btn_start:
            animation.start();
            break;
        case R.id.btn_stop:
            animation.stop();
            break;
    }
}
```

Скомпилируйте проект и запустите его на выполнение. У нас получится анимированная фигурка андроида, выполняющая физзарядку (рис. 23.7).

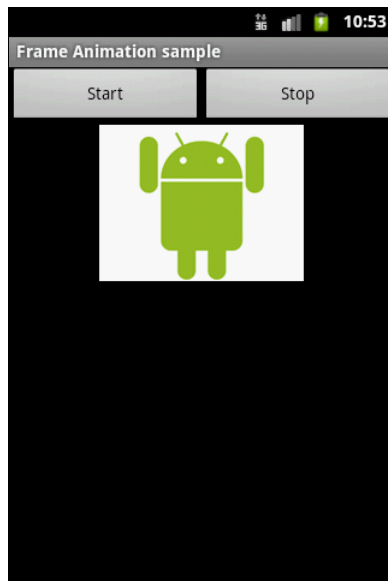


Рис. 23.7. Приложение с фреймовой анимацией

Создание анимации в коде программы

В отличие от анимации преобразований, кодирование для кадровой анимации более простое — достаточно загрузить последовательно ресурсы кадров и определить время показа для каждого кадра.

В качестве примера создадим в Eclipse новый проект, взяв за основу приложение для кадровой анимации из предыдущего раздела.

Заполним поля в окне **New Android Project**:

- ☐ **Project name** — `FrameAnimationImageView`;
- ☐ **Application name** — `Frame Animation Sample`;
- ☐ **Package name** — `com.samples.res.frameanimationview`;
- ☐ **Create Activity** — `FrameAnimationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch23_FrameAnimationImageView`.

Файл компоновки для Activity используйте из листинга 23.15 предыдущего примера с анимацией. В классе окна приложения `FrameAnimationActivity` определим два внутренних метода, `start()` и `stop()`, которые будем вызывать из обработчиков события `onClick()` кнопок **Start** и **Stop**.

В методе `start()` реализуем создание анимации. Для этого сначала надо получить кадры анимации в виде набора объектов `Drawable`, загрузив изображения из ресурсов. Для каждого кадра создается отдельный объект `Drawable`:

```
BitmapDrawable frame1 =
    (BitmapDrawable) getResources().getDrawable(R.drawable.android1);
BitmapDrawable frame2 =
    (BitmapDrawable) getResources().getDrawable(R.drawable.android2);
BitmapDrawable frame3 =
    (BitmapDrawable) getResources().getDrawable(R.drawable.android3);
```

Созданные объекты `BitmapDrawable` необходимо добавить в объект `AnimationDrawable` методом `addFrame()`. Метод `addFrame()` принимает два параметра: кадр анимации (объект `Drawable`) и продолжительность показа в миллисекундах. Код для создания анимации должен выглядеть примерно так:

```
AnimationDrawable mAnimation = new AnimationDrawable();
// Устанавливаем циклическое повторение анимации
mAnimation.setOneShot(false);
mAnimation.addFrame(frame1, 100);
mAnimation.addFrame(frame2, 100);
mAnimation.addFrame(frame3, 100);

// Устанавливаем анимацию как фон для ImageView
image.setBackgroundDrawable(mAnimation);

// Делаем объект Drawable видимым
mAnimation.setVisible(true, true);
mAnimation.start();
```

Полный код класса Activity `FrameAnimationActivity` приводится в листинге 23.18.

Листинг 23.18. Файл класса окна приложения `FrameAnimationActivity.java`

```
package com.samples.res.frameaninimageview;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class FrameAnimationActivity extends Activity
    implements View.OnClickListener {
    private final static int DURATION = 300;
```

```
private AnimationDrawable animation = null;
private ImageView image;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    image = (ImageView) findViewById(R.id.image);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.btn_start:
            start();
            break;
        case R.id.btn_stop:
            stop();
            break;
    }
}

private void start()
{
    BitmapDrawable frame1 =
        (BitmapDrawable) getResources().getDrawable(R.drawable.android1);
    BitmapDrawable frame2 =
        (BitmapDrawable) getResources().getDrawable(R.drawable.android2);
    BitmapDrawable frame3 =
        (BitmapDrawable) getResources().getDrawable(R.drawable.android3);

    animation = new AnimationDrawable();
    animation.setOneShot(false);
    animation.addFrame(frame1, DURATION);
    animation.addFrame(frame2, DURATION);
    animation.addFrame(frame3, DURATION);

    image.setBackgroundDrawable(animation);

    animation.setVisible(true,true);
    animation.start();
}

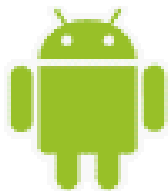
private void stop()
{
    animation.stop();
    animation.setVisible(false,false);
}
}
```

Внешний вид работающего приложения ничем не отличается от приложения с загрузкой анимации из XML-файла (см. рис. 23.7 из предыдущего примера).

Резюме

В этой главе мы рассмотрели вопросы использования анимации в Android-приложениях. Как вы убедились, платформа Android предоставляет множество способов "оживления" создаваемых приложений. Кроме того, платформа Android предоставляет библиотеку OpenGL для поддержки 3D-графики, но это очень обширная тема, которую лучше рассматривать в рамках отдельной книги.

Далее мы переходим к следующей части книги, где будем рассматривать создание и вызов служб, работу с компонентом Broadcast Receiver и системными службами Android.



ЧАСТЬ IV

Системные службы

Глава 24. Компонент Service

Глава 25. Broadcast Receiver

Глава 26. Home Screen

Глава 27. Уведомления в строке состояния

Глава 28. Action Bar

Глава 29. Служба оповещений

Глава 30. Буфер обмена и API для работы с текстом



ГЛАВА 24

Компонент Service

Служба (Service) в системе Android является таким же компонентом, как и Activity. Но в отличие от Activity службы в Android работают как фоновые процессы. Они не имеют пользовательского интерфейса, что делает их идеальными для задач, не требующих вмешательства пользователя. Служба будет продолжать работать до тех пор, пока кто-нибудь не остановит ее или пока она не остановит себя сама.

Клиентские приложения могут через объекты Intent устанавливать подключение к службам и использовать это подключение для взаимодействия со службой. С одной и той же службой может связываться множество клиентских приложений.

Работа служб в Android

Службу в системе Android представляет класс `Service`. Так же, как и Activity (см. главу 12), служба имеет свой набор методов, которые вы можете реализовать при разработке, чтобы контролировать изменения в состоянии службы при ее работе. Но в отличие от класса Activity у класса Service этих методов только три:

- ❑ `void onCreate();`
- ❑ `void onStart(Intent intent);`
- ❑ `void onDestroy();`

Реализовывая эти методы обратного вызова в своей службе, вы можете контролировать вложенные жизненные циклы службы.

Из клиентского приложения службу можно запустить вызовом метода `Context.startService()`, а остановить через вызов `Context.stopService()`. Служба может остановить сама себя, вызывая методы `Service.stopSelf()` или `Service.stopSelfResult()`.

Можно установить подключение к работающей службе и использовать это подключение для взаимодействия со службой. Подключение устанавливают вызовом метода `Context.bindService()` и закрывают вызовом `Context.unbindService()`. Если служба уже была остановлена, вызов метода `bindService()` может ее запустить.

Методы `onCreate()` и `onDestroy()` вызываются для всех служб независимо от того, запускаются ли они через `Context.startService()` или `Context.bindService()`. Однако метод обратного вызова `onStart()` вызывается только для служб, запущенных вызовом метода `startService()`. Любая служба, независимо от того, как она стартовала, может потенциально позволить клиентам связываться с собой, т. е. любая служба может полу-

чать клиентские запросы. Если служба разрешает другим приложениям связываться с собой, то привязка осуществляется с помощью дополнительных методов:

```
❑ IBinder onBind(Intent intent);
❑ boolean onBind(Intent intent);
❑ void onBind(Intent intent).
```

В метод обратного вызова `onBind()` передают объект `Intent`, который был параметром в методе `bindService()`, а в метод обратного вызова `onUnbind()` — объект `Intent`, который передавали в метод `unbindService()`. Если служба разрешает связывание, метод `onBind()` возвращает канал связи, который используют клиенты, чтобы взаимодействовать со службой. Метод обратного вызова `onRebind()` может быть вызван после `onUnbind()`, если новый клиент соединяется со службой.

Создание службы

Чтобы определить службу, необходимо создать новый класс, расширяющий базовый класс `Service`. В создаваемом классе надо будет определить методы обратного вызова `onBind()` и `onCreate()`, как показано далее:

```
public class MyService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        // Действия при связывании клиента со службой
    }

    @Override
    public void onCreate() {
        // Инициализация службы при создании
    }
    ...
}
```

В большинстве случаев необходимо будет реализовать в классе метод `onStart()`. Этот метод выполняется всякий раз, когда служба запускается вызовом метода `startService()`.

```
@Override
public void onStart(Intent intent, int startId) {
    // Действия при запуске службы
}
```

Как только вы создали новую службу, ее необходимо зарегистрировать в манифесте приложения. Для этого используется элемент `<service>` внутри элемента `<application>`. Вы можете использовать атрибуты элемента `<service>`, чтобы запускать или останавливать службу. Например, зарегистрировать службу с именем `MyService` можно так:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
```

```
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name">
    ...
    <service
        android:enabled="true"
        android:name=".MyService">
    </service>
    ...
</application>
...
</manifest>
```

Для службы также можно определять разрешения, которые требуются для управления этой службой из других приложений, используя флаги в элементе `<requires-permission>`.

Вызов службы

Чтобы запустить службу, в клиентском приложении необходимо вызывать метод `startService()`. Существует два способа вызова службы:

- явный вызов службы;
- неявный вызов службы.

Пример явного вызова службы с именем `MyService` может выглядеть следующим образом:

```
startService(new Intent(this, MyService.class));
```

Также можно явно определить службу, создав экземпляр класса этой службы.

Пример неявного вызова службы выглядит так:

```
startService(new Intent(MyService.SERVICE_ACTION));
```

Чтобы использовать этот пример, необходимо включить константу `SERVICE_ACTION`, идентифицирующую службу, в класс `MyService`, например:

```
private static String SERVICE_ACTION = "com.samples.app.media.PLAYER";
```

и использовать фильтр намерений, чтобы зарегистрировать его как провайдера `SERVICE_ACTION`. Если служба потребует разрешений, которые не имеет ваше приложение, то этот запрос вызовет исключение `SecurityException`.

Чтобы остановить службу, необходимо вызвать метод `stopService()` и передать намерение, которое определено службой для остановки:

```
stopService(new Intent(this, service.getClass()));
```

Если метод `startService()` вызывают для службы, которая уже запущена, метод `onStart()` службы будет выполнен заново. Клиентские вызовы метода `startService()` не являются вложенными, таким образом, единственный вызов метода `stopService()` останавливает службу независимо от того, сколько было вызовов метода `startService()`.

Давайте теперь создадим практическое приложение-службу. Наша служба будет запускать на воспроизведение в фоновом режиме музыкальный файл. Управлять службой можно будет из Activity. Создайте новый проект и в окне **Create New Project** введите следующие значения:

- ☐ **Project name** — ServiceLauncher;
- ☐ **Application name** — Service Launcher Sample;
- ☐ **Package name** — com.samples.app.servicelaunch;
- ☐ **Create Activity** — LaunchActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch24_ServiceLauncher.

Для службы создайте отдельный класс `PlayService`, наследуемый от класса `Service`. Служба будет загружать музыкальный файл `sample.mp3` из каталога `res/raw/` (если вы используете пример из архива книги, там этот файл отсутствует, поэтому разместите в данном каталоге файл с вашей любимой музыкой). Полный код класса службы приведен в листинге 24.1.

Листинг 24.1. Файл класса службы PlayService.java

```
package com.samples.app.servicelaunch;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.widget.Toast;

public class PlayService extends Service {
    MediaPlayer player;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        Toast.makeText(this, "Service Created", Toast.LENGTH_SHORT).show();
        player = MediaPlayer.create(this, R.raw.sample);
        player.setLooping(false);
    }

    @Override
    public void onStart(Intent intent, int startid) {
        Toast.makeText(this, "Service Started", Toast.LENGTH_SHORT).show();
        player.start();
    }
}
```

```
@Override
public void onDestroy() {
    Toast.makeText(this, "Service Stopped", Toast.LENGTH_SHORT).show();
    player.stop();
}
```

В файле манифеста приложения необходимо зарегистрировать службу, написав код вручную, или используя редактор манифеста, как было рассказано ранее. Код файла `AndroidManifest.xml` представлен в листинге 24.2.

Листинг 24.2. Файл `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.servicelaunch"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity
            android:name=".LaunchActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service
            android:enabled="true"
            android:name=".PlayService">
        </service>
    </application>
    <uses-sdk android:minSdkVersion="10" />
</manifest>
```

В файле компоновки для Activity определим две кнопки: **Start Player** и **Stop Player** (листинг 24.3).

Листинг 24.3. Файл компоновки `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">
```

```

<Button
    android:id="@+id/btn_start"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:text="Start Player"
    android:onClick="onClick" />
<Button
    android:id="@+id/btn_stop"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Stop Player"
    android:onClick="onClick" />

</LinearLayout>

```

В классе `Activity` в обработчиках событий кнопок будем вызывать методы `startService()` и `stopService()` для управления службой. Полный код класса приведен в листинге 24.4.

Листинг 24.4. Файл класса окна приложения `LaunchActivity.java`

```

package com.samples.app.servicelaunch;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class LaunchActivity extends Activity
    implements View.OnClickListener {

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

    @Override
    public void onClick(View view) {
        switch (view.getId()) {
            case R.id.btn_start:
                startService(new Intent(LaunchActivity.this, PlayService.class));
                break;
            case R.id.btn_stop:
                stopService(new Intent(LaunchActivity.this, PlayService.class));
                break;
        }
    }
}

```


Внешний вид созданного приложения со службой приведен на рис. 24.1. Запущенная служба будет выполняться независимо от состояния Activity, несмотря на то что эти компоненты находятся в одном приложении: если ее завершить, служба все равно останется работать.

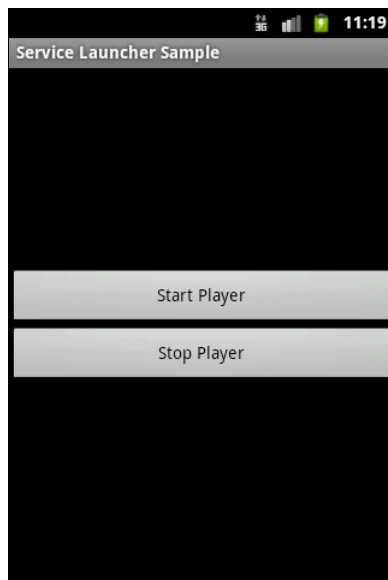


Рис. 24.1. Приложение для вызова службы

Доступ к системным и сетевым сервисам

Система Android — это многофункциональная платформа, внутри нее уже реализованы многочисленные компоненты, которые можно использовать в ваших приложениях. Поэтому при разработке собственных приложений нет необходимости заново изобретать велосипед, а лучше по возможности использовать функциональность существующих компонентов системы Android.

Не стоит также забывать и тот факт, что мобильный телефон является в первую очередь средством коммуникации и обладает, помимо обеспечения связи между абонентами сотовой сети, еще и доступом к Интернету. Это означает, что в ваших приложениях можно использовать различные удаленные сетевые службы Google, например, Google Maps, Street View и др.

Платформа Android предоставляет в ваше распоряжение множество служб, функциональность которых вы можете использовать в разрабатываемых приложениях. Количество служб с выходом каждой новой версии постоянно увеличивается, и на момент выхода версии 4.03 их уже более 35.

Мы кратко рассмотрим доступные в системе Android службы и их предназначение, чтобы вы могли получить полное представление о возможностях служб, входящих в состав платформы Android.

Мобильный телефон Android предоставляет пользователю удобный и красивый графический интерфейс. Для его поддержки в системе существует группа служб для управления графическим пользовательским интерфейсом:

- ❑ *Wallpaper Service* — служба для управления "обоями", загружаемыми на Home Screen (домашний экран) мобильного устройства;
- ❑ *Layout Inflater Service* — служба для управления компоновкой окон при их динамической загрузке;

- ❑ *UI Mode Service* — служба, предназначенная для управления режимами работы пользовательского интерфейса.

Для организации взаимодействия пользователя с мобильным устройством в состав платформы Android включены службы, которые обеспечивают мониторинг и управление пользовательским вводом:

- ❑ *Input Method Service* — служба, используемая для текстового ввода. Данная служба обеспечивает отображение автодополнения при вводе текста пользователем в текстовое поле. Например, эта служба работает, когда вы вводите текст SMS-сообщения;
- ❑ *Accessibility Service* — обеспечивает доступ к событиям, возникающим в пользовательском интерфейсе, например, при нажатии пользователем на кнопку, получении или потере фокуса виджетом;
- ❑ *Clipboard Service* — управляет буфером обмена. Буфер обмена в Android, в отличие от других систем, является глобальным, т. е. виден для всей системы, а не для отдельного приложения;
- ❑ *Search Service* — служба для управления функциями глобального поиска на устройстве;
- ❑ *Keyguard Service* — служба для управления блокировкой клавиатуры мобильного устройства.

Важную роль в платформе Android играет группа системных служб, предназначенных для организации взаимодействия компонентов операционной системы и приложений, ведения журнала системных событий и оповещения пользователя о событиях, происходящих в системе:

- ❑ *Notification Service* — управляет пользовательскими уведомлениями (например, при получении SMS-сообщения), которые отображаются в строке состояния мобильного устройства;
- ❑ *Alarm Service* — предназначена для отправления пользователю разовых или периодических оповещений в заданное время в виде звукового сигнала (например, будильник) или световых сигналов, виброзвонка, тестовых сообщений;
- ❑ *Window Service* — служба для доступа к системному менеджеру окон;
- ❑ *Activity Service* — служба для взаимодействия с объектами Activity, открытыми в данный момент времени в системе. Кстати, служба Activity Service может взаимодействовать не только с Activity, но и с другими службами, находящимися в системе;
- ❑ *Droptbox Service* — предназначена для обеспечения записи системных событий в диагностический файл.

Для управления медиа — записью и воспроизведением музыки и видео, громкостью, режимами телефонных звонков — на платформе Android предусмотрена служба *Audio Service*.

Для работы с оборудованием на платформе Android также существует набор служб, которые обеспечивают доступ к "железу" мобильного телефона и использование его в приложениях:

- ❑ *Power Service* — служба для управления энергопотреблением. Эта служба предназначена в первую очередь для эффективного использования батареи на мобильном устройстве. Она позволяет управлять энергопотреблением процессора, подсветкой экрана и клавиатуры (если таковая имеется на телефоне);
- ❑ *Battery Manager* — служба для контроля состояния аккумуляторной батареи. Эта служба получает данные о степени заряда батареи, ее температуре, подключении зарядного устройства и другую важную информацию об источнике питания мобильного устройства;
- ❑ *Sensor Service* — служба для доступа к встроенным датчикам, например акселерометру, датчику температуры, освещенности мобильного устройства;
- ❑ *Storage Service* — служба для управления сохранением данных в файловой системе: во внутренней памяти устройства и на съемной карте памяти;
- ❑ *Vibrator Service* — служба, обеспечивающая доступ и управление вибровзвоном мобильного устройства.

Поскольку основным назначением мобильного устройства является обеспечение коммуникации, система Android располагает большим количеством служб для управления сетевыми соединениями и передачей данных:

- ❑ *Telephony Service* — служба для управления телефонными функциями мобильного устройства. Эта служба управляет телефонными вызовами, отправкой и приемом SMS-сообщений;
- ❑ *Connectivity Service* — служба для управления сетевыми соединениями мобильного телефона;
- ❑ *Download Service* — служба для управления загрузками данных через HTTP;
- ❑ *Wifi Service* — служба для управления сетевым соединением Wi-Fi;
- ❑ *Location Service* — служба для отслеживания физических координат местоположения мобильного устройства. Эта служба использует провайдеров местоположения и сетевые сервисы Google;
- ❑ *NFC Service* — служба для управления NFC (Near Field Communication). Это технология беспроводной высокочастотной связи малого радиуса действия, позволяющая производить соединение и обмен данными между устройствами, находящимися на расстоянии нескольких сантиметров. Эта новая технология, предназначенная для мобильных телефонов и платежных систем.

В мобильном устройстве Android также предусмотрен набор служб для управления безопасностью, учетными записями и политиками безопасности:

- ❑ *Account Service* — используется для управления пользовательскими учетными записями, предназначенными для доступа к различным онлайн-сервисам, например Gmail, Facebook и др.;
- ❑ *Device_policy Service* — предназначена для администрирования устройства и управления пользовательскими политиками безопасности, например, устанавливает минимальную длину пароля, время действия пароля и др.

Для доступа к службам в приложениях в Android SDK используются менеджеры служб. У каждой службы, за редким исключением, есть собственный менеджер. Так, напри-

мер, для управления службой Power Service используется *Power Manager*, для Telephony Service — *Telephony Manager* и т. д.

Подробно все службы платформы Android мы рассмотреть не можем ввиду ограниченного объема книги, но применению большинства из перечисленных служб, которые могут наиболее часто использоваться в приложениях, мы уделим достаточно внимания в следующих главах книги.

Резюме

В этой главе мы рассмотрели создание служб, вызов и управление этими службами из кода приложения. Система Android также содержит большой набор встроенных системных служб и может взаимодействовать с удаленными сервисами. Работой с системными и сетевыми службами мы будем заниматься далее на протяжении почти всей этой книги.

Однако сначала нам надо научиться отслеживать события, возникающие при работе служб и использовать их в приложениях. В следующей главе мы будем изучать компонент Broadcast Receiver, с помощью которого сможем перехватывать различные события, генерируемые системой или приложениями.



ГЛАВА 25

Broadcast Receiver

Broadcast Receiver — это компонент для отслеживания внешних событий и реакции на них. Инициализировать события могут другие приложения, пользовательские или системные службы, которые рассылают эти события в виде объектов Intent.

Класс *BroadcastReceiver*

Класс `BroadcastReceiver` является базовым для класса, в котором должны происходить получение и обработка Intent, посылаемых клиентским приложением с помощью вызова метода `sendBroadcast()`.

Для объекта `BroadcastReceiver` нет никаких возможностей видеть или фиксировать объекты Intent, используемые в методе `startActivity()`. Аналогично, когда вы передали Intent для запуска Activity через объект `BroadcastReceiver`, вы не сможете найти или запустить требуемый Activity. Эти две операции семантически полностью различаются: запуск Activity через Intent является приоритетной операцией для системы, изменяющей содержимое экрана устройства, с которым в настоящее время взаимодействует пользователь. В то же время передача системных оповещений с помощью объектов Intent является фоновой работой, о которой обычно не знает пользователь и которая, соответственно, имеет более низкий приоритет. Эти компоненты могут также реагировать на анонимные сообщения между компонентами через вызов метода `sendBroadcast()`.

Класс `BroadcastReceiver` имеет единственный метод обратного вызова:

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

Когда сообщение прибывает к получателю, система Android вызывает его методом `onReceive()` и передает в него объект Intent, содержащий сообщение. Компонент Broadcast Receiver является активным только во время выполнения этого метода. Процесс, который в настоящее время выполняет `BroadcastReceiver`, т. е. выполняющийся в настоящее время код в методе обратного вызова `onReceive()`, как полагает система Android, является приоритетным процессом и будет сохранен, кроме случаев критического недостатка памяти в системе.

Когда программа возвращается из `onReceive()`, объект `Broadcast Receiver` становится неактивным и система полагает, что работа объекта `BroadcastReceiver` закончена. Про-

цесс с активным получателем защищен от уничтожения системой. Однако процесс, содержащий неактивные компоненты, может быть уничтожен системой в любое время, когда память, которую он потребляет, будет необходима другим процессам.

Это представляет проблему, когда ответ на сообщение занимает длительное время. Если метод `onReceive()` порождает отдельный поток, а затем возвращает управление, то полный процесс, включая и порожденный поток, система Android считает неактивным (если другие компоненты приложения не активны в процессе), а также считает этот процесс кандидатом на уничтожение.

В частности, вы не можете отобразить диалог или осуществить связывание со службой внутри экземпляра `BroadcastReceiver`. Для первого случая необходимо вместо этого использовать методы класса `NotificationManager`. Во втором случае можно использовать вызов метода `Context.startService()`, чтобы послать команду для запуска службы.

Решение этой проблемы возможно, если запустить в методе `onReceive()` отдельную службу вместе с `BroadcastReceiver` и позволить службе выполнять задание, чтобы сохранить содержание процесса активным в течение всего времени вашей операции.

Фактически прием событий через объект `Intent` весьма прост в реализации. В приложении необходимо создать объект `Intent`, который вы хотите передать, и использовать вызов метода `sendBroadcast()`, чтобы послать этот объект. Установите поля `action`, `data` и `category` (действие, данные и категорию) для вашего объекта `Intent` и путь, который позволяет приемнику — компоненту `Broadcast Receiver` — точно определить "свой" объект `Intent`.

В экземпляре `Intent` строка `action` используется, чтобы идентифицировать передаваемое действие, таким образом, это должна быть уникальная строка-идентификатор действия. В соответствии с соглашением строки `action` создаются, используя те же правила именования, как и правила именования пакетов Java. Например, при создании объекта `Intent` для взаимодействия с медиаплеером (см. пример службы в предыдущей главе) можем объявить идентификатор:

```
private static String ACTION = "com.samples.app.media.PLAYER";
```

В коде, посылающем `Intent` с использованием идентификатора действия `ACTION` и с включением дополнительной информации, необходимо создать объект `Intent`, загрузить в него нужную информацию и вызвать метод `sendBroadcast()`, передав ему в качестве параметра созданный объект `Intent`:

```
private static final String TYPE = "type";
private static final int ID_ACTION_PLAY = 0;
...
Intent intent = new Intent(ACTION);
intent.putExtra(TYPE, ID_ACTION_PLAY);
sendBroadcast(intent);
```

Далее надо создать и зарегистрировать компонент `Broadcast Receiver` для приема этого объекта `Intent`.

Прослушивание событий компонентом Broadcast Receiver

Чтобы создать и использовать компонент Broadcast Receiver, его необходимо зарегистрировать в манифесте приложения, так же как мы до этого регистрировали дополнительные Activity или Service. Регистрируя компонент Broadcast Receiver, вы должны использовать Intent-фильтр, чтобы определить, какие из генерируемых объектов Intent, компонент Broadcast Receiver должен прослушивать. Для этого надо в элемент `<application>` добавить дочерний элемент `<receiver>`, определяющий имя класса Broadcast Receiver для его регистрации.

Элемент `<receiver>` должен также включать фильтр Intent `<intent-filter>`, который определяет строку действия. Например, для регистрации компонента Broadcast Receiver для взаимодействия с медиаплеером (см. предыдущую главу) файл манифеста будет иметь следующий вид:

```
<application
...
  <receiver android:name=".PlayerReceiver">
    <intent-filter>
      <action android:name="com.samples.app.media.PLAYER" />
    </intent-filter>
  </receiver>
...
</application>
```

Некоторые оповещения Broadcast Intent посылаются системой с большой частотой, например событие таймера. Другие объекты Broadcast Intent, которые, например, генерируются при изменении уровня сигнала сетевого соединения, могут изменять частоту своих рассылок в зависимости от внешних условий. При неустойчивой связи или быстрых перемещениях мобильного телефона вместе с пользователем частота генерации таких оповещений будет намного больше, чем при неподвижном положении мобильного телефона.

Поэтому при использовании объектов Broadcast Intent в приложениях желательно устанавливать фильтры, чтобы не получать оповещений, которые не требуются для работы вашего приложения. Мы их также будем применять, в случае необходимости, при создании приложений в этой книге.

Чтобы создать новый Broadcast Receiver в коде приложения, необходимо создать класс, расширяющий базовый класс `BroadcastReceiver`, и реализовать метод обратного вызова `onReceive()` обработчика событий, как показано в примере:

```
public class PlayerReceiver extends BroadcastReceiver{
    private static final String TYPE = "type";
    private static final int ID_ACTION_PLAY = 0;
    private static final int ID_ACTION_STOP = 1;

    @Override
    public void onReceive (Context context, Intent intent) {
        int type = intent.getIntExtra(TYPE, ID_ACTION_STOP);
```

```

switch (type) {
case ID_ACTION_PLAY:
    // Выполнение полученного Intent
    context.startService(new Intent (context, PlayService.class));
    break;
}
}
}

```

Метод `onReceive()` будет выполнен при получении Broadcast Intent (объекта Intent для общего оповещения в пределах системы). Приложения с зарегистрированными компонентами Broadcast Receiver будут запущены автоматически при получении соответствующего Intent.

Пример приложения с Broadcast Receiver

Для создания приложения с Broadcast Receiver возьмем за основу приложение, запускающее службу фонового воспроизведения музыки, созданную в предыдущей главе, удалив из него класс `LaunchActivity`, который будет вынесен в отдельное приложение, описанное далее в этой главе. Создайте в Eclipse новый проект без определения главной деятельности приложения:

- ❑ **Project name** — `BroadcastReceiver`;
- ❑ **Application name** — `Music Launcher Sample`;
- ❑ **Package name** — `com.samples.app.broadcastreceiver`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch25_BroadcastReceiver`.

Добавьте в файл манифеста регистрацию Broadcast Receiver с именем `PlayerReceiver` с фильтром Intent и службы `PlayService`, как показано в листинге 25.1.

Листинг 25.1. Файл `AndroidManifest.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.app.broadcastreceiver"
    android:versionCode="1"
    android:versionName="1.0">
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <service
            android:enabled="true"
            android:name=".PlayService">
        </service>
        <receiver android:name=".PlayerReceiver">
            <intent-filter>
                <action android:name="com.samples.app.media.PLAYER" />
            </intent-filter>
        </receiver>
    </application>
</manifest>

```



```
</intent-filter>
</receiver>
</application>
<uses-sdk android:minSdkVersion="3" />

</manifest>
```

Создайте новый класс, расширяющий класс `BroadcastReceiver`, и напишите код, приведенный в листинге 25.2.

Листинг 25.2. Файл класса `PlayerReceiver.java`

```
package com.samples.app.broadcastreceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PlayerReceiver extends BroadcastReceiver{

    private static final String TYPE = "type";
    private static final int ID_ACTION_PLAY = 0;
    private static final int ID_ACTION_STOP = 1;

    @Override
    public void onReceive(Context context, Intent intent) {

        int type = intent.getIntExtra(TYPE, ID_ACTION_STOP);
        switch (type) {
            case ID_ACTION_PLAY:
                Toast.makeText(context,
                    "Received action: play", Toast.LENGTH_LONG).show();
                // Запускаем службу
                context.startService(new Intent(context, PlayService.class));
                break;
            case ID_ACTION_STOP:
                Toast.makeText(context,
                    "Received action: stop", Toast.LENGTH_LONG).show();
                // Останавливаем службу
                context.stopService(new Intent(context, PlayService.class));
                break;
        }
    }
}
```

Далее добавьте в приложение класс `PlayService`, код которого приведен в главе 24, в листинге 24.1. Таким образом, мы создали приложение с компонентом `Broadcast Receiver`. Теперь необходимо создать приложение, посылающее `Intent`.

Пример приложения-передатчика события

Для приложения-передатчика также можно взять за основу приложение, запускающее службу для работы с медиаплеером. Для этого создайте в Eclipse новый проект:

- ❑ **Project name** — `BroadcastSender`;
- ❑ **Application name** — `Music Launcher Sample`;
- ❑ **Package name** — `com.samples.app.broadcastsender`;
- ❑ **Create Activity** — `LaunchActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch25_BroadcastSender`.

Файл разметки возьмите из листинга 24.3. Класс `LaunchActivity` необходимо переделать для вызова `Broadcast Intent`, используя методику, приведенную ранее в этом разделе. Полный код измененного класса `LaunchActivity` приведен в листинге 25.3.

Листинг 25.3. Файл класса окна приложения `LaunchActivity.java`

```
package com.samples.app.broadcastsender;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class LaunchActivity extends Activity
    implements View.OnClickListener {

    private static String ACTION = "com.samples.media.PLAYER";
    private static final String TYPE = "type";
    private static final int ID_ACTION_PLAY = 0;
    private static final int ID_ACTION_STOP = 1;

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

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.bStart:
                Intent iStart = new Intent(ACTION);
                iStart.putExtra(TYPE, ID_ACTION_PLAY);
                sendBroadcast(iStart);
                break;
```

```
        case R.id.bStop:
            Intent iStop = new Intent(ACTION);
            iStop.putExtra(TYPE, ID_ACTION_STOP);
            sendBroadcast(iStop);
            break;
    }
}
```

Внешний вид созданного приложения для запуска службы останется таким же, как и в *главе 24* на рис. 24.1, только теперь служба фонового воспроизведения музыки будет запускаться или останавливаться из приложения через отправку объекта Broadcast Intent.

Резюме

В данной главе мы рассмотрели создание и использование простейшего компонента Broadcast Receiver. Этот компонент используется для получения внешних событий и реакции на них. При этом источником события могут быть другие приложения или службы, как пользовательские, так и системные.

Компонент Broadcast Receiver играет важную роль при организации взаимодействия между приложениями, системными и сетевыми службами. Этой главой изучение Broadcast Receiver не заканчивается, мы будем постоянно использовать его в *части V*, когда будем изучать стандартные службы и взаимодействие этих служб с нашими приложениями.



ГЛАВА 26

Home Screen

В этой главе мы рассмотрим работу с Home Screen — домашним экраном устройства Android. Библиотеки Android предоставляют приложениям доступ к домашнему экрану — например, можно программно изменять внешний вид Home Screen, меняя его обои, и добавлять или убирать значки приложений.

Кроме того, на домашнем экране можно размещать Home Screen App Widget — виджеты для домашнего экрана. Эти виджеты представляют собой небольшие программы, видимые на домашнем экране. Это могут быть, например, часы, RSS, информация о погоде и др.

Обои для домашнего экрана

Доступ к управлению обоями домашнего экрана обеспечивает `WallpaperManager` — менеджер обоев. Чтобы получить экземпляр `WallpaperManager`, надо вызвать статический метод `getInstance()` этого класса:

```
WallpaperManager manager = WallpaperManager.getInstance(  
    getApplicationContext());
```

Экземпляр класса `WallpaperManager` позволяет получить текущие обои, узнать размер экрана для обоев, установить обои.

```
// Загружаем новые обои из ресурса  
manager.setResource(R.drawable.wallpaper);
```

Давайте посмотрим `WallpaperManager` в действии. Создайте новый проект и в мастере **Create New Project** введите следующие значения:

- ☐ **Project name** — `WallpaperManager`;
- ☐ **Application name** — `WallpaperManager`;
- ☐ **Package name** — `com.samples.homescreen.wallpapermanager`;
- ☐ **Create Activity** — `WallpaperActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch26_WallpaperManager`.

Приложение будет простым. В файле компоновки `main.xml` будет только кнопка для установки обоев, как показано в листинге 26.1.

Листинг 26.1. Файл компоновки окна приложения main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/bWallpaper"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:text="Set Wallpaper"/>
</LinearLayout>
```

В классе `WallpaperActivity` в методе `onClick()` для кнопки мы получаем экземпляр класса `WallpaperManager` и загружаем внешний ресурс — любую картинку и отображаем всплывающее сообщение об удачной загрузке обоев на рабочий стол. Полный код класса `WallpaperActivity` показан в листинге 26.2.

Листинг 26.2. Файл класса окна приложения WallpaperActivity.java

```
package com.samples.homescreen.wallpapermanager;

import java.io.IOException;

import android.app.Activity;
import android.app.WallpaperManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class WallpaperActivity extends Activity
    implements View.OnClickListener{
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public void onClick(View v) {
        WallpaperManager manager = WallpaperManager.getInstance(
            getApplicationContext());

        try {
            manager.setResource(R.drawable.background);
            Toast.makeText(this, "Wallpaper has been changed",
```

```
        Toast.LENGTH_LONG).show();  
    }  
    catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
}
```

Запустите приложение и выполните загрузку обоев для рабочего стола. Когда вы закроете приложение и вернетесь на домашний экран, вы должны увидеть новые обои на экране (рис. 26.1).

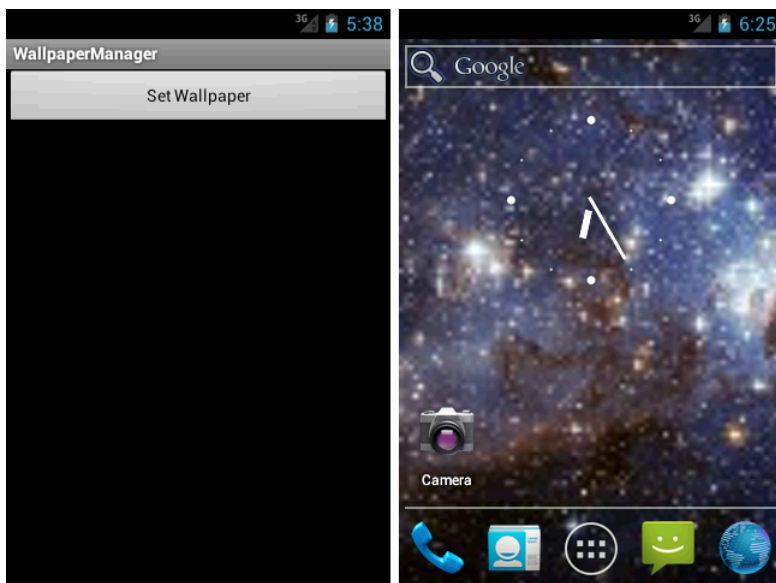


Рис. 26.1. Новые обои для домашнего экрана

Виджеты для домашнего экрана

Виджеты для домашнего экрана представляют собой миниатюрные приложения, располагающиеся на домашнем экране устройства Android. Они могут показывать время, погоду, другую полезную информацию или выполнять какие-либо действия.

Удобство их использования заключается в быстром доступе и запуске такого приложения, но, конечно, экран мобильного устройства имеет ограниченный размер, поэтому обычно на нем размещают только те приложения, которые должны отображать важную для пользователя информацию и которые он может быстро открыть.

Создание виджета

Виджет для домашнего экрана является компонентом Broadcast Receiver, т. е. приемником, реагирующим на события, возникающие в системе с предустановленным Intent-

фильтром. Но в отличие от обычного Broadcast Receiver, который мы изучали в предыдущей главе, виджет состоит из трех основных частей.

- AppWidgetProvider — класс, реализующий функциональность для перехвата событий;
- AppWidgetProviderInfo — конфигурация виджета;
- AppWidgetManager — менеджер для управления виджетом.

Чтобы создать виджет, нужно сделать последовательно довольно много действий, поэтому я распишу их по порядку:

1. Объявить виджет в файле манифеста приложения.
2. Создать объект AppWidgetProviderInfo. Данный объект будет описывать метаданные для виджета — видимый размер, частоту обновления и класс AppWidgetProvider. Эти метаданные определяются в отдельном файле XML.
3. Определить визуальную компоновку в XML-файле — точно так же, как и для обычного окна приложения.
4. Реализовать класс, расширяющий класс AppWidgetProvider, — в этом классе надо реализовать основные методы, которые позволяют системе взаимодействовать с виджетом. Все взаимодействия виджета с системой происходят через события Broadcast Intent. Через этот класс будут получены события загрузки, обновления или уничтожения виджета.
5. Дополнительно, если в этом есть необходимость, можно создать конфигурационный Activity, который будет открываться при установке на домашний экран и первом запуске виджета, и где пользователь может установить все необходимые параметры для работы виджета.

В файле манифеста приложения виджет определяется как компонент Receiver:

```
<receiver android:name=".MyAppWidgetProvider" >
    <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>
```

Элемент <receiver> требует обязательного атрибута android:name, который указывает на ваш класс, расширяющий AppWidgetProvider.

- Вложенный элемент <intent-filter> должен включать элемент <action> с обязательным атрибутом android:name. Этот атрибут указывает, что ваша реализация класса AppWidgetProvider принимает объект Broadcast Intent типа android.appwidget.action.ACTION_APPWIDGET_UPDATE.
- Элемент <meta-data> определяет XML-файл, в котором определен ресурс AppWidgetProviderInfo, и должен содержать два обязательных атрибута:
 - android:name — имя метаданных, которое обычно имеет значение android.appwidget.provider, чтобы идентифицировать данные как дескриптор типа AppWidgetProviderInfo;
 - android:resource — указывает расположение ресурса AppWidgetProviderInfo.

Объект `AppWidgetProviderInfo` определяется в XML-файле, который должен находиться в каталоге `res/xml/`. Этот файл имеет один корневой элемент `<appwidget-provider>`:

```
<appwidget-provider
    xmlns:android=http://schemas.android.com/apk/res/android ...>
    ...
</appwidget-provider>
```

`AppWidgetProviderInfo` определяет характеристики виджета, такие как его минимальные размеры расположения, его начальный ресурс расположения, частоту обновления виджета и, необязательно, конфигурационный `Activity` для первого запуска виджета. Для задания этих свойств служат следующие XML-атрибуты:

- ☐ `android:initialLayout` — указывает на ресурс для графической компоновки окна виджета;
- ☐ `android:minHeight` — минимальная по умолчанию высота виджета при добавлении его на домашний экран;
- ☐ `android:minWidth` — минимальная по умолчанию ширина виджета при добавлении его на домашний экран;
- ☐ `android:minResizeHeight` — минимальная высота (в `dp`) — это абсолютный минимум высоты виджета (он, как правило, меньше чем `minHeight`), меньше которого внешний вид виджета становится неразборчивым, и его будет трудно использовать;
- ☐ `android:minResizeWidth` — минимальная ширина, аналогично `minResizeHeight`;
- ☐ `android:resizeMode` — указывает режим изменения размеров виджета. Этот атрибут используется в случае, если требуется сделать виджет изменяемого размера, и может принимать значения:
 - `RESIZE_NONE` — фиксированный размер виджета;
 - `RESIZE_HORIZONTAL` — допускает изменения по горизонтали;
 - `RESIZE_VERTICAL` — допускает изменения по вертикали;
 - `RESIZE_BOTH` — допускает изменения по горизонтали и вертикали;
- ☐ `android:updatePeriodMillis` — время в миллисекундах, которое определяет, как часто будет происходить обновление виджета.

Сейчас мы сформируем простейший виджет и разместим его на домашнем экране мобильного устройства. Создайте новый проект и в мастере **Create New Project** введите следующие значения:

- ☐ **Project name** — `AppWidget`;
- ☐ **Application name** — `My AppWidget`;
- ☐ **Package name** — `com.samples.home.appwidget`;
- ☐ **Create Activity** — оставьте пустым.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch26_AppWidget`.

В файл манифеста приложения добавим элемент `<receiver>` с именем нашего класса, реализующего `AppWidgetProvider`. Назовем его `HomeScreenWidgetProvider` и добавим в элемент `<receiver>` фильтр событий, как показано в листинге 26.3.

Листинг 26.3. Файл манифеста приложения `AndroidManifest.xml`

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

    <uses-sdk android:minSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <receiver android:name=".HomeScreenWidgetProvider" >
            <intent-filter>
                <action
                    android:name="android.appwidget.action.APPWIDGET_UPDATE"/>
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                android:resource="@xml/hswidget" />
        </receiver>
    </application>

</manifest>
```

В файл компоновки добавим единственный виджет `TextView`, для которого зададим текст **My AppWidget** и определим дополнительно размер, цвет текста и цвет фона, как показано в листинге 26.4.

Листинг 26.4. Файл компоновки окна виджета `main.xml`

```
<?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:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="My AppWidget"
        android:textSize="18sp"
        android:textColor="#000000"
        android:textStyle="bold"/>

</LinearLayout>
```

В каталог `res/` добавим подкаталог `xml/`, в котором создадим новый XML-файл с именем `hswidget.xml`. В этот файл добавим конфигурацию виджета, как показано в листинге 26.5.

Листинг 26.5. Файл конфигурации виджета `hswidget.xml`

```
<appwidget-provider
    android:minWidth="146dp"
    android:minHeight="72dp"
    android:initialLayout="@layout/main"
</appwidget-provider>
```

В этом файле мы используем только три основных атрибута, остальные нам пока не понадобятся.

Далее надо создать класс с именем `HomeScreenWidgetProvider`, расширяющий класс `AppWidgetProvider`. Тело класса пока оставим пустым (листинг 26.6).

Листинг 26.6. Файл класса `HomeScreenWidgetProvider`

```
package com.samples.home.appwidget;
import android.appwidget.AppWidgetProvider;

public class HomeScreenWidgetProvider extends AppWidgetProvider { }
```

На этом создание виджета закончено, осталось его скомпилировать и установить.

Установка виджета

После компиляции и развертывания виджета на целевом устройстве он должен появиться на закладке **WIDGETS** окна **Application Launcher**. Для его установки на домашний экран выполните Long Click ("долгое" касание — примерно 2 секунды). При этом мы автоматически перейдем на домашний экран, где мы можем выбрать свободное место для нашего виджета (рис. 26.2).

Удаление виджета

Если вы не хотите использовать виджет, его можно удалить. Для его удаления выполните действие Long Click на области виджета. При этом он поменяет цвет фона на красный и вверху (у старых версий Android — внизу) появится область **Remove** (или значок корзины, в зависимости от версии), куда надо перетащить виджет (рис. 26.3).

Работа с классом `AppWidgetProvider`

Класс `AppWidgetProvider` является производным классом от класса `BroadcastReceiver` и добавляет нужную функциональность для виджетов домашнего экрана. В классе

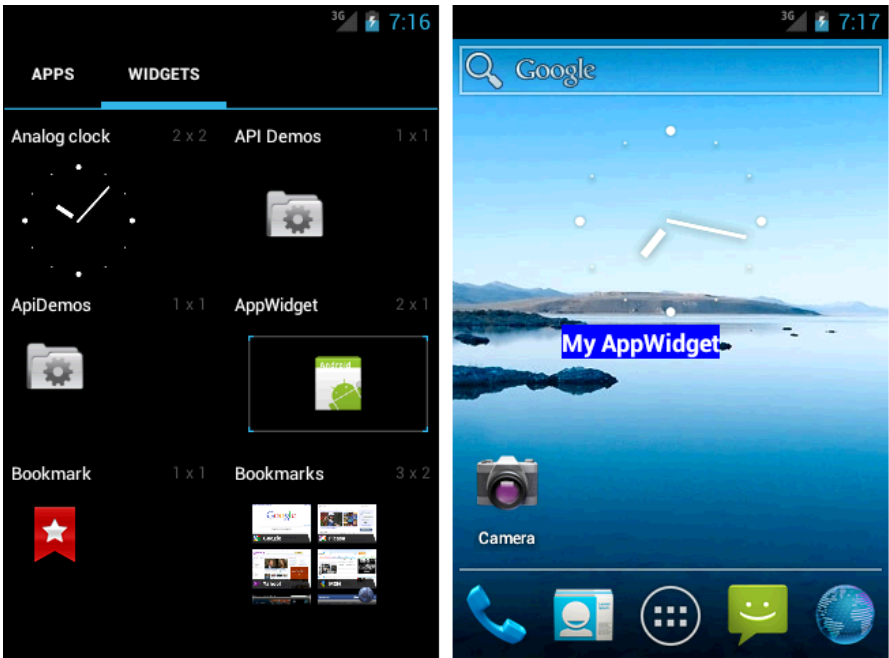


Рис. 26.2. Установка виджета на домашний экран

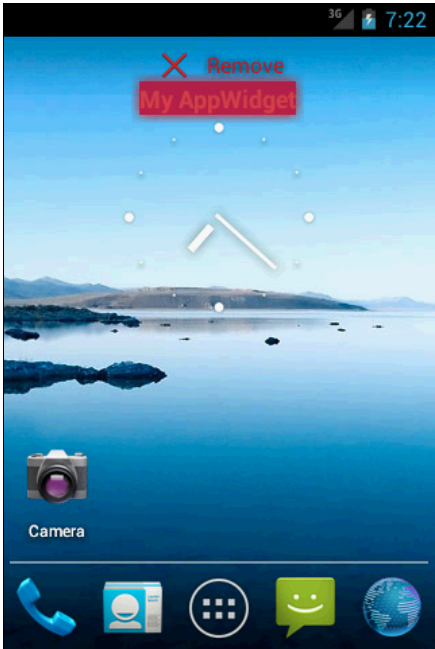


Рис. 26.3. Удаление виджета с домашнего экрана

`AppWidgetProvider` объявлена группа методов жизненного цикла виджета, которые надо будет переопределить в вашем классе, производном от `AppWidgetProvider`:

- ❑ `onEnabled()` — вызывается, когда экземпляр виджета создается впервые, если пользователь добавляет экземпляр виджета на домашний экран. Этот метод вызывается только один раз;
- ❑ `onUpdate()` — вызывается, чтобы обновить виджет в заданный интервал времени, который определяется атрибутом `updatePeriodMillis` в `AppWidgetProviderInfo`. Этот метод также вызывается при добавлении виджета на домашний экран;
- ❑ `onDeleted()` — вызывается каждый раз, когда виджет стирается из репозитория виджетов;
- ❑ `onDisabled()` — вызывается, когда уничтожается последний экземпляр виджета из системного репозитория виджетов *App Widget Host*;
- ❑ `onReceive()` — вызывается при получении `Broadcast Intent` и перед каждым из перечисленных ранее методов обратного вызова. Обычно в реализации этого метода нет необходимости, потому что при реализации класса, расширяющего класс `AppWidgetProvider`, уже по умолчанию фильтруются все генерируемые в системе объекты `Broadcast Intent`.

Самый важный из методов класса `AppWidgetProvider` — метод `onUpdate()`. В теле этого метода необходимо определить все действия, которые происходят при получении события обновления (например, от системного таймера) — загрузить в виджет новую информацию, перерисовать внешний вид виджета и т. д.

Сейчас мы создадим более функциональный виджет, отображающий системное время, который будет обновляться через заданный промежуток времени. Создайте новый проект и в мастере **Create New Project** введите следующие значения:

- ❑ **Project name** — `AppWidgetDateTime`;
- ❑ **Application name** — `AppWidgetDateTime`;
- ❑ **Package name** — `com.samples.home.hsdatetime`;
- ❑ **Create Activity** — оставьте пустым.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch26_AppWidgetDateTime`.

В файле манифеста приложения объявите компонент `Receiver` аналогично листингу 26.3. Имя класса виджета будет таким же — `HomeScreenWidgetProvider`.

Для компоновки и конфигурации виджета используйте созданные в предыдущем приложении файлы.

В файле компоновки `main.xml` из листинга 26.4 сделайте только два небольших изменения — добавьте в файл компоновки для элемента `TextView` дополнительный атрибут — идентификатор:

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

Этот идентификатор нам понадобится для использования текстового поля в классе `HomeScreenWidgetProvider`.

Второе изменение надо будет внести в файл конфигурации виджета hswidget.xml из листинга 26.5 — добавить атрибут для обновления виджета:

```
android:updatePeriodMillis="60000"
```

Таким образом виджет будет обновляться каждую минуту.

В классе `HomeScreenWidgetProvider` переопределим все методы жизненного цикла, добавив в них вызов всплывающего сообщения для того, чтобы мы могли отследить все события, происходящие с виджетом, начиная с его момента установки на домашний экран. В теле метода `onUpdate()` мы будем обновлять время, отображаемое виджетом.

Код класса `HomeScreenWidgetProvider` представлен в листинге 26.7.

Листинг 26.7. Файл класса виджета `HomeScreenWidgetProvider.java`

```
package com.samples.home.hsdatetime;

import java.text.SimpleDateFormat;
import java.util.Date;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.Context;
import android.widget.RemoteViews;
import android.widget.Toast;

public class HomeScreenWidgetProvider extends AppWidgetProvider {
    private SimpleDateFormat dateFormat =
        new SimpleDateFormat("dd MMM yyyy hh:mm a");

    @Override
    public void onDeleted(Context context, int[] appWidgetIds) {
        super.onDeleted(context, appWidgetIds);
        Toast.makeText(context, "Delete", Toast.LENGTH_LONG).show();
    }

    @Override
    public void onDisabled(Context context) {
        super.onDisabled(context);
        Toast.makeText(context, "Disable", Toast.LENGTH_LONG).show();
    }

    @Override
    public void onEnabled(Context context) {
        super.onEnabled(context);
        Toast.makeText(context, "Enable", Toast.LENGTH_LONG).show();
    }

    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds) {
```

```
super.onUpdate(context, appWidgetManager, appWidgetIds);
String currentTime = dateFormat.format(new Date());

RemoteViews updateViews = new RemoteViews(context.getPackageName(),
    R.layout.main);
updateViews.setTextViewText(R.id.text, currentTime);
appWidgetManager.updateAppWidget(appWidgetIds, updateViews);
Toast.makeText(context, "Update", Toast.LENGTH_LONG).show();
}
}
```

Скомпилируйте и запустите приложение на выполнение. Поскольку мы определили в тексте программы в теле каждого метода всплывающие сообщения, вы можете наблюдать всю последовательность событий, происходящих при установке или удалении виджета на домашний экран.

После установки виджета он будет периодически обновлять отображаемое время, а в момент обновления будет также показываться всплывающее сообщение (рис. 26.4).

Конечно, можно приспособить этот виджет для выполнения более полезных функций, например, показывать погоду, периодически обновляемую из Интернета, или при получении SMS выводить на экран устройства текст сообщения и информацию об отправителе. Вы сможете реализовать такие виджеты, когда мы начнем изучать сетевые сервисы в *части V* книги.

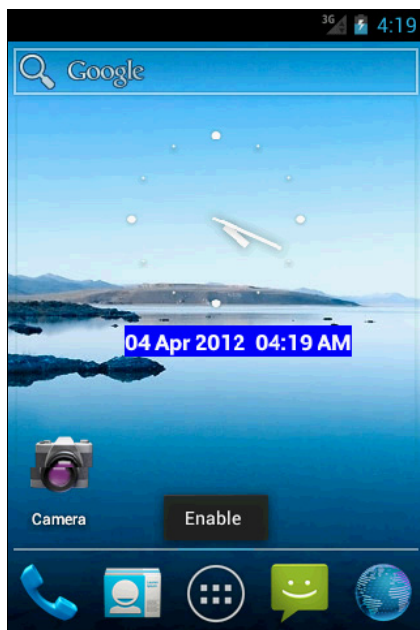


Рис. 26.4. Виджет с временным обновлением

Резюме

В этой главе мы рассмотрели использование домашнего экрана устройства Android — программную замену обоев на домашнем экране и создание виджетов — маленьких программ, отображаемых на экране, которые предназначены для постоянного отображения пользователю различной актуальной информации.

В следующей главе мы изучим работу со Status Bar (строкой состояния) устройства Android, генерирование и отображение уведомлений в строке состояния для интерактивного взаимодействия с пользователем.



ГЛАВА 27

Уведомления в строке состояния

В этой главе описывается использование системной службы Notification Service для создания приложений, уведомляющих о наступлении в системе или в работающем приложении какого-либо события, требующего реакции пользователя мобильного устройства.

Это уведомление отображается в строке состояния (Status Bar) мобильного устройства. Для привлечения внимания пользователя менеджер уведомлений позволяет использовать также звуковые сигналы, виброзвонок, светодиодный индикатор и подсветку экрана.

Пользовательские уведомления генерируются системой при возникновении какого-либо события, например получение SMS, окончание загрузки файла и др. В программном коде можно также генерировать собственные уведомления.

Менеджер уведомлений

В системе Android пользовательскими уведомлениями управляет системная служба Notification Service. В коде приложения эта служба доступна через экземпляр класса NotificationManager. Объект класса NotificationManager создается стандартным способом, через вызов метода getSystemService() с параметром Context.NOTIFICATION_SERVICE:

```
NotificationManager manager =  
(NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
```

Уведомления могут отображаться несколькими способами:

- ☐ появление значка в строке состояния;
- ☐ включение светодиода на устройстве;
- ☐ включение подсветки дисплея;
- ☐ включение заданного звукового сигнала;
- ☐ включение виброзвонка.

Для управления уведомлениями в классе NotificationManager определена группа методов:

- ☐ notify() — отображает оповещение в строке состояния мобильного устройства;
- ☐ cancel() — закрывает показанное ранее оповещение;
- ☐ cancelAll() — закрывает все показанные ранее оповещения.

Каждый из этих методов, кроме `cancelAll()`, принимает входной параметр `id`, который является идентификатором уведомления, и необязательный параметр `tag` типа `String`. Эта пара параметров идентифицирует уведомление, генерируемое приложением, и должна была быть уникальна для приложения. Если приложение сгенерирует новое уведомление с таким же идентификатором, это уведомление перепишет ранее сгенерированное.

Кроме этих параметров, для метода `notify()`, конечно, передается само уведомление, представленное объектом класса `Notification`, которое мы сейчас рассмотрим более подробно.

Создание уведомления

Структура уведомления создается с помощью объекта класса `Notification`, который затем передается в метод `notify()` класса `NotificationManager`.

Первым шагом в создании уведомления является создание объекта `Notification` и заполнение полей этого класса для настройки уведомления.

Класс `Notification` содержит различные поля, которые используются при создании уведомления для настройки различных режимов его отображения:

- ❑ `icon` — значок, который будет отображаться в строке состояния;
- ❑ `sound` — звук при появлении уведомления;
- ❑ `tickerText` — текст, который "прокручивается" в строке состояния при появлении уведомления;
- ❑ `when` — время появления уведомления.

При появлении уведомления можно также управлять режимом работы светодиодного индикатора мобильного устройства. Для этого в классе `Notification` определены три поля:

- ❑ `ledARGB` — определяет цвет светодиода;
- ❑ `ledOnMS` — определяет время нахождения светодиода во включенном состоянии в миллисекундах;
- ❑ `ledOffMS` — определяет время нахождения светодиода в погашенном состоянии в миллисекундах.

В классе `Notification` есть поле `default` для установки режима отображения уведомления по умолчанию. Это поле состоит из набора битовых флагов для установки режима отображения уведомления по умолчанию:

- ❑ `DEFAULT_LIGHTS` — подсветка по умолчанию;
- ❑ `DEFAULT_SOUND` — звуковой сигнал по умолчанию;
- ❑ `DEFAULT_VIBRATE` — вибрация по умолчанию;
- ❑ `DEFAULT_ALL` — используются все значения по умолчанию.

Поля класса `Notification` можно использовать для установки режимов поведения уведомления при взаимодействии с ним пользователя. Например, в этом классе определе-

но поле `flags`, которое использует сочетание одного или нескольких флагов для настройки режимов поведения:

- ❑ `FLAG_AUTO_CANCEL` — флаг, означающий, что уведомление должно быть закрыто после просмотра его пользователем;
- ❑ `FLAG_FOREGROUND_SERVICE` — флаг, показывающий, что уведомление представляет службу, работающую в текущем процессе;
- ❑ `FLAG_HIGH_PRIORITY` — устанавливает высокий приоритет для уведомления, которое требуется немедленно показать пользователю, даже если строка состояния скрыта;
- ❑ `FLAG_INSISTENT` — флаг, устанавливающий режим периодического повторения звукового сигнала до тех пор, пока уведомление не будет сброшено или окно уведомления не будет открыто пользователем;
- ❑ `FLAG_NO_CLEAR` — флаг, означающий, что уведомление не сбрасывается при нажатии пользователем кнопки **Clear**;
- ❑ `FLAG_ONGOING_EVENT` — флаг, означающий, что данное уведомление ссылается на продолжающееся только в данный момент событие (например, когда пользователь звонит по телефону, в строке состояния мобильного устройства отображается значок телефонной трубки, а после окончания звонка он сбрасывается);
- ❑ `FLAG_ONLY_ALERT_ONCE` — флаг, устанавливающий режим оповещения только с помощью звука или вибровзвонка при генерации уведомления;
- ❑ `FLAG_SHOW_LIGHTS` — флаг, устанавливающий режим включения светодиода при генерации уведомления.

С помощью этой функциональности можно гибко настраивать уведомления, используя помимо текста различные значки, звуковые сигналы и подсветку экрана мобильного устройства. Чтобы создать простое уведомление со значком и текстом в строке состояния, можно выполнить следующий код:

```
Notification notification = new Notification();
// Устанавливаем значок
notification.icon = android.R.drawable.ic_notification_overlay;
// Устанавливаем текст уведомления
notification.tickerText = "New Notification";
// Устанавливаем время появления уведомления
notification.when = System.currentTimeMillis();
// Задаем режим поведения уведомления
notification.flags =
    Notification.FLAG_SHOW_LIGHTS | Notification.FLAG_AUTO_CANCEL;
```

Следующим шагом является формирование представления уведомления, которое будет отображаться пользователю, когда он разворачивает уведомление на экране мобильного устройства. Для этого в классе `Notification` определен метод `setLatestEventInfo()`, в который надо передать четыре входных параметра:

- ❑ объект `Context` приложения или `Activity`;
- ❑ строку с заголовком уведомления;
- ❑ строку с текстом уведомления;

- ❑ объект `PendingIntent`, который запустится, когда пользователь выберет уведомление в строке состояния и развернет его на экране.

В коде программы сформировать представление для уведомления можно так:

```
Intent intent = new Intent(
    getApplicationContext(), NotificationActivity.class);
PendingIntent pendIntent = PendingIntent.getActivity(
    getApplicationContext(), 0, intent, 0);
```

```
notification.setLatestEventInfo(this, "Notification!",
    "Notification from app", pendIntent);
```

Последним шагом является вызов уведомления. Для управления отображением уведомления используется метод `notify()` класса `NotificationManager`. В этот метод надо передать идентификатор уведомления, который должен быть уникальным в пределах приложения, и сам объект класса `Notification`:

```
NotificationManager manager = (NotificationManager) getSystemService(
    Context.NOTIFICATION_SERVICE);
manager.notify(1, notification);
```

Теперь создадим пример приложения, генерирующего уведомление в строке состояния. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `NotificationManager`;
- ❑ **Application name** — `Notification`;
- ❑ **Package name** — `com.samples.notificationmanager`;
- ❑ **Create Activity** — `NotificationActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch27_NotificationManager`.

В файле компоновки главного окна приложения `main.xml` будут расположены две командные кнопки для управления уведомлением:

- ❑ **Create Notification** с идентификатором `bCreate` для генерации уведомления;
- ❑ **Clear Notification** с идентификатором `bClear` для сброса уведомления.

Код файла `main.xml` представлен в листинге 27.1.

Листинг 27.1. Файл компоновки окна приложения `main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <Button
        android:id="@+id/bCreate"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Create Notification"
        android:onClick="onClick"/>
<Button
    android:id="@+id/bClear"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Clear Notification"
    android:onClick="onClick"/>

</LinearLayout>
```

В классе `NotificationActivity` мы создадим простое уведомление, используя объект `Notification` и действуя по описанным ранее в этом разделе шагам. Полный код класса `NotificationActivity` представлен в листинге 27.2.

Листинг 27.2. Файл класса главного окна приложения `NotificationActivity.java`

```
package com.samples.notificationmanager;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class NotificationActivity extends Activity
    implements View.OnClickListener {
    // Идентификатор уведомления
    private static final int ID_NOTIFY = 101;

    private NotificationManager manager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
```

```
case R.id.bCreate:
    // Получаем экземпляр менеджера уведомлений
    manager = (NotificationManager) getSystemService(
        Context.NOTIFICATION_SERVICE);

    // Задаем параметры отображения
    Notification notification = new Notification();
    notification.icon = android.R.drawable.ic_notification_overlay;
    notification.tickerText = "New Notification";
    notification.when = System.currentTimeMillis();

    // Создаем представление для отображения уведомления
    // в развернутом виде
    Intent intent = new Intent(this, NotificationActivity.class);

    PendingIntent pendIntent = PendingIntent.getActivity(
        getApplicationContext(), 0, intent, 0);
    notification.setLatestEventInfo(this, "Notification!",
        "Notification from app", pendIntent);

    // Отображаем уведомление
    manager.notify(ID_NOTIFY, notification);
    break;

case R.id.bClear:
    // Закрываем уведомление
    manager.cancel(ID_NOTIFY);
    break;
}
}
```

Выполните компиляцию проекта и запустите его на эмуляторе Android. При нажатии на кнопку **Create Notification** приложение создаст уведомление со значком в виде красного кружка и заголовком "New Notification", отображаемыми в строке состояния. Это уведомление можно или открыть и сбросить нажатием кнопки **Clear** на панели уведомления, или просто сбросить кнопкой **Clear Notification** в окне приложения.

Внешний вид приложения представлен на рис. 27.1.

Это приложение было создано как основа для создания и управления уведомлениями из приложений. Если у вас есть желание, поэкспериментируйте с этим приложением, используя различные режимы отображения и управления уведомлениями самостоятельно. Эти навыки вам пригодятся впоследствии при создании приложений, предполагающих интерактивное взаимодействие с пользователем.

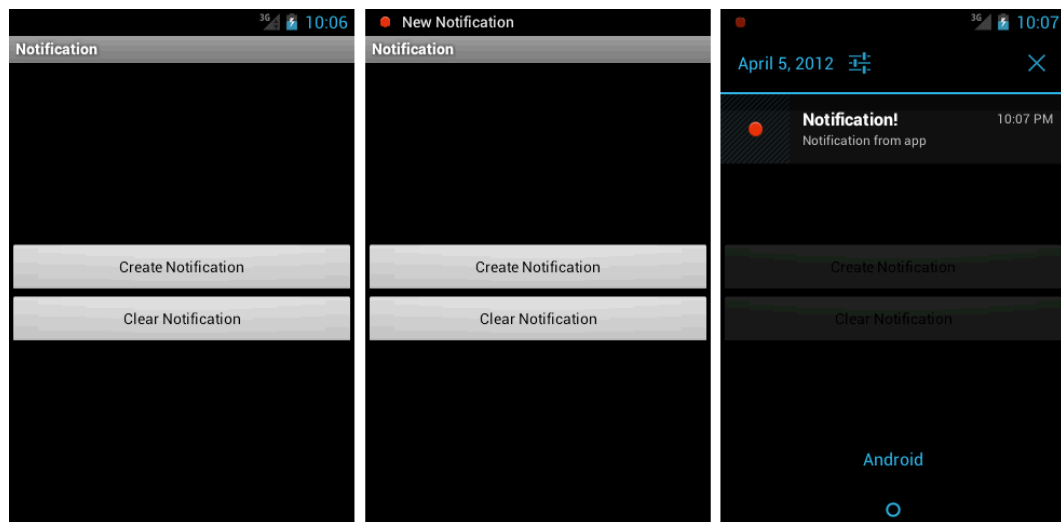


Рис. 27.1. Приложение, создающее уведомления в строке состояния

Резюме

В этой главе мы рассмотрели управление пользовательскими уведомлениями из приложений. Использование службы Notification Service позволяет создавать приложения для интерактивного взаимодействия с пользователем мобильного устройства.

В следующей главе мы рассмотрим работу с Action Bar — новым элементом пользовательского интерфейса для планшетных ПК.



ГЛАВА 28

Action Bar

Action Bar (строка действий) — это новая функциональность, появившаяся в Android 3.0. Вместо традиционной строки заголовка окна, расположенной наверху экрана устройства, как было в версиях до Android 3.0, Action Bar выводит на экран значок приложения вместе с текстовым заголовком. Кроме этого, Action Bar позволяет выводить опции меню (если оно реализовано в приложении) с текстом и значками (которая выглядит почти как строка главного меню в приложениях для настольных систем), что упрощает работу с приложением.

Action Bar идентифицирует приложение и обеспечивает более удобную навигацию. Этот элемент можно отключать, если в нем нет необходимости (Action Bar занимает гораздо больше места, чем традиционная строка заголовка, поэтому на устройствах с маленьким экраном его использование нецелесообразно).

Управление видимостью Action Bar

Самый простой способ управления видимостью Action Bar вы уже видели в *главе 3* — это использование элемента `<uses-sdk>`. Если задать значение атрибута, определяющего минимальную версию SDK `android:minSdkVersion` от 10 и меньше (это ниже версии 3.0, первой версии, предназначенной для планшетных ПК), или совсем удалить этот атрибут из файла манифеста, Action Bar станет невидимым. Правда, это неофициальный способ, в принципе, если программа с таким значением `android:minSdkVersion` будет запускаться на устройстве с API Level 11-15, Action Bar должен отображаться, скорей всего, такое поведение ошибочно и будет исправлено в будущем.

Еще один простой способ — использование в файле манифеста в элементе `<activity>` атрибута `android:theme`. Чтобы отключить Action Bar, надо установить стиль отображения в значение `Theme.Holo.NoActionBar`.

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

Управлять видимостью Action Bar можно и в коде программы. Получить ссылку на Action Bar во время выполнения можно вызовом метода `getActionBar()`:

```
ActionBar actionBar = getActionBar();
```

Для скрытия Action Bar используется метод `hide()`, для отображения — `show()`. Однако, если в манифесте приложения используется атрибут `android:theme` для отключения отображения Action Bar, вызов метода `getActionBar()` возвращает нуль во время выполнения программы (если его не обработать, будет сгенерировано исключение). Поэтому все-таки, если вы хотите программно управлять видимостью Action Bar, гораздо удобнее не использовать атрибут `android:theme` в файле манифеста приложения, а создавать экземпляр класса `ActionBar` в программе.

Создайте новый проект и в мастере **Create New Project** введите следующие значения:

- ☐ **Project name** — `ShowHideActionBar`;
- ☐ **Application name** — `ShowHideActionBar`;
- ☐ **Package name** — `com.samples.home.showhideactionbar`;
- ☐ **Create Activity** — `ShowHideActionBarActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch28_ShowHideActionBar`.

В файле компоновки окна приложения добавим две кнопки с надписями **Hide Action Bar** и **Show Action Bar**, как показано в листинге 28.1.

Листинг 28.1. Файл компоновки окна приложения `main.xml`

```
<?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" >
    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <Button
            android:id="@+id/bHide"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="Hide Action Bar" />
        <Button
            android:id="@+id/bShow"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="onClick"
            android:text="Show Action Bar" />
    </LinearLayout>
</LinearLayout>
```

В классе `ShowHideActionBarActivity` в теле метода `onCreate()` мы получим ссылку на объект `Action Bar` с помощью вызова метода `getActionBar()`, а в обработчике нажатия кнопок добавим методы для управления видимостью `Action Bar`. Код класса `ShowHideActionBarActivity` представлен в листинге 28.2.

Листинг 28.2. Файл класса главного окна приложения `ShowHideActionBarActivity.java`

```
package com.samples.home.showhideactionbar;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

public class ShowHideActionBarActivity extends Activity
    implements OnClickListener {
    private ActionBar actionBar;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        actionBar = getActionBar();
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.bHide:
                actionBar.hide();
                break;
            case R.id.bShow:
                actionBar.show();
                break;
        }
    }
}
```

Скомпилируйте и запустите приложение на выполнение. С помощью кнопки вы теперь можете управлять видимостью `Action Bar` так же, как и на рис. 28.1.

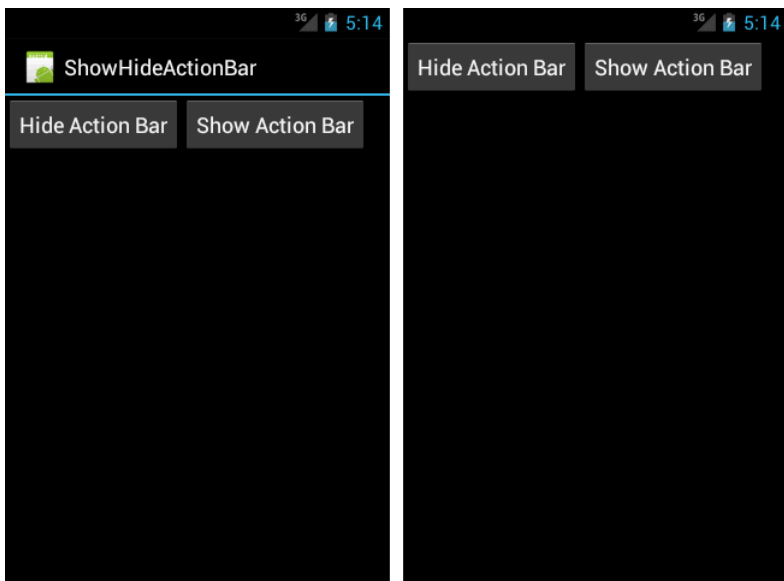


Рис. 28.1. Управление видимостью Action Bar

Добавление опций меню в Action Bar

Помимо отображения значка приложения и заголовка действия слева Action Bar, можно также вывести на Action Bar дополнительные элементы — пункты меню в виде текста или значков (примерно как значки на панели инструментов в программах для настольных систем).

Если для этого Activity определено обычное меню выбора опций (Options Menu), то для того, чтобы заставить пункт меню появиться в области Action Bar, при создании меню для каждого выводимого пункта надо добавить вызов метода `setShowAsAction()`, передав ему в качестве параметра константу `SHOW_AS_ACTION_IF_ROOM`, определенную в классе `MenuItem`:

```
private void CreateMenu(Menu menu)
{
    menu.add(Menu.NONE, IDM_OPEN, 1, "Open")
        .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
    ...
    return (super.onCreateOptionsMenu(menu));
}
```

Метод `setShowAsAction()` дает указание системе Android выводить на экран пункт меню в правую область Action Bar, но при условии, что есть место для его отображения. На разных размерах экрана будет выведено различное количество опций меню.

Давайте сейчас создадим тестовое приложение и посмотрим его поведение при разных положениях экрана. Создайте новый проект и в мастере **Create New Project** введите следующие значения:

- ❑ **Project name** — ActionBarItems;
- ❑ **Application name** — Action Bar Items;
- ❑ **Package name** — com.samples.homescreen.actionbaritems;
- ❑ **Create Activity** — ActionBarItemsActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch28_ActionBarItems.

В классе, реализующем Activity, создадим меню из 5 пунктов, аналогично тому, как вы уже создавали меню в *главе 11*, добавив к каждому пункту по вызову `setShowAsAction()`, как представлено в листинге 28.3.

Листинг 28.3. Файл класса главного окна приложения ActionBarItemsActivity.java

```
package com.samples.homescreen.actionbaritems;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class ActionBarItemsActivity extends Activity {
    private static final int IDM_OPEN = 1001;
    private static final int IDM_SAVE = 1002;
    private static final int IDM_EDIT = 1003;
    private static final int IDM_HELP = 1004;
    private static final int IDM_EXIT = 1005;

    private ActionBar actionBar;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        actionBar = this.getActionBar();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        menu.add(Menu.NONE, IDM_OPEN, 1, "Open")
            .setIcon(R.drawable.ic_menu_open)
            .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        menu.add(Menu.NONE, IDM_SAVE, 2, "Save")
            .setIcon(R.drawable.ic_menu_save)
            .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
```

```

        menu.add(Menu.NONE, IDM_EDIT, 3, "Edit")
            .setIcon(R.drawable.ic_menu_edit)
            .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        menu.add(Menu.NONE, IDM_HELP, 4, "Help")
            .setIcon(R.drawable.ic_menu_help)
            .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);
        menu.add(Menu.NONE, IDM_EXIT, 5, "Exit")
            .setIcon(R.drawable.ic_menu_exit)
            .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM);

        return (super.onCreateOptionsMenu(menu));
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        CharSequence message = "";
        switch (item.getItemId()) {
            case IDM_OPEN:
                message = "Open item selected";
                break;
            case IDM_SAVE:
                message = "Save item selected";
                break;
            case IDM_HELP:
                message = "Help item selected";
                break;
            case IDM_EDIT:
                message = "Edit item selected";
                break;
            case IDM_EXIT:
                message = "Exit item selected";
                break;
            default:
                return false;
        }

        Toast toast = Toast.makeText(this, message, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
        return true;
    }
}

```

Если запустить приложение на эмуляторе с разрешением экрана HVGA и вертикальной ориентацией экрана, мы увидим, что из меню, в котором мы определили 5 пунктов, 2 первых пункта меню, **Open** и **Save**, переместились на Action Bar и отображаются в виде значков (рис. 28.2).

Если установить горизонтальную ориентацию экрана, у нас увеличится место для отображения меню, и, в зависимости от размеров экрана, система автоматически добавит еще опции меню для заполнения пустого места (рис. 28.3).

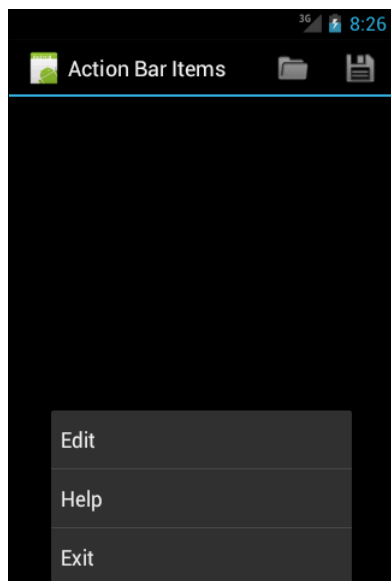


Рис. 28.2. Отображение меню в вертикальной ориентации экрана

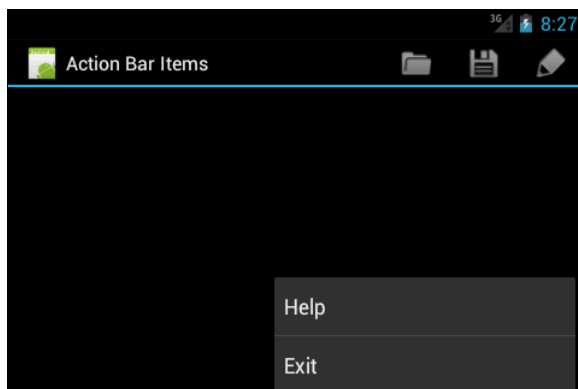


Рис. 28.3. Отображение меню в горизонтальной ориентации экрана

При достаточно большом разрешении экрана в область Action Bar могут быть выведены все пункты меню. Конечно, такое меню более удобно для работы с приложением, чем обычное, открывающееся только при нажатии кнопки **MENU** на клавиатуре мобильного устройства.

Добавление текста в меню

В предыдущем примере пункты меню выводились на экран только в виде значков, без текста. Если нет необходимости показывать значки, можно для опций меню вывести только текст. Для этого достаточно удалить из программы вызовы метода `setIcon()`. В результате внешний вид приложения будет таким, как на рис. 28.4.

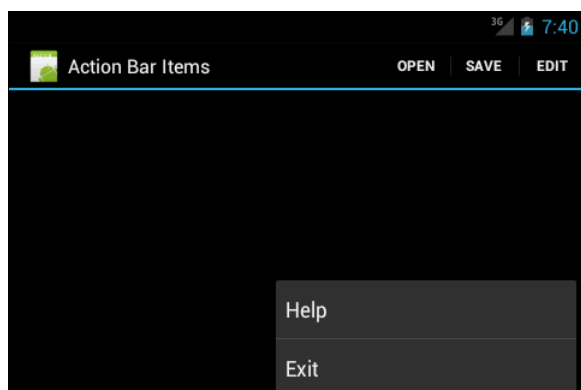


Рис. 28.4. Отображение опций меню в виде текста, без значков

Можно также вывести на экран для каждой опции текст вместе со значком. Для этого нужно использовать константу `SHOW_AS_ACTION_WITH_TEXT` в дополнение к `SHOW_AS_ACTION_IF_ROOM`, соединив их знаком "|", как показано в листинге 28.4.

Листинг 28.4. Изменение в методе `onCreateOptionsMenu()` класса `ActionBarItemsActivity`

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    menu.add(Menu.NONE, IDM_OPEN, 1, "Open")
        .setIcon(R.drawable.ic_menu_open)
        .setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_ROOM |
            MenuItem.SHOW_AS_ACTION_WITH_TEXT);
    ...
    return (super.onCreateOptionsMenu(menu));
}
```

Если теперь перекомпилировать приложение, пункты меню будут отображаться вместе с текстом, как показано на рис. 28.5.

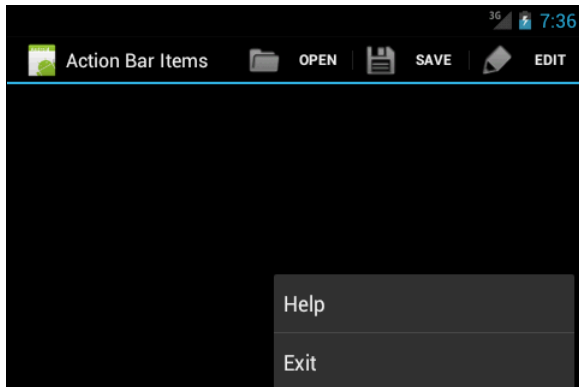


Рис. 28.5. Отображение опций меню с текстом и значками

Резюме

В этой главе мы познакомились с Action Bar, относительно новым элементом пользовательского интерфейса, появившемся в версии Android 3.0 и предназначенным в основном для более удобной навигации и работы с приложением, сопоставимой с удобством работы в настольных приложениях планшетных ПК.

В следующей главе мы будем изучать использование Alarm Manager для управления системной службой, которая может создавать настраиваемые периодические пользовательские оповещения.



ГЛАВА 29

Служба оповещений

В этой главе описывается использование системной службы Alarm Service для создания приложений, осуществляющих взаимодействие с пользователем мобильного устройства.

Службу Alarm Service можно использовать в своем приложении для отправления пользователю разовых или периодических оповещений в заданное время. На основе этой службы можно создавать различные планировщики заданий, органайзеры, будильники и таймеры, т. е. приложения для запрограммированного разового или периодического оповещения пользователя.

Менеджер оповещений

Доступ к Alarm Service и управление службой в приложении осуществляется через объект `AlarmManager`, который создается следующим образом:

```
AlarmManager manager = (AlarmManager) getSystemService(  
    Context.ALARM_SERVICE);
```

Объект `AlarmManager` предназначен для случаев, когда вам требуется создать приложение, которое будет запускаться в определенное время и выполнять какие-либо действия, даже если в этот момент приложение закрыто. С помощью класса `AlarmManager` можно создавать как разовые оповещения, так и периодические.

Класс `AlarmManager` содержит набор методов для работы с оповещениями:

- ❑ `cancel()` — удаляет все оповещения;
- ❑ `setTime()` — устанавливает системное время;
- ❑ `setTimeZone()` — устанавливает для системы временную зону по умолчанию;
- ❑ `set()` — задает оповещение;
- ❑ `setRepeating()` — задает повторяющееся оповещение со строго заданным временным периодом;
- ❑ `setInexactRepeating()` — устанавливает повторяющееся оповещение без строгого требования к точности периода повторения. С помощью этого метода можно установить оповещение, которое будет гарантированно повторяться, например, каждый час, но необязательно в начале каждого часа, а будет иметь некоторую погрешность во времени.

Методы `set()`, `setRepeating()` и `setInexactRepeating()` содержат набор параметров, определяющих режим создания оповещений:

- ☐ `typeOne` — указывает на тип используемого времени. Это может быть системное или всемирное координированное время (UTC) для запуска оповещения, которое определяется целочисленными константами в классе `AlarmManager`:
 - `ELAPSED_REALTIME` — устанавливается при использовании системного времени;
 - `ELAPSED_REALTIME_WAKEUP` — устанавливается при использовании системного времени с возможностью запуска оповещения, если мобильное устройство находится в спящем режиме;
 - `RTC` — устанавливается при использовании всемирного координированного времени (UTC);
 - `RTC_WAKEUP` — устанавливается при использовании всемирного координированного времени с возможностью запуска оповещения, если мобильное устройство находится в спящем режиме;
- ☐ `triggerAtTime` — время работы оповещения;
- ☐ `interval` — определяет интервал между отправкой повторных оповещений. Этот параметр присутствует только в методах `setRepeating()` и `setInexactRepeating()`;
- ☐ `operation` — объект `PendingIntent`, который будет определять действие, выполняемое при запуске оповещения.

Временной интервал для периодических оповещений в параметре `interval` задается в миллисекундах. Можно также воспользоваться константами, которые определяют значения для наиболее часто встречающихся интервалов:

- ☐ `INTERVAL_DAY`;
- ☐ `INTERVAL_HALF_DAY`;
- ☐ `INTERVAL_HOUR`;
- ☐ `INTERVAL_HALF_HOUR`;
- ☐ `INTERVAL_FIFTEEN_MINUTES`.

Эти константы представляют собой числовые значения, равные длине временного интервала, который они представляют, выраженного в миллисекундах. Например, для константы `INTERVAL_HOUR` значение будет равно 3 600 000.

Использование оповещений

Логика выполнения оповещения реализуется в виде службы, т. е. в классе, наследуемом от базового класса `Service`:

```
public class AlarmService extends Service { ... }
```

Запуск и управление этой службы производится с помощью объекта `Intent`, например, следующим образом:

```
public class AlarmManagerActivity extends Activity
{
    ...
}
```

```
Intent intent = new Intent(  
    AlarmManagerActivity.this, AlarmService.class);  
PendingIntent pendingIntent = PendingIntent.getService(  
    getApplicationContext(), 0, intent, 0);  
...  
}
```

Для задания времени работы оповещения необходимо установить его время запуска и добавить к нему длительность работы этого оповещения. Например, нам необходимо, чтобы оповещение отработывало 5 секунд после запуска. Это можно сделать следующим образом:

```
long time = SystemClock.elapsedRealtime() + 5000;
```

Для задания времени работы оповещения также используют объект `Calendar`. Его удобно применять при преобразовании времени. Например, нам требуется, чтобы продолжительность сигнала оповещения была 10 секунд, а период повторения оповещения был один час. Это можно сделать с помощью следующего кода:

```
Calendar calendar = Calendar.getInstance();  
calendar.setTimeInMillis(System.currentTimeMillis());  
calendar.add(Calendar.SECOND, 10);  
long time = calendar.getTimeInMillis();
```

```
manager.setRepeating(AlarmManager.RTC_WAKEUP, time,  
    AlarmManager.INTERVAL_HOUR, pendingIntent);
```

Давайте теперь создадим практическое приложение, использующее объект `AlarmManager` для создания циклических оповещений. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `AlarmManager`;
- ☐ **Application name** — `Alarm Manager`;
- ☐ **Package name** — `com.samples.alarm.alarmmanager`;
- ☐ **Create Activity** — `AlarmManagerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch29_AlarmManager`.

Для службы, которая будет управляться из класса `AlarmManagerActivity`, создайте отдельный класс и назовите его `AlarmService`.

В файл манифеста приложения `AndroidManifest.xml` добавьте элемент `<service>` с именем класса, представляющего службу. Код файла манифеста приложения представлен в листинге 29.1.

Листинг 29.1. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.samples.alarm.alarmmanager"
```

```

    android:versionCode="1"
    android:versionName="1.0">

    <application
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".AlarmManagerActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".AlarmService">
        </service>
    </application>
</manifest>

```

В файле компоновки главного окна приложения `main.xml` будут три кнопки для управления службой:

- ❑ кнопка запуска с надписью **Start** и идентификатором `bStart`;
- ❑ кнопка останова с надписью **Cancel** и идентификатором `bCancel`;
- ❑ кнопка сброса с надписью **Stop** и идентификатором `bStop`.

Код файла `main.xml` представлен в листинге 29.2.

Листинг 29.2. Файл компоновки главного окна приложения `main.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gravity="center">

    <Button
        android:id="@+id/bStart"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Start"
        android:onClick="onClick"/>

    <Button
        android:id="@+id/bCancel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:onClick="onClick"/>

```

```
<Button
    android:id="@+id/bStop"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Stop"
    android:onClick="onClick"/>
```

```
</LinearLayout>
```

В коде класса главного окна приложения `AlarmManagerActivity` реализуем принцип создания периодических оповещений, рассмотренный ранее в этом разделе. Код класса `AlarmManagerActivity` представлен в листинге 29.3.

Листинг 29.3. Файл класса главного окна приложения `AlarmManagerActivity.java`

```
package com.samples.alarm.alarmmanager;

import java.util.Calendar;

import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class AlarmManagerActivity extends Activity
    implements View.OnClickListener {

    private PendingIntent pendingIntent;
    private AlarmManager manager;
    private Intent intent;

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

        manager = (AlarmManager) getSystemService(ALARM_SERVICE);
        intent = new Intent(AlarmManagerActivity.this,
            AlarmService.class);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.bStart:
                pendingIntent = PendingIntent.getService(
                    getApplicationContext(), 0, intent, 0);
```

```

Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.add(Calendar.SECOND, 5);

manager.setRepeating(
    AlarmManager.RTC_WAKEUP,
    calendar.getTimeInMillis(),
    5000,
    pendingIntent);

break;

case R.id.btnCancel:
    manager.cancel(pendingIntent);
    break;

case R.id.bStop:
    this.stopService(intent);
    break;
}
}
}

```

В классе `AlarmService`, представляющем нашу службу, для оповещения пользователя мы будем использовать простые всплывающие уведомления (`Toast Notification`). В этом классе также определим целочисленную переменную `count` для мониторинга количества сгенерированных оповещений. В методе `onStart()` будет выдаваться периодическое сообщение для пользователя, включающее также его порядковый номер.

Также добавим уведомления во все методы, вызываемые системой. Они нам понадобятся для отслеживания жизненного цикла службы, когда будем тестировать это приложение. Код класса `AlarmService` представлен в листинге 29.4.

Листинг 29.4. Файл класса службы `AlarmService.java`

```

package com.samples.alarm.alarmmanager;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class AlarmService extends Service {
    private int count;

    @Override
    public void onCreate() {
        count = 0;
        Toast.makeText(this, "Create service",
            Toast.LENGTH_LONG).show();
    }
}

```

```
@Override
public IBinder onBind(Intent intent) {
    Toast.makeText(this, "Bind service",
        Toast.LENGTH_LONG).show();
    return null;
}

@Override
public void onDestroy() {
    super.onDestroy();
    Toast.makeText(this, "Destroy service",
        Toast.LENGTH_LONG).show();
}

@Override
public void onStart(Intent intent, int startId) {
    super.onStart(intent, startId);
    count++;
    Toast.makeText(this, "Notify from AlarmManager #" + count,
        Toast.LENGTH_LONG).show();
}

@Override
public boolean onUnbind(Intent intent) {
    Toast.makeText(this, "Unbind service",
        Toast.LENGTH_LONG).show();
    return super.onUnbind(intent);
}
}
```

Выполните компиляцию проекта и запустите его на эмуляторе Android. При нажатии на кнопку **Start** запустится служба оповещения. При этом каждые 5 секунд на экране эмулятора будет появляться всплывающее уведомление с порядковым номером, как показано на рис. 29.1.

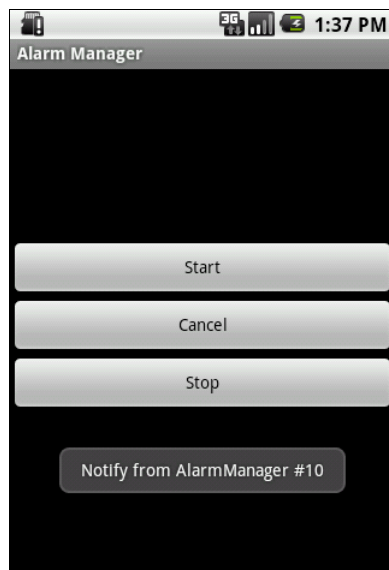


Рис. 29.1. Приложение для создания периодических оповещений

Теперь я хотел бы обратить внимание на управление нашей службой. Если нажать кнопку **Cancel**, в классе `AlarmManagerActivity` будет вызван метод `AlarmManager.cancel()`, и служба перестанет работать, но не завершится, т. к. не будет

вызван метод `onDestroy()` службы. Если теперь снова нажать кнопку **Start**, снова будут генерироваться уведомления, и их отсчет продолжится.

При нажатии кнопки **Stop** будет вызван метод `stopService()`, и служба завершит работу и освободит ресурсы. Но поскольку в этом случае экземпляр `AlarmManager` не был закрыт методом `cancel()`, будет создан и запущен новый экземпляр службы, счетчик обнулится и начнет работу сначала. Для того чтобы наша служба была полностью остановлена и освобождены ресурсы, надо в классе `AlarmManagerActivity` последовательно вызвать методы `cancel()` и `stopService()`.

При желании можете поэкспериментировать с приложением, добавив уведомления в строке состояния со значками, надписями и звуковыми оповещениями вместо простых всплывающих уведомлений.

Резюме

В этой главе мы рассмотрели управление пользовательскими оповещениями из приложений. Использование службы `Alarm Service` позволяет создавать приложения для интерактивного взаимодействия с пользователем мобильного устройства.

В следующей главе мы рассмотрим работу с менеджером буфера обмена и его возможности при работе с текстом, а также библиотеку для синтеза речи на основе введенного пользователем текста.



ГЛАВА 30

Буфер обмена и API для работы с текстом

В состав системы Android также входят различные службы для работы с текстом: служба буфера обмена Clipboard Service и специальная библиотека для синтеза речи на основе введенного текста — Text To Speech.

В этой главе мы рассмотрим использование в своих приложениях функциональностей, предоставляемых службой Clipboard Service и библиотеками для синтеза речи.

Менеджер буфера обмена

Для доступа к службе Clipboard Service в коде приложения используется класс `ClipboardManager`, который находится в библиотеке `android.text`. Эта библиотека содержит набор классов для работы с текстом.

Начиная с версии Android SDK 3.0 (API Level 11) определено два класса `ClipboardManager`, которые включены в разные пакеты. Один из них находится в пакете `android.text` — это устаревший менеджер буфера обмена. Начиная с версии Android 3.0 произошли большие изменения в менеджере буфера обмена и появился еще один класс `ClipboardManager`, расположенный в пакете `android.content`. Используя новую версию `ClipboardManager`, приложения теперь могут копировать и вставлять не только обычный текст, но также URI и объекты `Intent`.

Объект `ClipboardManager` в коде программы можно получить так же, как и почти все остальные системные менеджеры, которые мы рассматривали в этой книге, используя вызов метода `getSystemService()`, передав этому методу входной параметр со значением `CLIPBOARD_SERVICE`:

```
ClipboardManager manager =  
    (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
```

Класс `ClipboardManager` очень простой и предназначен только для работы с текстом, он не может копировать графику и файлы, как в настольных системах. Класс `ClipboardManager` содержит набор методов для работы с текстом:

- ☐ `getText()` — возвращает текст, находящийся в буфере обмена;
- ☐ `setText()` — вставляет текст, который передается методу во входном параметре, в буфер обмена;

- ❑ `hasText()` — возвращает `true`, если в буфере обмена находится текст, и `false` — если буфер обмена пустой.

Использовать менеджер буфера обмена в коде приложения очень просто, и мы сейчас это рассмотрим. Создайте в IDE Eclipse новый проект Android и заполните поля в окне **New Android Project**:

- ❑ **Project name** — `ClipboardManager`;
- ❑ **Application name** — `Clipboard service`;
- ❑ **Package name** — `com.samples.clipboardmanager`;
- ❑ **Create Activity** — `ClipboardManagerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch30_ClipboardManager`.

В файле компоновки главного окна приложения `main.xml` создайте четыре виджета:

- ❑ `EditText` с идентификатором `textCopy` — поле для ввода текста пользователем;
- ❑ `Button` с идентификатором `bCopy` и надписью **Copy** — для копирования текста в буфер обмена;
- ❑ `Button` с идентификатором `bPaste` и надписью **Paste** — для вставки текста в буфер обмена;
- ❑ `TextView` с идентификатором `textPaste` — для отображения скопированного текста из буфера обмена.

Код файла компоновки `main.xml` приведен в листинге 30.1.

Листинг 30.1. Файл компоновки окна приложения `main.xml`

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

    <EditText
        android:id="@+id/textCopy"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Enter text"/>

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent">
        <Button
            android:id="@+id/bCopy"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:text="Copy"
```

```

        android:onClick="onClick"
        android:layout_weight="1"/>
<Button
    android:id="@+id/bPaste"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:text="Paste"
    android:onClick="onClick"
    android:layout_weight="1"/>
</LinearLayout>

<TextView
    android:id="@+id/textPaste"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textStyle="bold"/>

</LinearLayout>

```

Код класса главного окна приложения `ClipboardManagerActivity` будет очень простым: в методе `onCreate()` создается объект `ClipboardManager`. Этот объект мы используем в обработчике события `onClick()` кнопок для копирования и вставки введенного текста в буфер обмена с помощью метода `setText()`, а также для последующего чтения текста из буфера обмена и отображения его в текстовом поле, используя метод `getText()` класса `ClipboardManager`.

Код класса главного окна приложения представлен в листинге 30.2.

Листинг 30.2. Файл класса главного окна приложения `ClipboardManagerActivity.java`

```

package com.samples.clipboardmanager;

import android.app.Activity;
import android.content.ClipboardManager;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class ClipboardManagerActivity extends Activity
    implements View.OnClickListener {

    private ClipboardManager manager;
    private EditText textCopy;
    private TextView textPaste;

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

```

```

textCopy = (EditText)findViewById(R.id.textCopy);
textPaste = (TextView)findViewById(R.id.textPaste);

manager = (ClipboardManager) getSystemService(CLIPBOARD_SERVICE);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bCopy:
            manager.setText(textCopy.getText());
            break;
        case R.id.bPaste:
            textPaste.append(manager.getText() + "\n");
            break;
    }
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. В текстовое поле можно вводить произвольный текст, который при нажатии кнопки будет скопирован в буфер обмена. Внешний вид приложения представлен на рис. 30.1.



Рис. 30.1. Использование менеджера буфера обмена

Синтез речи на основе текста

Платформа Android включает сервис Text To Speech (TTS), который позволяет на мобильном устройстве синтезировать речь на основе текста. Синтезируемая речь может быть немедленно воспроизведена или сохранена в звуковой файл, который можно затем запустить как обычный звуковой файл.

Для того чтобы можно было использовать эту функциональность, у мобильного устройства Android должен быть установлен сервис TTS, который доступен, начиная

с версии Android 1.6 (API Level 4). Также должны быть установлены файлы ресурсов для соответствующего языка.

По умолчанию на мобильном телефоне обычно присутствуют ресурсы для английского языка. Если требуются дополнительные языки, их можно установить через опцию в **Settings | Text-to-speech | Install voice data** (рис. 30.2), используя удаленный сетевой сервис.

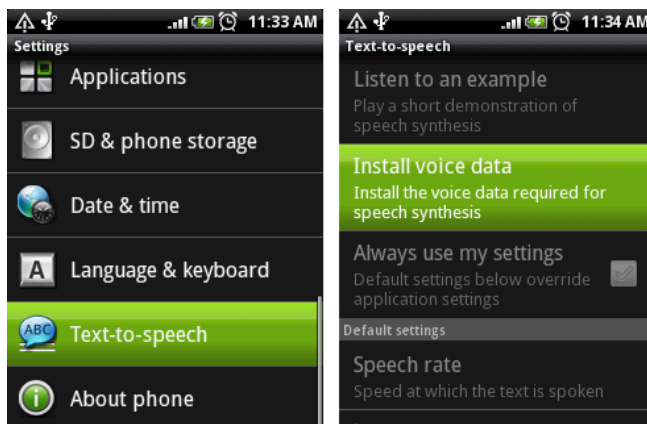


Рис. 30.2. Установка файлов языковых ресурсов

Библиотека Text To Speech API является частью стандартного Android SDK и находится в пакете `android.speech.tts`. Непосредственно за синтез речи в библиотеке `android.speech.tts` отвечает класс `TextToSpeech`. Этот класс содержит набор методов, управляющих синтезированием речи. Вот основные методы, обеспечивающие эту функциональность:

- `speak()` — воспроизводит текст;
- `stop()` — прерывает текущий речевой синтез и не воспроизводит другие тексты, находящиеся в очереди;
- `shutdown()` — освобождает ресурсы, использовавшиеся TTS;
- `synthesizeToFile()` — синтезирует текст и отправляет его в указанный звуковой файл;
- `isSpeaking()` — возвращает `true`, если сервис TTS в данный момент синтезирует речь.

Метод `speak()` принимает три параметра: воспроизводимый текст, режим воспроизведения и необязательный список дополнительных параметров воспроизведения. Речевое воспроизведение текста занимает довольно длительное время, новые записи в сервис TTS могут добавляться быстрее, чем они будут воспроизводиться. При добавлении записи в сервис TTS можно предусмотреть пополнение сервиса новыми записями, которые будут становиться в очередь на воспроизведение. Для установки режима добавления записей в классе `TextToSpeech` определены две константы:

- `QUEUE_ADD` — устанавливает режим очереди, когда новая запись добавляется в конец очереди;

- ❑ `QUEUE_FLUSH` — устанавливает режим очереди, когда все записи, находящиеся в очереди на воспроизведение, удаляются и заменяются новой записью.

При использовании сервиса TTS очень важно наличие поддержки языков, которые будут воспроизводиться, поэтому в классе `TextToSpeech` определена группа методов для определения используемых и установки требуемых языков:

- ❑ `getLanguage()` — возвращает объект `Locale`, который содержит язык, используемый в данный момент в TTS;
- ❑ `setLanguage()` — устанавливает язык, который передается в качестве параметра в виде объекта `Locale`, для TTS;
- ❑ `isLanguageAvailable()` — проверяет язык, переданный в качестве параметра, на доступность для TTS.

Если у вас будет предусмотрена возможность работы с разными языками, в приложение нужно добавить функциональность для проверки наличия нужных языков и произвести установку, если они отсутствуют.

Инициализация объекта `TextToSpeech` в приложении занимает некоторое время. Для того чтобы узнать о завершении процесса инициализации сервиса `Text To Speech`, используется интерфейс `OnInitListener`. Этот интерфейс содержит единственный метод `onInit()`, в который в качестве входного параметра передается состояние сервиса `Text To Speech`. Значение для состояния сервиса определяется в классе `TextToSpeech` двумя константами:

- ❑ `ERROR`;
- ❑ `SUCCESS`.

Объект класса `TextToSpeech` может быть использован для синтеза текста, только когда он завершит свою инициализацию. При создании объекта `TextToSpeech` в конструктор класса необходимо передать два параметра: текущий контекст приложения и текущий экземпляр `TextToSpeech.OnInitListener`. Реализация интерфейса `TextToSpeech.OnInitListener` позволит приложению получать уведомления о завершении инициализации:

```
public class TtsActivity extends Activity implements OnInitListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        // Создаем объект TextToSpeech и передаем
        // контекст приложения и экземпляр OnInitListener
        tts = new TextToSpeech(getApplicationContext(), this);
        ...
    }

    // Реализация метода onInit интерфейса OnInitListener
    @Override
    public void onInit(int status) {
        if (status == TextToSpeech.SUCCESS) {
            // Действия при инициализации
        }
        ...
    }
}
```

Для запуска синтеза речи необходимо вызвать метод `speak()`, например, следующим образом:

```
String text = "Hello";
tts.speak(text, TextToSpeech.QUEUE_FLUSH, null);
```

Когда приложение закрывается, для экземпляра `TextToSpeech` необходимо вызвать метод `shutdown()`, чтобы освободить используемые им ресурсы.

Сейчас мы испытаем работу с этим сервисом в приложении. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `TextToSpeech`;
- ☐ **Application name** — `Text to speech`;
- ☐ **Package name** — `com.samples.api.tts`;
- ☐ **Create Activity** — `TtsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch30_TextToSpeech`.

В код файла компоновки главного окна приложения `main.xml` добавьте следующие виджеты:

- ☐ `Button` с идентификатором `bSpeech`;
- ☐ `EditText` с идентификатором `text`, развернутый на весь экран.

Код файла `main.xml` представлен в листинге 30.3.

Листинг 30.3. Файл компоновки окна приложения `main.xml`

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

    <Button
        android:text="Speech"
        android:id="@+id/bSpeech"
        android:layout_width="fill_parent"
        android:onClick="onClick"
        android:layout_height="wrap_content"/>

    <EditText
        android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="@string/hello"/>

</LinearLayout>
```

Код класса `TtsActivity` очень простой. В методе `onCreate()` создается объект `TextToSpeech`. В обработчике `onClick()` кнопки вызывается метод `speak()`, которому в качестве входного параметра передается текст, введенный пользователем в поле `EditText`.

Код класса главного окна приложения представлен в листинге 30.4.

Листинг 30.4. Файл класса главного окна приложения `TtsActivity.java`

```
package com.samples.api.tts;
import android.app.Activity;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class TtsActivity extends Activity
    implements TextToSpeech.OnInitListener, View.OnClickListener {

    private TextToSpeech tts;
    private Button bSpeech;
    private EditText text;

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

        text = (EditText)findViewById(R.id.text);
        bSpeech = (Button)findViewById(R.id.bSpeech);
        bSpeech.setEnabled(false);

        tts = new TextToSpeech(getApplicationContext(), this);
    }

    @Override
    public void onStop() {
        super.onStop();

        // Освобождаем ресурсы, используемые TTS
        tts.shutdown();
    }

    @Override
    public void onInit(int status) {
        // Если TextToSpeech успешно инициализирован,
        // кнопка Speech будет разблокирована
    }
}
```



```
        if (status == TextToSpeech.SUCCESS) {  
            bSpeech.setEnabled(true);  
        }  
    }  
  
    @Override  
    public void onClick(View v) {  
        // Запускаем синтез речи  
        tts.speak(text.getText().toString(),  
            TextToSpeech.QUEUE_FLUSH, null);  
    }  
}
```

Скомпилируйте проект и запустите его на эмуляторе Android или на своем мобильном устройстве. Внешний вид приложения представлен на рис. 30.3.

Введите в поле любой текст на английском языке и нажмите кнопку **Speak**. Ваш введенный текст будет синтезирован в голосовое сообщение. Конечно, синтезированный голос звучит довольно примитивно и напоминает речь роботов из старых фантастических фильмов, однако используя библиотеку Text To Speech, можно создавать очень интересные приложения для интерактивного взаимодействия с пользователем мобильного телефона!

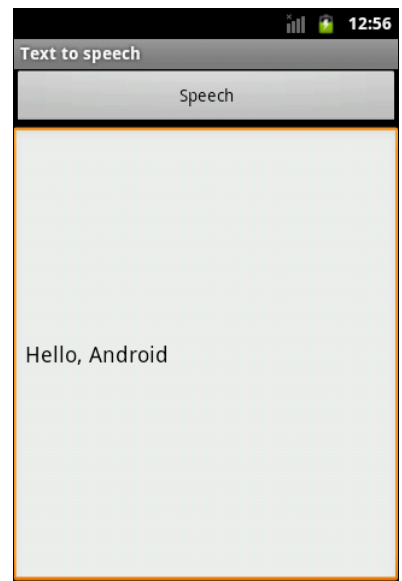
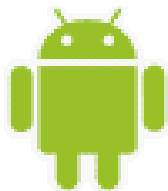


Рис. 30.3. Приложение с использованием TextToSpeech API

Резюме

В этой главе мы рассмотрели функциональность, предоставляемую Android SDK для работы с буфером обмена. Также мы рассмотрели использование в приложении библиотеки Text To Speech для синтеза речи на основе текста, введенного пользователем.

Далее мы переходим к следующей части книги, где будем изучать сетевые возможности устройств Android и работу с удаленными сервисами.



ЧАСТЬ V

Сетевые сервисы

Глава 31. Получение информации о телефоне и сети сотовой связи

Глава 32. Обработка телефонных вызовов

Глава 33. Отправка и получение SMS

Глава 34. Мобильный Интернет

Глава 35. Управление Wi-Fi-соединениями

Глава 36. Определение местоположения

Глава 37. Сервис Geocoding

Глава 38. Использование карт Google Maps в приложениях



ГЛАВА 31

Получение информации о телефоне и сети сотовой связи

В этой главе будет рассказано о том, как получить информацию о телефоне и сети сотовой связи и использовать ее в своих приложениях. Система Android позволяет установленным на мобильном устройстве приложениям производить телефонные вызовы, отправлять и принимать SMS-сообщения, отслеживать состояние телефона и сети сотовой связи. Весь этот богатый набор средств управления телефоном вы можете использовать при разработке собственных приложений.

Информация о телефоне

Для доступа к информации о телефоне и сети сотовой связи используется класс `TelephonyManager`, который находится в пакете `android.telephony`. Этот класс позволяет приложению получить доступ к системной службе мобильной телефонии на устройстве.

Экземпляр этого класса не создается напрямую в программе. Для получения экземпляра `TelephonyManager` необходимо вызвать метод `getSystemService()`, передав ему параметр `Context.TELEPHONY_SERVICE`:

```
TelephonyManager manage = (TelephonyManager) getSystemService(  
    Context.TELEPHONY_SERVICE);
```

Класс `TelephonyManager` имеет большой набор методов для определения типа и состояния телефона. Кроме того, с помощью этих методов вы можете в своем приложении определять тип и доступность сети сотовой связи, состояние SIM-карты и многое другое. Далее в этой главе мы подробно разберем использование функциональности, предоставляемой классом `TelephonyManager` для управления службой `Telephony Service` и получения от этой службы нужной нам информации.

Определение типа телефона и сети сотовой связи

Для определения типа телефона и доступности сети сотовой связи в классе `TelephonyManager` есть методы `getPhoneType()` и `getNetworkType()`. Эти методы возвращают целочисленные константы, определяющие типы телефона и сети сотовой связи.

Метод `getPhoneType()` определяет тип мобильного телефона и возвращает одно из четырех значений:

- ❑ `PHONE_TYPE_GSM` — телефон стандарта GSM (Global System for Mobile communications, глобальный цифровой стандарт для сотовой связи). Этот стандарт, я думаю, известен всем. Он был разработан Европейским институтом стандартов телекоммуникаций ETSI еще в 80-х годах XX века. Стандарт GSM на сегодняшний день является наиболее распространенным стандартом связи;
- ❑ `PHONE_TYPE_CDMA` — телефон стандарта CDMA (Code Division Multiple Access, множественный доступ с кодовым разделением). Это технология мобильной связи, при которой каналы передачи имеют общую полосу частот, но разную кодовую модуляцию;
- ❑ `PHONE_TYPE_SIP` — телефон стандарта SIP (Session Initiation Protocol, протокол инициализации соединения). Несмотря на то, что эта технология известна уже давно и широко используется в IP-телефонии, поддержка SIP в Android появилась совсем недавно, в версии Android SDK 2.3 (API Level 9);
- ❑ `PHONE_TYPE_NONE` — это значение используется, если по каким-то причинам тип телефона не подходит под предыдущие определения.

Метод `getNetworkType()` определяет тип сети сотовой связи, в зоне действия которой в данный момент находится мобильный телефон. В настоящее время существует большое количество разнообразных типов сетей, поэтому данный метод возвращает гораздо больше вариантов значений, чем метод `getPhoneType()`:

- ❑ `NETWORK_TYPE_GPRS` — сеть GPRS (General Packet Radio Service, сеть с пакетной передачей данных). Это расширение технологии GSM для пакетной передачи данных. Технология GPRS позволяет пользователю мобильного телефона производить обмен данными с другими устройствами в сети GSM и с внешними сетями, в том числе с Интернетом. Весь поток данных отправителя разбивается на отдельные пакеты и затем доставляется получателю. Технология GPRS позволяет передавать данные на существенно более высоких скоростях, чем в обычной GSM-сети;
- ❑ `NETWORK_TYPE_CDMA` — сеть CDMA (эта технология уже была описана ранее в этом разделе);
- ❑ `NETWORK_TYPE_1xRTT` — сеть 1xRTT (One Times Radio Transmission Technology). Это мобильная технология передачи цифровых данных, основанная на CDMA-технологии, но использующая принцип передачи с коммутацией пакетов;
- ❑ `NETWORK_TYPE_EDGE` — сеть EDGE (Enhanced Data Rates for Global Evolution, технология высокоскоростной передачи данных в сетях GSM). Технология EDGE является расширением технологии передачи данных с коммутацией каналов для увеличения пропускной способности этого сервиса;
- ❑ `NETWORK_TYPE_EHRPD` — сеть EHRPD (Evolved High-Rate Packet Data, улучшенная технология высокоскоростной пакетной передачи данных);
- ❑ `NETWORK_TYPE_LTE` — сеть LTE (Long Term Evolution,). Технология LTE является усовершенствованием технологий CDMA и UMTS с повышенной скоростью передачи данных и уже относится к сетям четвертого поколения (4G);
- ❑ `NETWORK_TYPE_EVDO_0`, `NETWORK_TYPE_EVDO_A`, `NETWORK_TYPE_EVDO_B` — сети типа EVDO (Evolution-Data Optimized). Это технология передачи данных, используемая в сетях сотовой связи стандарта CDMA.1X;

- ❑ `NETWORK_TYPE_HSPA` — сеть HSPA (High Speed Packet Access, технология высокоскоростной передачи данных);
- ❑ `NETWORK_TYPE_HSDPA` — сеть HSDPA (High Speed Downlink Packet Access). Это расширение HSPA — технология высокоскоростной передачи данных по направлению от сети на мобильное устройство;
- ❑ `NETWORK_TYPE_HSUPA` — сеть HSUPA (High Speed Uplink Packet Access). Это технология, аналогично HSDPA, высокоскоростной пакетной передачи данных для передачи данных в обратном направлении, от мобильного устройства в сеть;
- ❑ `NETWORK_TYPE_IDEN` — сеть IDEN (Integrated Digital Enhanced Network). Это интегрированная технология, объединяющая групповую диспетчерскую связь с сотовой телефонной связью и технологию передачи алфавитно-цифровых сообщений и данных. Она была разработана компанией Motorola для построения телекоммуникационных систем для больших организаций;
- ❑ `NETWORK_TYPE_UMTS` — сеть UMTS (Universal Mobile Telecommunications System). Этот тип сети представляет собой усовершенствованную базовую сотовую сеть GSM и радиоинтерфейс по технологии WCDMA. Эта технология тоже была разработана институтом ETSI;
- ❑ `NETWORK_TYPE_UNKNOWN` — значение, возвращаемое в случае, когда тип сети неизвестен или не подходит под предыдущие определения.

Конечно, в одном месте столько типов сетей одновременно не используются, но библиотека `android.telephony` предусматривает все возможные типы сетей, т. к. мобильные телефоны Android могут работать в любой точке мира, и к тому же сегодня уже существует множество производителей мобильных устройств Android, которые поставляют свою продукцию практически во все регионы мира. Кстати, список констант в классе `TelephonyManager`, определяющий типы мобильных сетей, постоянно пополняется: с каждой новой версией Android SDK обычно добавляются одна-две новые константы.

Определение базовой станции сотовой связи

Для получения информации о базовой станции сети сотовой связи в классе `TelephonyManager` предусмотрен метод `getCellLocation()`. Этот метод возвращает объект `CellLocation`.

Сам тип `CellLocation` является абстрактным классом, но от него наследуются два класса, которые, в зависимости от типа используемой сети, можно применить для получения информации о базовой станции:

- ❑ `GsmCellLocation` — для сетей GSM;
- ❑ `CdmaCellLocation` — для сетей CDMA.

Класс `GsmCellLocation` имеет набор методов для определения базовой станции сотовой связи:

- ❑ `getCid()` — возвращает идентификатор базовой станции сотовой связи;
- ❑ `getLac()` — возвращает LAC (Location Area Code). Это территориальная единица, в пределах которой в данный момент находится абонент мобильного устройства;

- ❑ `getPsc()` — возвращает PSC (Primary Scrambling Code). Это код базовой станции для сетей UMTS. Метод `getPsc()` появился в классе `GsmCellLocation` недавно, начиная с Android SDK 2.31.1 (API Level 9).

Класс `CdmaCellLocation` предназначен для сетей CDMA и является более "продвинутым" по сравнению с классом `GsmCellLocation`. Его набор методов позволяет получить более полную информацию о базовой станции сети CDMA:

- ❑ `getBaseStationId()` — возвращает идентификатор базовой станции;
- ❑ `getNetworkId()` — возвращает идентификатор сети CDMA;
- ❑ `getSystemId()` — возвращает системный идентификатор сети CDMA;
- ❑ `getBaseStationLatitude()` — возвращает географические координаты широты базовой станции;
- ❑ `getBaseStationLongitude()` — возвращает географические координаты долготы базовой станции.

Определение состояния вызова

Для определения состояния вызова мобильного телефона в классе `TelephonyManager` предусмотрен метод `getCallState()`. Этот метод возвращает целочисленное значение, которое определяет три возможных состояния:

- ❑ `CALL_STATE_IDLE` — телефон не активен;
- ❑ `CALL_STATE_OFFHOOK` — производится попытка вызова;
- ❑ `CALL_STATE_RINGING` — телефон в состоянии соединения с абонентом.

Этот метод позволяет отслеживать состояние телефона в коде приложения, чтобы при необходимости произвести определенные действия.

Получение информации о роуминге

Если абонент находится в другом государстве или за пределами территории обслуживания своего сетевого оператора, в приложении можно получить информацию о подключении роуминга. Для этих целей в классе `TelephonyManager` есть метод `isNetworkRoaming()`, который возвращает переменную типа `boolean`, показывающую, находится ли данный абонент в роуминге.

Использование класса *TelephonyManager*

Как вы видите, класс `TelephonyManager`, представляющий службу `Telephony Service`, позволяет получить большое количество самой разнообразной информации. Сейчас мы попробуем в практическом приложении использовать возможности, предоставляемые `TelephonyManager`. Создайте в IDE Eclipse новый проект, заполнив следующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `PhoneInfo`;
- ❑ **Application name** — `Phone info`;

- ❑ **Package name** — `com.samples.telephony.phoneinfo`;
- ❑ **Create Activity** — `PhoneInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch31_PhoneInfo`.

В файл манифеста приложения `AndroidManifest.xml` обязательно добавьте разрешение `READ_PHONE_STATE`, чтобы приложение могло получать информацию от системы Android о состоянии телефона, и разрешение `ACCESS_COARSE_LOCATION` для получения информации о базовой станции сети сотовой связи, как показано в листинге 31.1 (выделено полужирным шрифтом), иначе при запуске приложения у вас будет генерироваться исключение.

Листинг 31.1. Файл манифеста приложения `AndroidManifest.xml`

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

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity
            android:name="PhoneInfoActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission
        android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION" />
</manifest>
```

В файл компоновки окна приложения `main.xml` добавьте текстовое поле `TextView`, присвойте ему идентификатор `android:id="@+id/text"`. В это поле будет выводиться информация о мобильном телефоне и сети сотовой связи. Код файла компоновки окна `main.xml` показан в листинге 31.2.

Листинг 31.2. Файл компоновки окна приложения `main.xml`

```
<?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:layout_margin="5px"
    android:textStyle="bold"/>

</LinearLayout>

```

В нашем классе `PhoneInfoActivity` будет создаваться экземпляр `TelephonyManager`, из которого мы сможем получить всю нужную информацию о телефоне и сети сотовой связи. Код класса `PhoneInfoActivity` представлен в листинге 31.3.

Листинг 31.3. Файл класса главного окна приложения `PhoneInfoActivity.java`

```

package com.samples.telephony.phoneinfo;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.telephony.cdma.CdmaCellLocation;
import android.telephony.gsm.GsmCellLocation;
import android.widget.TextView;

public class PhoneInfoActivity extends Activity {
    private TextView text;
    private TelephonyManager manager;

    @Override
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        this setContentView(R.layout.main);

        text = (TextView)findViewById(R.id.text);

        manager = (TelephonyManager) getSystemService(
            Context.TELEPHONY_SERVICE);

        text.append("\nCallState:\t" +
            convertCallStateToString(manager.getCallState()));
        text.append("\nDevice ID:\t" +
            manager.getDeviceId());
        text.append("\nDevice Software Version:\t" +
            manager.getDeviceSoftwareVersion());
    }
}

```

```
text.append("\nLine1 Number:\t" +
            manager.getLine1Number());
text.append("\nNetwork Type:\t" +
            convertNetworkTypeToString(manager.getNetworkType()));
text.append("\nNetwork Country ISO:\t" +
            manager.getNetworkCountryIso());
text.append("\nNetwork Operator:\t" +
            manager.getNetworkOperator());
text.append("\nNetwork Operator Name:\t" +
            manager.getNetworkOperatorName());
text.append("\nPhone Type:\t" +
            convertPhoneTypeToString(manager.getPhoneType()));
text.append("\nData Activity:\t" +
            convertDataActivityToString(manager.getDataActivity()));
text.append("\nData State:\t" +
            convertDataStateToString(manager.getDataState()));
text.append("\nSubscriber ID:\t" +
            manager.getSubscriberId());
text.append("\nVoice Mail Alpha Tag:\t" +
            manager.getVoiceMailAlphaTag());
text.append("\nVoice Mail Number:\t" +
            manager.getVoiceMailNumber());
text.append("\nIcc Card:\t" +
            manager.hasIccCard());
text.append("\nNetwork Roaming:\t" +
            manager.isNetworkRoaming());

GsmCellLocation gsmCell =
    (GsmCellLocation)manager.getCellLocation();
if (gsmCell != null) {
    text.append("\nGSM Cell Location:");
    text.append("\n\tCID:\t" + gsmCell.getCid());
    text.append("\n\tLAC:\t" + gsmCell.getLac());
}

private String convertCallStateToString(int callState) {
    switch (callState) {
        case TelephonyManager.CALL_STATE_IDLE:
            return "IDLE";
        case TelephonyManager.CALL_STATE_OFFHOOK:
            return "OFFHOOK";
        case TelephonyManager.CALL_STATE_RINGING:
            return "RINGING";
        default:
            return "Not defined";
    }
}
```

```
private String convertNetworkTypeToString(int networkType) {
    switch (networkType) {
        case TelephonyManager.NETWORK_TYPE_1xRTT:
            return "1xRTT";
        case TelephonyManager.NETWORK_TYPE_CDMA:
            return "CDMA";
        case TelephonyManager.NETWORK_TYPE_EDGE:
            return "EDGE";
        case TelephonyManager.NETWORK_TYPE_EVDO_0:
            return "EVDO revision 0";
        case TelephonyManager.NETWORK_TYPE_EVDO_A:
            return "EVDO revision A";
        //case TelephonyManager.NETWORK_TYPE_EVDO_B:
        //    return "EVDO revision B";
        case TelephonyManager.NETWORK_TYPE_GPRS:
            return "GPRS";
        case TelephonyManager.NETWORK_TYPE_HSDPA:
            return "HSDPA";
        case TelephonyManager.NETWORK_TYPE_HSPA:
            return "HSPA";
        case TelephonyManager.NETWORK_TYPE_HSUPA:
            return "HSUPA";
        //case TelephonyManager.NETWORK_TYPE_IDEN:
        //    return "iDen";
        case TelephonyManager.NETWORK_TYPE_UMTS:
            return "UMTS";
        case TelephonyManager.NETWORK_TYPE_UNKNOWN:
            return "Unknown";
        default:
            return "Not defined";
    }
}

private String convertDataActivityToString(int dataActivity) {
    switch (dataActivity) {
        case TelephonyManager.DATA_ACTIVITY_DORMANT:
            return "Dormant";
        case TelephonyManager.DATA_ACTIVITY_IN:
            return "In";
        case TelephonyManager.DATA_ACTIVITY_INOUT:
            return "In-out";
        case TelephonyManager.DATA_ACTIVITY_NONE:
            return "None";
        case TelephonyManager.DATA_ACTIVITY_OUT:
            return "Out";
        default:
            return "Not defined";
    }
}
```

```
private String convertDataStateToString(int dataState) {
    switch (dataState) {
        case TelephonyManager.DATA_CONNECTED:
            return "Data connected";
        case TelephonyManager.DATA_CONNECTING:
            return "Data connecting";
        case TelephonyManager.DATA_DISCONNECTED:
            return "Data suspended";
        case TelephonyManager.DATA_SUSPENDED:
            return "Data suspended";
        default:
            return "Not defined";
    }
}

private String convertPhoneTypeToString(int phoneType) {
    switch (phoneType) {
        case TelephonyManager.PHONE_TYPE_GSM:
            return "GSM";
        case TelephonyManager.PHONE_TYPE_CDMA:
            return "CDMA";
        case TelephonyManager.PHONE_TYPE_NONE:
            return "NONE";
        default:
            return "Not defined";
    }
}
}
```

Скомпилируйте и запустите приложение на вашем мобильном устройстве. В зависимости от типа подключения и сети сотовой связи, приложение выведет всю доступную информацию на экран мобильного устройства. Например, на рис. 31.1 представлен возможный вариант вывода данных о телефоне и сети сотовой связи для конкретного телефона, подключенного к сети.

В принципе, эмулятор Android также в состоянии имитировать подключение к сети сотовой связи. Поэтому часть функциональности разрабатываемого приложения вполне можно отлаживать на эмуляторе. Пример вывода информации о состоянии телефона и сотовой сети при развертывании приложения на эмуляторе Android показан на рис. 31.2.

В целом, для тестирования и отладки приложений подобного типа использовать эмулятор Android даже удобнее, т. к. при помощи инструмента **Dalvik Debug Monitor Server** можно имитировать динамику изменений состояния сетей сотовой связи, например переход из одной сети в другую. Подробнее об этом будет рассказано далее в этой главе, а сейчас мы рассмотрим доступ к SIM-карте мобильного телефона.

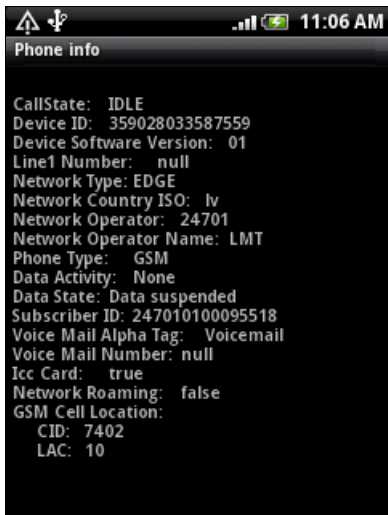


Рис. 31.1. Вывод информации о телефоне на реальном мобильном устройстве

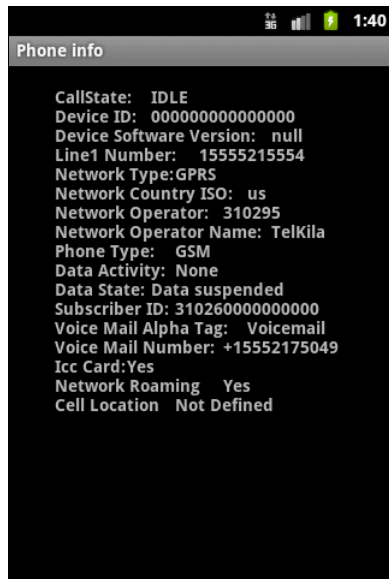


Рис. 31.2. Вывод информации о телефоне на эмуляторе Android

Доступ к SIM-карте

Используя класс `TelephonyManager`, можно получить полезную информацию о SIM-карте, находящейся в мобильном телефоне. Для этих целей в классе `TelephonyManager` предусмотрено несколько методов:

- ❑ `getSimCountryIso()` — этот метод возвращает код государства, в котором находится SIM-провайдер сотовой связи. Этот код соответствует международному стандарту ISO 3166 и использует 2-буквенный код (alpha-2) для обозначения государств и территорий, например `ru`, `sw` и т. д.;
- ❑ `getSimOperator()` — возвращает строку MCC + MNC (Mobile Country Code + Mobile Network Code, код государства и код мобильной сети провайдера SIM-карты). Эта строка является глобальным уникальным идентификатором оператора мобильной связи;
- ❑ `getSimOperatorName()` — возвращает SPN (Service Provider Name, имя сервис-провайдера);
- ❑ `getSimSerialNumber()` — возвращает серийный номер SIM-карты при условии, что он доступен (срабатывает не на всех картах);
- ❑ `getSimState()` — возвращает целочисленный идентификатор, определяющий состояние SIM-карты (о состояниях SIM-карты будет рассказано далее в этой главе).

Как вы видите, набор методов для программного доступа к SIM-карте телефона, предоставляемый классом `TelephonyManager`, довольно ограничен и выдает информацию только для чтения. Это сделано по соображениям безопасности пользователей мобильного телефона.

Состояние SIM-карты

С помощью метода `getSimState()` в приложении можно определить текущее состояние (работоспособность) SIM-карты. Целое число, возвращаемое методом `getSimState()`, может, в зависимости от состояния SIM-карты, принимать значения, определяемые константами, объявленными в классе `TelephonyManager`:

- ❑ `SIM_STATE_READY` — SIM-карта доступна;
- ❑ `SIM_STATE_ABSENT` — SIM-карта отсутствует на мобильном устройстве;
- ❑ `SIM_STATE_PIN_REQUIRED` — SIM-карта заблокирована, от пользователя требуется ввести PIN-код для разблокирования карты;
- ❑ `SIM_STATE_PUK_REQUIRED` — SIM-карта заблокирована, от пользователя требуется ввести PUK-код (8-значный код) для разблокирования карты;
- ❑ `SIM_STATE_NETWORK_LOCKED` — SIM-карта заблокирована, от пользователя требуется ввести сетевой PIN-код для разблокирования карты;
- ❑ `SIM_STATE_UNKNOWN` — состояние SIM-карты неизвестно.

Эти константы можно применять в приложениях, в которых требуется функциональность для определения состояния SIM-карты, например, для автоматической рассылки SMS.

Доступ к SIM-карте из приложения

Давайте теперь создадим приложение, которое способно получить доступ к SIM-карте мобильного телефона и информацию о всех ее доступных параметрах. Для этого откройте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `SimInfo`;
- ❑ **Application name** — `SIM card info`;
- ❑ **Package name** — `com.samples.telephony.siminfo`;
- ❑ **Create Activity** — `SimInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch31_SimInfo`.

В файле манифеста приложения `AndroidManifest.xml` добавьте разрешение для чтения состояния телефона `READ_PHONE_STATE`, как в листинге 31.1. Для компоновки окна приложения используйте файл `main.xml`, представленный в листинге 31.2.

Класс главного окна приложения `SimInfoActivity` в целом аналогичен классу `PhoneInfoActivity` из предыдущего примера. Код класса `SimInfoActivity` представлен в листинге 31.4.

Листинг 31.4. Файл класса главного окна приложения `SimInfoActivity.java`

```
package com.samples.telephony.siminfo;

import android.app.Activity;
import android.content.Context;
```

```
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class SimInfoActivity extends Activity {

    private TextView text;

    @Override
    public void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this setContentView(R.layout.main);

        this.text = (TextView)findViewById(R.id.text);

        final TelephonyManager manager = (TelephonyManager) getSystemService(
            Context.TELEPHONY_SERVICE);

        String simCountryIso = manager.getSimCountryIso();
        String simOperator = manager.getSimOperator();
        String simOperatorName = manager.getSimOperatorName();
        String simSerialNumber = manager.getSimSerialNumber();
        String simSubscriberId = manager.getSubscriberId();
        int simState = manager.getSimState();

        String sSimStateString = "Not Defined";

        switch (simState) {
            case TelephonyManager.SIM_STATE_ABSENT:
                sSimStateString = "ABSENT";
                break;
            case TelephonyManager.SIM_STATE_NETWORK_LOCKED:
                sSimStateString = "NETWORK_LOCKED";
                break;
            case TelephonyManager.SIM_STATE_PIN_REQUIRED:
                sSimStateString = "PIN_REQUIRED";
                break;
            case TelephonyManager.SIM_STATE_PUK_REQUIRED:
                sSimStateString = "PUK_REQUIRED";
                break;
            case TelephonyManager.SIM_STATE_READY:
                sSimStateString = "STATE_READY";
                break;
            case TelephonyManager.SIM_STATE_UNKNOWN:
                sSimStateString = "STATE_UNKNOWN";
                break;
        }

        text.setText(
            "\nSim CountryIso: " + simCountryIso +
```



```
        "\nSim Operator: " + simOperator +  
        "\nSim OperatorName: " + simOperatorName +  
        "\nSim SerialNumber: " + simSerialNumber +  
        "\nSim SubscriberId: " + simSubscriberId +  
        "\nSim StateString: " + sSimStateString);  
    }  
}
```

Скомпилируйте и запустите приложение на вашем мобильном устройстве. Эмулятор Android способен симулировать SIM-карту, и это приложение, запущенное на эмуляторе Android, также выдаст информацию о SIM-карте (рис. 31.3).

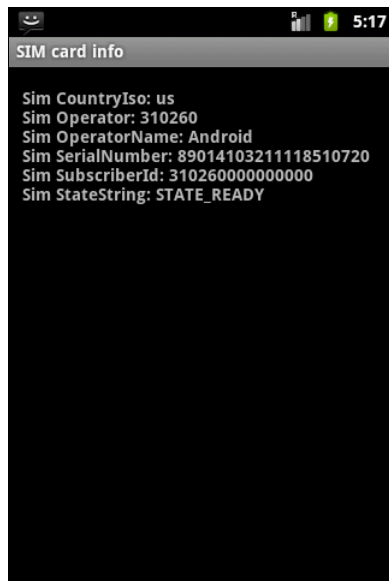


Рис. 31.3. Вывод информации о SIM-карте

Перехват изменений состояния параметров телефона

Мобильный телефон постоянно перемещается вместе со своим владельцем. При этом может изменяться уровень сигнала сети сотовой связи, тип сети, ее доступность и еще целый ряд других параметров.

Для отслеживания этих изменений в пакете `android.telephony` предназначен класс `PhoneStateListener`. Этот класс содержит объявления методов обратного вызова, предназначенных для прослушивания различных изменений состояния телефона и сотовой сети.

Вот основные методы, определенные в классе `PhoneStateListener`:

- ❑ `onCallStateChanged()` — вызывается при изменении состояния вызова телефона, например, когда устройство переходит из состояния ожидания в состояние вызова. При этом первым параметром в метод передается целочисленное значение, определяющее новое состояние телефона (константы `CALL_STATE_IDLE`, `CALL_STATE_OFFHOOK`, `CALL_STATE_RINGING`), а вторым параметром передается номер входящего вызова;
- ❑ `onCellLocationChanged()` — вызывается при смене базовой станции сотовой сети;

- ❑ `onDataActivity()` — вызывается при смене направления передачи данных;
- ❑ `onDataConnectionStateChanged()` — вызывается при изменении состояния соединения передачи данных;
- ❑ `onServiceStateChanged()` — вызывается при изменении состояния мобильного устройства;
- ❑ `onSignalStrengthsChanged()` — вызывается при изменении уровня сигнала сети сотовой связи;
- ❑ `onCallForwardingIndicatorChanged()` — вызывается при изменении состояния булевого индикатора, который указывает на наличие перенаправленного вызова;
- ❑ `onMessageWaitingIndicatorChanged()` — вызывается при изменении состояния булевого индикатора, который определяет состояние отправки SMS-сообщения.

Далее мы рассмотрим возможности, предоставляемые классом `PhoneStateListener` для отслеживания изменений характеристик сети.

Запуск и остановка прослушивания изменений состояния сотовой сети

Для использования класса `PhoneStateListener` в приложении сначала необходимо создать объект `PhoneStateListener` и определить нужные вам методы (необязательно все), которые вы будете использовать для перехвата событий:

```
private PhoneStateListener listener = new PhoneStateListener() {
    @Override
    public void onCallStateChanged(
        final int state, final String incomingNumber) {
        // Действия при наступлении события
    }

    @Override
    public void onDataConnectionStateChanged(int state, int networkType) {
        // Действия при наступлении события
    }
    ...
};
```

Затем в коде программы вы должны для своего экземпляра `TelephonyManager` зарегистрировать слушатель событий. Для этого в классе `TelephonyManager` предусмотрен метод `listen()`, принимающий два параметра:

- ❑ объект `PhoneStateListener`, который вы определили ранее;
- ❑ целочисленную константу, определяющую нужное событие, которое вы собираетесь прослушивать.

Эти константы определены в классе `PhoneStateListener`. Вот их полный список:

- ❑ `LISTEN_CALL_FORWARDING_INDICATOR;`
- ❑ `LISTEN_CALL_STATE;`
- ❑ `LISTEN_CELL_LOCATION;`

- ☐ `LISTEN_DATA_ACTIVITY;`
- ☐ `LISTEN_DATA_CONNECTION_STATE;`
- ☐ `LISTEN_MESSAGE_WAITING_INDICATOR;`
- ☐ `LISTEN_SERVICE_STATE;`
- ☐ `LISTEN_SIGNAL_STRENGTHS.`

Как можно определить из названий, каждая из этих констант соответствует конкретному методу обратного вызова, объявленному в классе `PhoneStateListener`, которые уже были описаны в предыдущем разделе. Например, константа `LISTEN_CELL_LOCATION` соответствует методу `onCellLocationChanged()`.

Мы можем зарегистрировать прослушивание событий изменения базовой станции сети сотовой связи следующим образом:

```
TelephonyManager manager = (TelephonyManager) getSystemService(  
    Context.TELEPHONY_SERVICE);  
manager.listen(cellLocationListener,  
    PhoneStateListener.LISTEN_CELL_LOCATION);
```

Если в приложении нам требуется прослушивать сразу несколько событий, то регистрировать их обработку можно одновременно для требуемых типов слушателей событий, используя комбинацию ИЛИ:

```
manager.listen(cellLocationListener,  
    PhoneStateListener.LISTEN_CELL_LOCATION |  
    PhoneStateListener.LISTEN_SIGNAL_STREIGHT);
```

Для отключения прослушивания событий в приложении в классе `PhoneStateListener` предусмотрена константа `LISTEN_NONE`. Чтобы остановить прослушивание изменений состояния сети, надо вызвать метод `listen()` и передать ему в качестве второго параметра значение этой константы:

```
manager.listen(cellLocationListener, PhoneStateListener.LISTEN_NONE);
```

Изменение уровня сигнала

Изменение уровня принимаемого сигнала — важная информация для функционирования мобильного устройства. Чем меньше уровень сигнала, тем больше вероятность возникновения ошибки при передаче или приеме данных. Поэтому если для работы приложения критичен уровень сигнала, необходимо в коде программы определить метод `onSignalStrengthsChanged()`, который будет вызываться каждый раз при изменении уровня сигнала сети сотовой связи.

Этот метод принимает параметр типа `SignalStrength`, который определяет состояние сигнала станции сотовой связи. Для измерения характеристик уровня сигнала класс `SignalStrength` содержит несколько методов:

- ☐ `getCdmaDbm()` — возвращает значение RSSI (Received Signal Strength Indication, индикатор уровня принимаемого сигнала). Значение RSSI характеризует уровень принимаемого сигнала базовой станции в dBm (русское обозначение — дБм). Этот метод возвращает целое число — значение в децибелах относительно опорного уровня мощности 1 милливатт. Например, уровень 30 dBm равен мощности 1 Вт (использу-

ется логарифмическая шкала). Можно использовать данный показатель для определения расстояния до базовой станции сети сотовой связи;

- ❑ `getEvdoDbm()` — возвращает значение RSSI для сетей EVDO;
- ❑ `getCdmaEcio()` — возвращает соотношение сигнал/шум (Ec/Io) принимаемого сигнала в dB * 10 для сетей CDMA;
- ❑ `getEvdoEcio()` — возвращает соотношение сигнал/шум (Ec/Io) принимаемого сигнала для сетей EVDO;
- ❑ `getEvdoSnr()` — возвращает значение соотношения сигнал/шум, которое может находиться в диапазоне от 0 до 8, где 8 является лучшим значением;
- ❑ `getGsmSignalStrength()` — возвращает значение уровня сигнала GSM Signal Strength;
- ❑ `getGsmBitErrorRate()` — возвращает GSM Error Rate. Error Rate — это рейтинг возможности появления ошибки в передающем или принимаемом сигнале в зависимости от соотношения сигнал/шум.

Значения GSM Signal Strength, GSM Error Rate и уровень сигнала определены в технической спецификации для терминалов GSM TS 27.007 8.5. Например, GSM Signal Strength может принимать значения от 0 до 31 и 99. Оно равно 0 при уровне сигнала -113 dBm или ниже, 1 при уровне -111 dBm, от 2 до 30 при уровне от -109 до -53 dBm, 31 при -51 dBm и выше, 99 — если уровень сигнала неизвестен или его невозможно измерить.

Изменение базовой станции сотовой связи

Приложение может получать уведомление при изменении базовой станции сотовой связи. В обработчик события `onCellLocationChanged()` передается параметр типа `CellLocation`. С помощью этого параметра можно определить характеристики базовой станции сотовой связи.

Чтобы отслеживать изменение базовой станции сотовой связи в приложении, требуется разрешение `ACCESS_COARSE_LOCATION`, которое необходимо добавить в файл манифеста приложения:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Мониторинг состояния подключения к сервису

Приложение может получать уведомление при отказе в обслуживании, например, при попадании в зону действия сети другого оператора, при низком уровне или при отсутствии сигнала базовой станции. Для этого используется метод `onServiceStateChanged()`, которому передается параметр типа `ServiceState`. Класс `ServiceState` определяет состояние подключения к сотовому сервису и идентифицирует оператора мобильной сети.

В классе `ServiceState` есть метод `getState()`, возвращающий одно из следующих целочисленных значений:

- ❑ `STATE_IN_SERVICE` — сотовый сервис доступен для данного абонента;
- ❑ `STATE_EMERGENCY_ONLY` — разрешены только экстренные вызовы;

- ❑ `STATE_OUT_OF_SERVICE` — абонент телефона не обслуживается;
- ❑ `STATE_POWER_OFF` — передатчик мобильного устройства отключен. Этот режим обычно ставится при включении опции телефона **Airplane Mode**.

В классе `ServiceState` также имеется набор методов для получения дополнительной информации об операторе сотовой сети:

- ❑ `getOperatorAlphaLong()` — возвращает полное имя оператора сети сотовой связи;
- ❑ `getOperatorAlphaShort()` — возвращает сокращенное имя оператора;
- ❑ `getOperatorNumeric()` — возвращает идентификатор оператора.

Кроме этого, у класса `ServiceState` есть булевы методы для индикации состояния роуминга `getRoaming()` и `getIsManualSelection()`, который возвращает `true`, если переключение операторов установлено в ручном режиме, и `false` — если в автоматическом.

Приложение для прослушивания изменений состояния сотовой сети

Возможность отслеживать динамику изменений состояния сотовой сети может оказаться очень полезной для приложений, которые активно используют сотовые коммуникации для передачи и приема данных. Сейчас мы разработаем приложение, которое будет отслеживать изменения состояния сотовой сети. Создайте в IDE Eclipse новый проект Android, заполнив следующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `PhoneChangeStateInfo`;
- ❑ **Application name** — `Phone state change info`;
- ❑ **Package name** — `com.samples.telephony.phonechangestateinfo`;
- ❑ **Create Activity** — `PhoneChangeStateInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch31_PhoneChangeStateInfo`.

В файле компоновки окна приложения мы расположим две командные кнопки **Start** и **Stop** для запуска и остановки мониторинга с идентификаторами `bStart` и `bStop`. Также создадим текстовое поле `TextView` с идентификатором `text` для вывода информации об изменениях состояния сети.

Файл компоновки окна приложения `main.xml` представлен в листинге 31.5.

Листинг 31.5. Файл компоновки окна приложения `main.xml`

```
<?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">
```

```

<LinearLayout
    android:layout_height="wrap_content"
    android:layout_width="fill_parent">

    <Button
        android:id="@+id/bStart"
        android:layout_height="wrap_content"
        android:text="Start"
        android:layout_width="fill_parent"
        android:onClick="onClick"
        android:layout_weight="1"/>

    <Button
        android:id="@+id/bStop"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Stop"
        android:onClick="onClick"
        android:layout_weight="1"/>

</LinearLayout>

<TextView
    android:id="@+id/text"
    android:text=""
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textStyle="bold"/>

</LinearLayout>

```

В классе `PhoneChangeStateInfoActivity` определим закрытую переменную `listener` типа `PhoneStateListener`, внутри которой реализуем обработчики событий различных изменений состояния сети. Информация о происходящих изменениях будет добавляться в текстовое поле `text`.

Код класса `PhoneChangeStateInfoActivity`, за исключением закрытых вспомогательных методов `convertCallStateToString()`, `convertNetworkTypeToString()`, `convertData-ActivityToString()`, `convertDataConnStateToString()` и `convertPhoneTypeToString()`, которые аналогичны методам из листинга 31.3, представлен в листинге 31.6.

Листинг 31.6. Файл класса окна приложения `PhoneChangeStateInfoActivity.java`

```

package com.samples.telephony.phonechangestateinfo;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.telephony.PhoneStateListener;
import android.telephony.ServiceState;
import android.telephony.TelephonyManager;
import android.view.View;
import android.view.View.OnClickListener;

```

```
import android.widget.Button;
import android.widget.TextView;

public class PhoneChangeStateInfoActivity extends Activity
    implements OnClickListener {
    private TextView text;
    private TelephonyManager manager;

    // Определяем обработчики событий изменений состояния сети
    private PhoneStateListener listener = new PhoneStateListener() {
        @Override
        public void onCallStateChanged(
            final int state, final String incomingNumber) {
            text.append("\nCall state:\t" +
                convertCallStateToString(state) + "\nIncoming number:\t");
        }

        @Override
        public void onDataConnectionStateChanged(
            int state, int networkType) {
            text.append("\nNetwork Type:\t" +
                convertNetworkTypeToString(networkType));
        }

        @Override
        public void onCallForwardingIndicatorChanged(boolean cfi) {
            text.append("\nCallForwardingIndicator:\t" + cfi);
        }

        @Override
        public void onDataActivity(int direction) {
            text.append("\nDataActivity:\t" +
                convertDataActivityToString(direction));
        }

        @Override
        public void onDataConnectionStateChanged(int state) {
            text.append("\nDataConnectionState:\t" +
                convertDataConnStateToString(state));
        }

        @Override
        public void onMessageWaitingIndicatorChanged(boolean mwi) {
            text.append("\nMessageWaitingIndicator:\t" + mwi);
        }

        @Override
        public void onServiceStateChanged(ServiceState serviceState) {
            text.append("\nService State:\t" +
                convertServiceStateToString(serviceState.getState()));
        }
    };
};
```

```

@Override
public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    this setContentView(R.layout.main);

    text = (TextView)findViewById(R.id.text);
    text.setText("Listener is stopped");

    // Создаем экземпляр TelephonyManager
    manager = (TelephonyManager) getSystemService(
        Context.TELEPHONY_SERVICE);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bStart:
            // Регистрируем слушателей событий
            manager.listen(listener,
                PhoneStateListener.LISTEN_CALL_STATE |
                PhoneStateListener.LISTEN_CALL_FORWARDING_INDICATOR |
                PhoneStateListener.LISTEN_DATA_ACTIVITY |
                PhoneStateListener.LISTEN_DATA_CONNECTION_STATE |
                PhoneStateListener.LISTEN_MESSAGE_WAITING_INDICATOR |
                PhoneStateListener.LISTEN_SERVICE_STATE);
            text.setText("Start phone info listener...");
            break;

        case R.id.bStop:
            // Останавливаем прослушивание событий
            manager.listen(listener,
                PhoneStateListener.LISTEN_NONE);
            text.setText("Listener is stopped");
            break;
    }
}

// Вспомогательный метод для конвертации ServiceState в строку
private String convertServiceStateToString(int serviceState) {
    switch (serviceState) {
        case ServiceState.STATE_EMERGENCY_ONLY:
            return "Emergency Only";
        case ServiceState.STATE_IN_SERVICE:
            return "In Service";
        case ServiceState.STATE_OUT_OF_SERVICE:
            return "Out of Service";
        case ServiceState.STATE_POWER_OFF:
            return "Power OFF";
    }
}

```

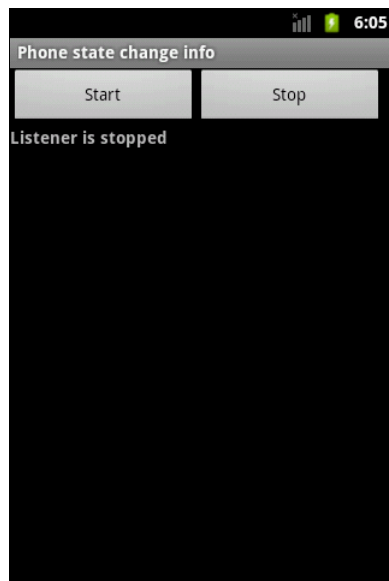


```
default:
    return "Not defined";
}
}
...
}
```

Скомпилируйте и запустите приложение на эмуляторе. Внешний вид нашего приложения представлен на рис. 31.4.

При запуске приложение находится в состоянии покоя и не реагирует на изменения состояния сотовой сети. Для начала прослушивания надо нажать кнопку **Start**.

Рис. 31.4. Начало работы приложения для прослушивания изменения состояний сети



Однако возникает вопрос: как отлаживать и тестировать такие приложения? Ведь вы же не будете ходить с мобильным телефоном по улице, чтобы ловить изменения характеристик сотовой сети. Оказывается, часть работы по тестированию такой функциональности можно сделать, не отходя от компьютера. Для тестирования такого приложения нам потребуется инструмент Dalvik Debug Monitor Server.

Использование эмулятора для тестирования приложений

При разработке приложений для работы с `TelephonyManager` совсем необязательно для отладки и тестирования программы использовать реальное мобильное устройство. Эмуляторы Android и Dalvik Debug Monitor Server предоставляют требуемую функциональность.

Запустите DDMS напрямую из каталога Android SDK (подкаталог `tools\ddms.bat`) или из среды Eclipse. В среде Eclipse надо открыть меню **Window | Show View | Other** и выбрать опции **Devices** и **Emulator Control** (рис. 31.5).

Представление **Devices** отображает запущенный эмулятор Android. Если запущено несколько экземпляров эмулятора, их можно различать по номеру порта, через который эмулятор связывается с DDMS. Номер порта отображается в заголовке окна запущенного экземпляра эмулятора. Например, для эмулятора с именем AVD1 в заголовке окна

будет надпись **5554:AVD1**. Кроме того, номер порта отображается в **Devices**, как показано на рис. 31.6.

Для управления эмулятором используется представление **Emulator Control**. С его помощью можно моделировать различные состояния мобильного устройства и действия пользователя. Внешний вид **Emulator Control** в среде Eclipse представлен на рис. 31.7.

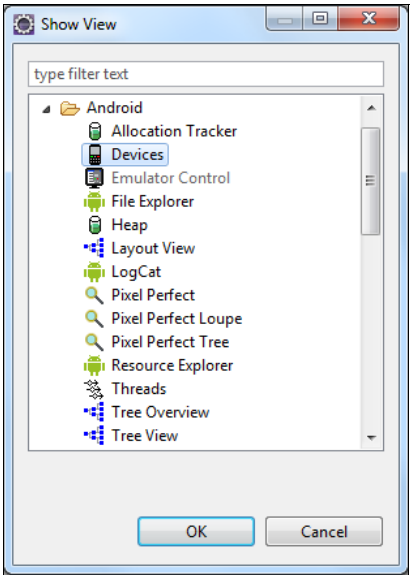


Рис. 31.5. Открытие представлений **Devices** и **Emulator Control** в IDE Eclipse

Name		
emulator-5554	Online	AVD2_3 [2.3.3, debug]
system_process	76	8600
jp.co.omronsoft.openwnn	162	8606
com.android.phone	167	8609
com.android.systemui	169	8611
com.android.settings	218	8616
com.android.launcher	233	8618
android.process.acore	250	8620
android.process.media	282	8625
com.android.quicksearchbc	301	8628
com.android.defcontainer	305	8630
com.android.protips	324	8632
com.android.music	337	8634
com.android.mms	350	8636
com.samples.telephony.phi	373	8638
com.android.deskclock	391	8639
com.android.email	401	8640
com.svox.pico	419	8641

Рис. 31.6. Представление **Devices**

Представление **Emulator Control** содержит три панели:

- ❑ **Telephony Status** — изменение состояния мобильного устройства и моделирование различных типов сети (GPRS, EDGE, UTMS и т. д.);
- ❑ **Telephony Actions** — выполнение моделируемых обращений по телефону и сообщений SMS к эмулятору;
- ❑ **Location Controls** — моделирование местоположения телефона (работу с местоположением мы рассмотрим в главе 36 книги).

Для установления связи с работающим экземпляром эмулятора необходимо в текстовое поле **Incoming number** ввести номер порта этого эмулятора (5554). Теперь можно протестировать наше приложение с помощью **Emulator Control**. Запустите в приложении мониторинг изменений состояния сотовой сети, нажав кнопку **Start**. Приложение выведет текущую информацию о состоянии сотовой сети, как показано на рис. 31.8.

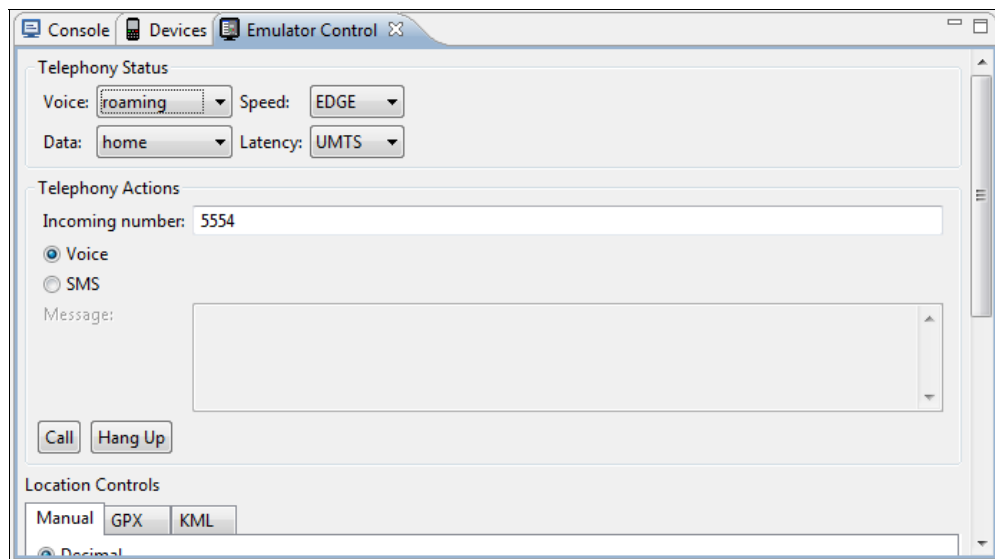


Рис. 31.7. Представление Emulator Control

Затем в представлении **Emulator Control** на панели **Telephony Status** попробуйте поменять значения в выпадающих списках **Voice**, **Data**, **Speed**, **Latency** (см. рис. 31.7). Приложение должно реагировать на изменение состояния сотовой сети, обновляя информацию на экране.

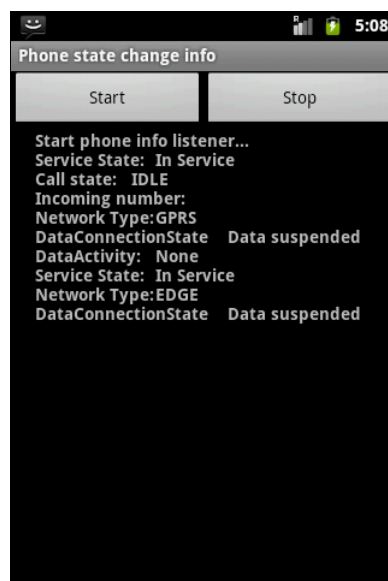


Рис. 31.8. Перехват приложением изменения статуса сети

Резюме

В этой главе мы рассмотрели программный доступ к базовой службе телефона `TelephonyManager` для получения информации о состоянии телефона и сети сотовой связи. Также мы научились реагировать в программе на изменения этих состояний.

В следующей главе мы рассмотрим возможности отправки и получения SMS-сообщений напрямую из нашего приложения.



ГЛАВА 32

Обработка телефонных вызовов

Платформа Android предоставляет пользовательским приложениям широкую функциональность для управления и отслеживания входящих и исходящих телефонных вызовов. Например, приложение может автоматически произвести вызов определенного абонента или сформировать группу номеров и осуществить их вызов в режиме конференции. Можно также создавать службы, работающие в фоновом режиме, которые будут отслеживать входящие и исходящие телефонные вызовы и сохранять эту информацию для пользователя.

В этой главе мы займемся созданием телефонных вызовов и управлением ими из программного кода приложения с использованием системных компонентов, предоставляемых платформой Android.

Использование эмулятора для тестирования обработки телефонных вызовов

При разработке приложений для работы с входящими и исходящими телефонными вызовами эмулятор Android предоставляет нам всю необходимую функциональность. Для имитации входящих и исходящих звонков в эмуляторе мобильного устройства существуют два способа:

- ☐ имитация телефонного вызова с использованием DDMS;
- ☐ имитация телефонного вызова между двумя экземплярами эмуляторов Android.

Эти возможности эмулятора Android и DDMS пригодятся нам в этой главе при разработке приложений, работающих с телефонными вызовами.

Имитация телефонного вызова из DDMS

В предыдущей главе мы уже использовали DDMS для изменения состояния мобильной сети. Аналогично, для работы с обычными телефонными звонками, можно использовать способы, описанные в предыдущей главе.

Чтобы послать вызов, достаточно установить связь с выполняющимся экземпляром эмулятора через представление **Devices** в IDE Eclipse, затем перейти в представление **Emulator Control**. Далее в группе **Telephony Actions** в текстовом поле **Incoming number** ввести номер порта запущенного экземпляра эмулятора Android, поставить переключатель в положение **Voice** и нажать кнопку **Call** (рис. 32.1).

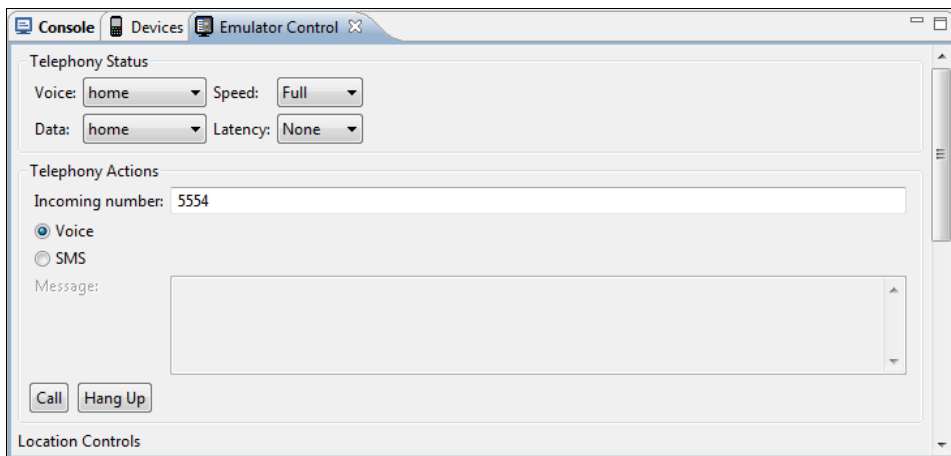


Рис. 32.1. Имитация телефонного звонка из Eclipse

В результате наш запущенный экземпляр эмулятора должен получить входящий вызов, как показано на рис. 32.2.

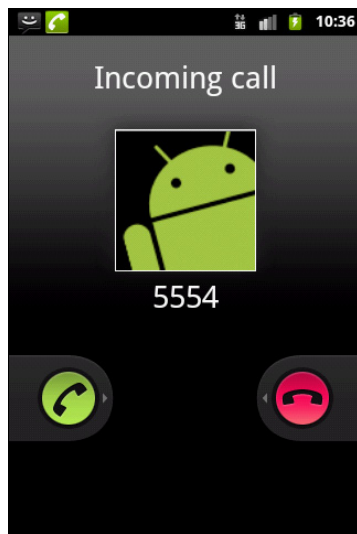
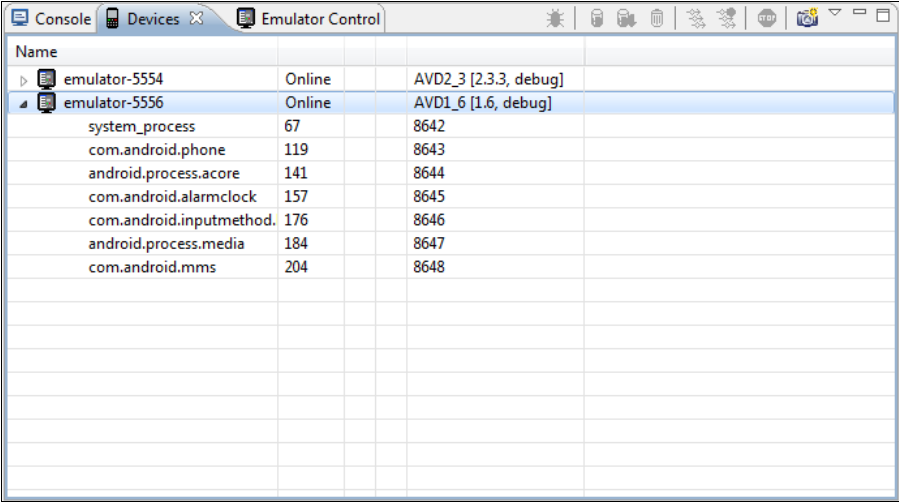


Рис. 32.2. Входящий звонок в эмуляторе Android

Имитация телефонного вызова между двумя эмуляторами Android

Для имитации телефонных вызовов между двумя эмуляторами Android запустите еще один экземпляр эмулятора. Запущенные экземпляры эмуляторов будут видны в среде Eclipse в представлении **Devices**, как показано на рис. 32.3.

Как вы уже видели, при создании телефонного вызова абонентским номером эмулятора является номер его порта. Например, если у вас уже создан один эмулятор, то по умолчанию его абонентским номером будет 5554. При создании дополнительных эмуляторов, т. е. новых экземпляров **Android Virtual Device**, каждому новому виртуальному устройству будет присвоен новый номер порта с номером, увеличенным на 2: например 5556, 5558 и т. д.



Name	Online	AVD
emulator-5554	Online	AVD2_3 [2.3.3, debug]
emulator-5556	Online	AVD1_6 [1.6, debug]
system_process	67	8642
com.android.phone	119	8643
android.process.acore	141	8644
com.android.alarmclock	157	8645
com.android.inputmethod	176	8646
android.process.media	184	8647
com.android.mms	204	8648

Рис. 32.3. Запущенные экземпляры эмуляторов Android в окне **Devices**

Каждый запущенный экземпляр эмулятора Android использует пару смежных портов: порт для консоли и порт для **Android Debug Bridge** (инструмента для отладки приложений Android, входящего в состав Android SDK). Это означает, что консоль первого экземпляра эмулятора, запущенного на компьютере, использует порт с номером 5554 для консоли и порт 5555 для **Android Debug Bridge**.

Следующие запущенные экземпляры эмуляторов Android будут использовать номера портов в сторону увеличения — например, пару портов 5556, 5557 для второго экземпляра эмулятора, пару портов 5558, 5559 для третьего и т. д. Всего одновременно можно запустить до 16 экземпляров эмуляторов, если хватит доступной памяти на машине.

Например, мы хотим инициализировать исходящий звонок из эмулятора с номером 5556 на эмулятор 5554. Для этого открываем на эмуляторе 5556 панель набора номера, вводим номер порта вызываемого эмулятора — 5554 и делаем вызов. При этом на эмулятор 5554 поступит входящий звонок.

Кстати, обратите внимание на номер абонента входящего вызова: при пересылке к нему добавляется префикс "1-555-521-" для имитации "настоящего" номера абонента, как показано на рис. 32.4.

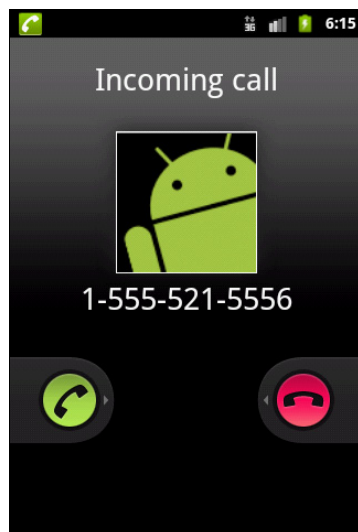


Рис. 32.4. Входящий звонок в эмуляторе 5554, полученный из эмулятора 5556

Такая симуляция телефонных вызовов в эмуляторе Android нам понадобится для тестирования приложений, которые мы будем разрабатывать далее в этой главе.

Установка разрешений

В приложении, которое программно создает или перехватывает телефонные вызовы, нужно также задавать разрешения. Далее перечислен список разрешений, требуемых для приложения, которое должно работать с телефонными вызовами.

- ☐ `CALL_PHONE` — разрешает приложению инициализацию телефонного вызова без подтверждения пользователя;
- ☐ `CALL_PRIVILEGED` — разрешает приложению инициализацию телефонного вызова любого номера, включая вызов экстренных служб без подтверждения пользователя;
- ☐ `PROCESS_OUTGOING_CALLS` — разрешает приложению получать оповещение типа `Broadcast Intent` при инициализации пользователем исходящего звонка;
- ☐ `READ_PHONE_STATE` — разрешает приложению получать доступ к информации о состоянии телефона (это разрешение мы уже использовали в предыдущей главе);
- ☐ `MODIFY_PHONE_STATE` — разрешает приложению модифицировать состояние телефона.

Например, чтобы приложение могло производить телефонные вызовы абонента из кода, необходимо добавить разрешение `CALL_PHONE` в файл манифеста приложения `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

В зависимости от функций, выполняемых приложением, может применяться одновременно несколько перечисленных разрешений.

Использование объектов *Intent* для создания телефонных вызовов

Приложения Android способны создавать телефонные вызовы, однако для отправки вызова абоненту необходимо запустить окно наборной панели, напрямую из приложения вызов абонента не производится. Вызов этого окна выполняется обычным способом, через объект `Intent`. В классе `Intent` для этих целей предусмотрен набор действий, определенный следующими константами:

- ☐ `ACTION_DIAL`;
- ☐ `ACTION_CALL`;
- ☐ `ACTION_CALL_BUTTON`.

Если в коде приложения требуется просто вызвать окно стандартной панели с наборными кнопками, используется действие `ACTION_CALL_BUTTON`, например, следующим образом:

```
Intent intent = new Intent(Intent.ACTION_CALL_BUTTON);  
startActivity(intent);
```

В результате на экран мобильного устройства будет вызвана панель набора номера, показанная на рис. 32.5.

При использовании `ACTION_CALL_BUTTON` набор номера и вызов автоматически не производятся, пользователю необходимо ввести номер и сделать вызов самостоятельно. Однако две другие константы имеют более широкую функциональность, которую мы рассмотрим далее.

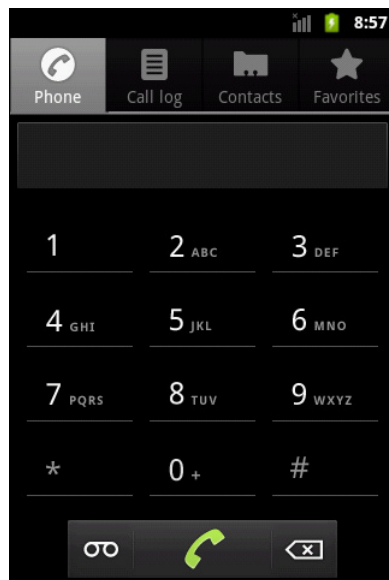


Рис. 32.5. Вызов стандартной панели набора номера

Вызов телефонного абонента из приложения

С помощью действий `ACTION_DIAL` и `ACTION_CALL` можно вызывать конкретного телефонного абонента, передавая в конструктор класса `Intent` в качестве Extra-параметра (дополнения) телефонный номер абонента.

Сам Extra-параметр для объекта `Intent` — не просто телефонный номер, а объект `URI`, имеющий следующий формат:

```
tel:номер_абонента
```

Таким образом, чтобы инициировать исходящий звонок с заданным номером из приложения, надо создать объект `URI` с заданной структурой и передать его в конструктор объекта `Intent`, например, следующим образом:

```
String phoneNum = "5554"
Uri uri = Uri.parse("tel:" + phoneNum);
Intent intent = new Intent(Intent.ACTION_DIAL, uri);
startActivity(intent);
```

Сейчас мы попробуем сделать это в приложении. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — PhoneCall;
- ☐ **Application name** — Phone call;
- ☐ **Package name** — com.samples.telephony.phonecall;
- ☐ **Create Activity** — PhoneCallActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch32_PhoneCall.

В файл манифеста нужно обязательно добавить разрешение `android.permission.CALL_PHONE` для обеспечения возможности телефонного вызова абонента из приложения (листинг 32.1).

Листинг 32.1. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.telephony.phonecall"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".PhoneCallActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.CALL_PHONE" />
</manifest>
```

В файле компоновки главного окна `main.xml` приложения создайте поле `EditText` для ввода телефонного номера и кнопку вызова с надписью **Call** и идентификатором `bCall` (листинг 32.2).

Листинг 32.2. Файл компоновки окна приложения `main.xml`

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

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Number:" />
        <EditText android:id="@+id/textNumber"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
```

```
        android:cursorVisible="true"
        android:editable="true"
        android:singleLine="true"/>
</LinearLayout>

<Button android:id="@+id/bCall"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Call"
        android:onClick="onClick"/>

</LinearLayout>
```

В коде класса `PhoneCallActivity` приложения в обработчике события нажатия кнопки `onClick()` будет создаваться объект `Intent` с Extra-параметром, содержащим URI введенного в текстовое поле номера абонента.

Код класса `PhoneCallActivity` представлен в листинге 32.3.

Листинг 32.3. Файл окна приложения `PhoneCallActivity.java`

```
package com.samples.telephony.phonecall;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class PhoneCallActivity extends Activity
    implements View.OnClickListener{

    private EditText textNumber;

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

        textNumber=(EditText)findViewById(R.id.textNumber);
    }

    @Override
    public void onClick(View arg0) {
        try {
            Uri uri = Uri.parse("tel:" + textNumber.getText().toString());
            startActivity(new Intent(Intent.ACTION_DIAL, uri));
        }
    }
}
```

```

catch (Exception e) {
    Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
}
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. Внешний вид приложения представлен на рис. 32.6.

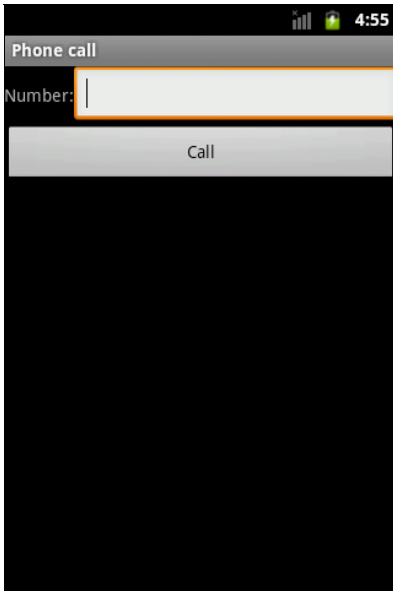


Рис. 32.6. Приложение для создания исходящего вызова

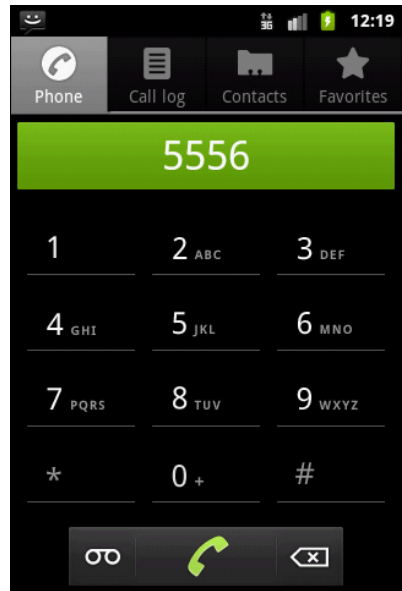


Рис. 32.7. Загрузка вызова абонента в окне наборной панели

Теперь протестируем наше приложение. Для этого запустим второй экземпляр эмулятора. Ему будет присвоен номер порта 5556. Теперь в поле для ввода номера нашего приложения введите "абонентский" номер второго эмулятора Android (5556) и нажмите кнопку **Call**. При этом в эмуляторе должно появиться окно наборной панели с введенным номером: 5556, как показано на рис. 32.7.

Сам вызов при использовании действия `ACTION_DIAL` не производится. Пользователь должен сам нажать кнопку вызова на наборной панели, чтобы инициировать исходящий звонок. При вызове абонента 5556 на втором экземпляре эмулятора Android должно появиться окно оповещения о входящем звонке, такое же, как на рис. 32.2.

Чтобы произвести телефонный вызов из приложения без участия пользователя мобильного устройства, используется действие `ACTION_CALL`. Для этого в приложении измените код в обработчике события `onClick()` кнопки, заменив в конструкторе объекта `Intent` значение `Intent.ACTION_DIAL` на значение `Intent.ACTION_CALL`, как показано в листинге 32.4.

Листинг 32.4. Обработчик события `onClick()` класса `PhoneCallActivity`

```
@Override
public void onClick(View arg0) {
    try {
        Uri uri = Uri.parse("tel:" + textNumber.getText().toString());
        startActivity(new Intent(Intent.ACTION_CALL, uri));
    }
    catch (Exception e) {
        Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
    }
}
```

Скомпилируйте проект и запустите его на эмуляторе Android с номером 5554. Теперь приложение будет инициировать исходящий звонок без участия пользователя, однако пользователю будет отображен экран вызова абонента, так что пользователь все же получит информацию о том, что его мобильный телефон производит телефонный вызов другого абонента (рис. 32.8).

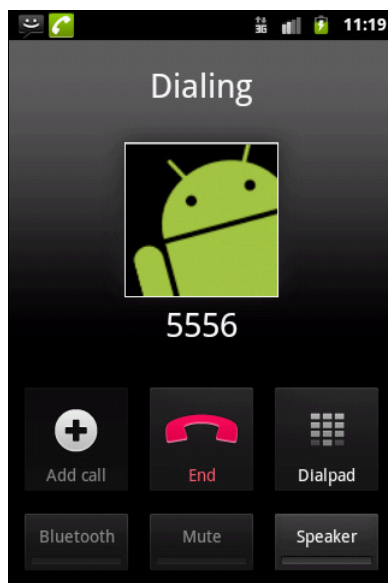


Рис. 32.8. Прямой телефонный вызов абонента 5556

Перехват исходящих звонков

Помимо вызова телефонного номера из приложения и создания вызовов, можно также программно отслеживать исходящие вызовы на мобильном устройстве. Для отслеживания событий, происходящих в системе, используются специальные приемники — объекты `BroadcastReceiver`.

Чтобы реализовать отслеживание исходящих звонков, необходимо создать пользовательский класс, наследуемый от класса `BroadcastReceiver`, и переопределить в нем метод `onReceive()`. Этот метод принимает два параметра: объекты `Context` и `Intent`. В объекте `Intent` будет находиться информация об исходящем звонке, номер которого можно получить, вызвав метод `getExtras()`. Например, это можно реализовать следующим образом:

```
public void onReceive(final Context context, final Intent intent) {
    ...
    String phoneNumber = intent.getExtras().getString(
        Intent.EXTRA_PHONE_NUMBER);
    ...
}
```

Теперь создадим приложение для перехвата исходящих вызовов. Это приложение будет выполняться в фоновом режиме и при инициализации исходящего звонка будет отображать всплывающее уведомление. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — OutgoingCall;
- ☐ **Application name** — Outgoing Call;
- ☐ **Package name** — com.samples.outgoingcall;
- ☐ **Create Activity** — оставляем незаполненным, наше приложение не будет иметь графического интерфейса.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch32_OutgoingCall.

В файле манифеста приложения AndroidManifest.xml необходимо объявить компонент Broadcast Receiver. Для этого создайте элемент `<receiver>` с атрибутом `android:name="OutgoingCallReceiver"`. Это будет имя нашего класса-приемника. Также в элементе `<receiver>` создайте вложенный Intent-фильтр, используя элемент `<intent-filter>`. В Intent-фильтр добавьте вложенный элемент `<action>` с атрибутом `android:name="android.intent.action.NEW_OUTGOING_CALL"`, который будет фильтром, принимающим только исходящие звонки, предназначенные для компонента Broadcast Receiver.

Кроме того, в файл манифеста необходимо также поместить разрешение `PROCESS_OUTGOING_CALLS` для возможности отслеживания исходящих звонков, как показано в листинге 32.5.

Листинг 32.5. Файл манифеста приложения AndroidManifest.xml

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

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <receiver android:name="OutgoingCallReceiver">
            <intent-filter>
                <action android:name=
                    "android.intent.action.NEW_OUTGOING_CALL"/>
            </intent-filter>
        </receiver>
    </application>
```

```
<uses-permission
    android:name="android.permission.PROCESS_OUTGOING_CALLS" />
<uses-sdk android:minSdkVersion="9" />
</manifest>
```

Теперь создайте в проекте новый класс с именем `OutgoingCallReceiver`, расширяющий класс `BroadcastReceiver`. В этом классе будет единственный перегруженный метод `onReceive()`, в котором будет обрабатываться событие инициализации исходящего звонка и показываться всплывающее уведомление.

Код класса `OutgoingCallReceiver` представлен в листинге 32.6.

Листинг 32.6. Файл класса окна приложения `OutgoingCallReceiver.java`

```
package com.samples.outgoingcall;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class OutgoingCallReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(final Context context, final Intent intent) {

        if (intent.getAction().equals(Intent.ACTION_NEW_OUTGOING_CALL)) {
            String phoneNumber =
                intent.getExtras().getString(Intent.EXTRA_PHONE_NUMBER);
            Toast.makeText(context, "Outgoing call: " + phoneNumber,
                Toast.LENGTH_LONG).show();
        }
    }
}
```

Скомпилируйте проект и запустите его в эмуляторе Android. Далее в наборной панели сделайте вызов абонента 5556 (второго экземпляра эмулятора Android). Наше приложение, работающее в фоновом режиме, должно перехватить исходящий вызов и отобразить на экране всплывающее уведомление с номером вызываемого абонента, как показано на рис. 32.9.

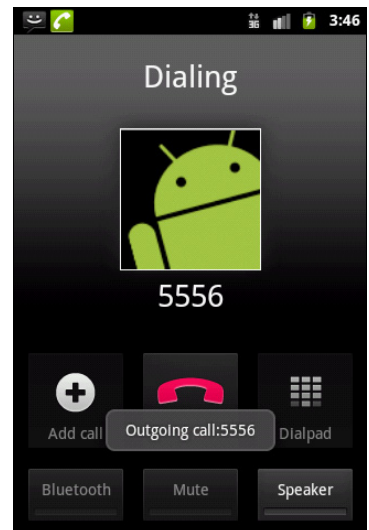


Рис. 32.9. Сообщение о перехвате входящего вызова приложением

Резюме

В этой главе мы рассмотрели возможности платформы Android для инициализации телефонных вызовов абонента из кода приложения. Как вы можете убедиться, системные компоненты Android предоставляют широкие возможности для создания приложений, работающих с входящими и исходящими вызовами, поэтому такую функциональность следует использовать с осторожностью и тщательно тестировать перед развертыванием на мобильные устройства.

В следующей главе мы будем анализировать возможности, предоставляемые платформой Android, для программной работы с SMS-сообщениями и рассмотрим способы отправки и получения SMS-сообщений напрямую из вашего приложения.



ГЛАВА 33

Отправка и получение SMS

Технология отправки и приема коротких сообщений Short Message Service (SMS) является важным средством коммуникации для мобильных устройств. SMS может использоваться как для передачи простых текстовых сообщений, так и для передачи небольших объемов данных по мобильным сетям.

В этой главе мы будем использовать функциональность Telephony API для программного управления передачей и приемом SMS-сообщений. Система Android также предлагает полный программный доступ к функциональным возможностям SMS, позволяя отправлять и получать SMS-сообщения напрямую из ваших приложений.

Использование эмулятора для отправки SMS

При разработке приложений для работы с SMS необязательно для отладки и тестирования программы использовать реальное мобильное устройство. Эмулятор Android предоставляет требуемую функциональность.

Для моделирования отправки сообщения SMS на эмуляторе мобильного устройства можно использовать те же способы, что и для имитации телефонных вызовов, описанных в предыдущей главе.

Для отправки текстовых сообщений из DDMS в представлении **Emulator Control** в группе **Telephony Actions** в поле **Incoming Number** введите абонентский номер запущенного экземпляра эмулятора: 5554. Затем поставьте переключатель **Voice/SMS** в режим **SMS** и напишите текст сообщения в поле **Message**, как показано на рис. 33.1.

В эмуляторе мобильного устройства должно появиться уведомление о приходе SMS. Если открыть это уведомление, можно прочитать наше SMS, отправленное из представления **Emulator Control** (рис. 33.2).

Для отправки SMS от одного эмулятора другому запустите еще один экземпляр эмулятора и откройте на нем приложение **Messaging**. Далее выберите опцию **New message** для создания SMS-сообщения. В верхнем текстовом поле укажите номер порта эмулятора Android, на который будет отправлено сообщение (5554). Напишите текст SMS-сообщения в нижнем текстовом поле **Message** и отправьте его на эмулятор кнопкой **Send**. На первый эмулятор Android с номером 5554 должно прийти SMS-сообщение от эмулятора 5556 (рис. 33.3).

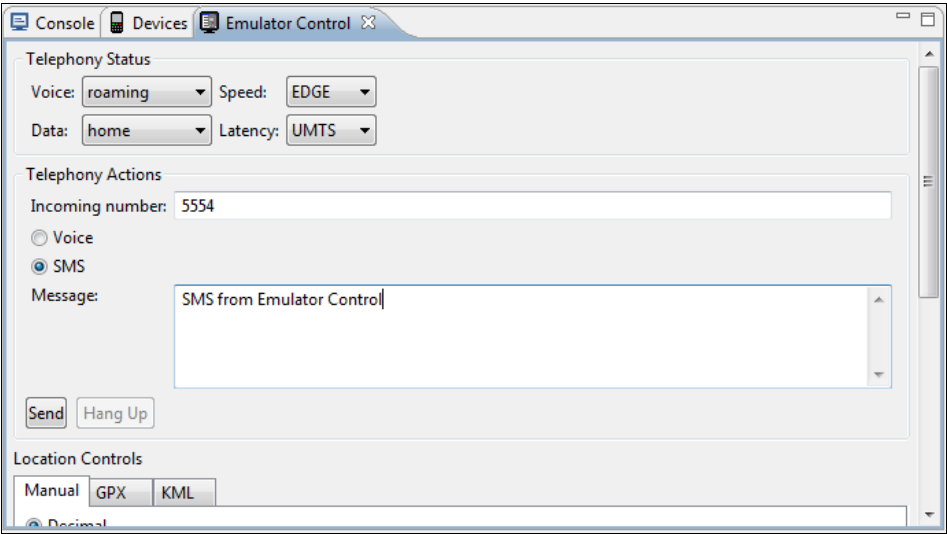


Рис. 33.1. Имитация отправки SMS из представления Emulator Control

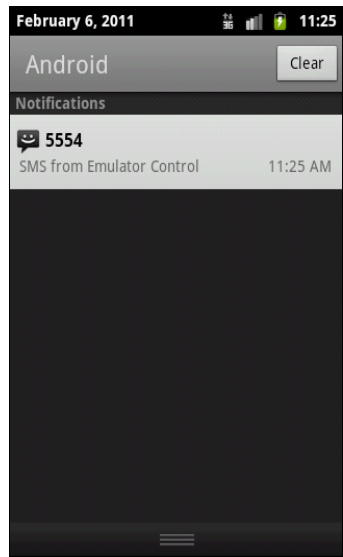


Рис. 33.2. SMS-сообщение, полученное эмулятором Android

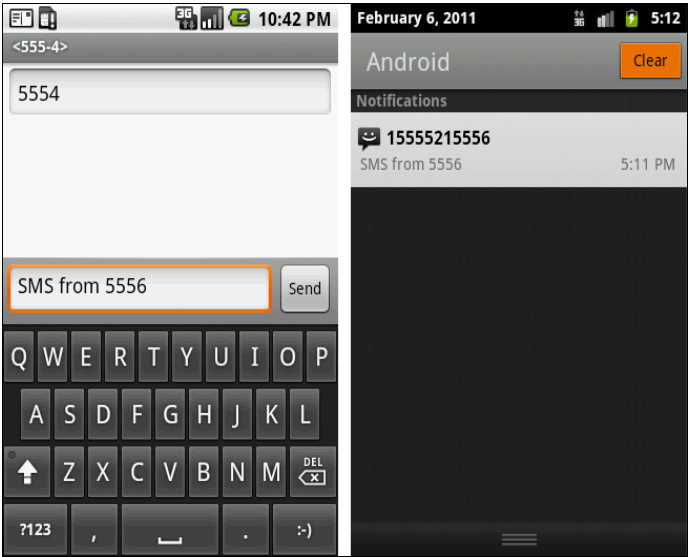


Рис. 33.3. Отправка SMS-сообщения между двумя эмуляторами

Отправка SMS из приложения

Библиотека Android SDK обеспечивает полные функциональные возможности программной отправки SMS из приложений с помощью класса `SmsManager`. Используя класс `SmsManager`, можно посылать текстовые сообщения, реагировать на входящие SMS и по-

лучать доступ к полям SMS-сообщений. В классе `SmsManager` предусмотрены методы для отправки SMS-сообщений разного типа.

В отличие от класса `TelephonyManager`, который мы рассматривали в предыдущих главах, для создания экземпляра класса `SmsManager` используется статический метод `getDefault()`, объявленный в этом же классе:

```
SmsManager manager = SmsManager.getDefault();
```

После создания экземпляра класса `SmsManager` можно с его помощью отправлять SMS-сообщения.

Для отправки текстовых сообщений используется метод `sendTextMessage()`. Этому методу необходимо передать пять параметров:

- ❑ `destinationAddress` — строка с адресом получателя SMS-сообщения;
- ❑ `scAddress` — строка с адресом сервисного центра SMS;
- ❑ `parts` — текст сообщения: объект `ArrayList<string>` или `String`, содержащий текст исходящего сообщения;
- ❑ `sentIntent` — объект `PendingIntent` для получения ответа, что сообщение было отправлено с мобильного устройства;
- ❑ `deliveryIntent` — объект `PendingIntent` для получения ответа, что сообщение было доставлено получателю.

В параметре `sentIntent` будет возвращен результирующий код со значением `Activity.RESULT_OK` в случае успешной отправки сообщения или один из кодов ошибки:

- ❑ `RESULT_ERROR_NO_SERVICE` — сервис SMS в данный момент недоступен;
- ❑ `RESULT_ERROR_NULL_PDU` — не поддерживается PDU (Protocol Description Unit, протокол передачи SMS-сообщений);
- ❑ `RESULT_ERROR_RADIO_OFF` — передатчик мобильного устройства выключен (например, включен режим `Airplane Mode`);
- ❑ `RESULT_ERROR_GENERIC_FAILURE` — какая-либо другая причина ошибки при отправке SMS.

Если результат отправки SMS-сообщения не важен, для параметров `sentIntent` или `deliveryIntent` можно передать `null`.

Чтобы отправить SMS-сообщение, сначала необходимо создать экземпляр класса `SmsManager` для управления отправкой SMS и, если требуется получать результаты отправки SMS, создать объект `PendingIntent`. Затем надо вызвать для объекта `SmsManager` метод `sendTextMessage()`. Вот возможный код для отправки SMS-сообщения из приложения:

```
public class SendSmsActivity extends Activity {  
    ...  
    SmsManager manager = SmsManager.getDefault();  
    PendingIntent intent = PendingIntent.getActivity(  
        this, 0, new Intent(this, SendSmsActivity.class), 0);  
  
    // Задаем номер получателя  
    String destinationAddress = "5556";
```

```
// Задаем текст сообщения
String smsText = "SMS from 5554"
// Отправляем сообщение
manager.sendTextMessage(
    destinationAddress, null, smsText, intent, null);
...
}
```

Отправка SMS с данными

Для отправки SMS с данными используется метод `sendDataMessage()`. Этому методу также необходимо передать набор из пяти параметров, но они имеют некоторые отличия от набора параметров для метода `sendTextMessage()`. Первые два параметра — строки `destinationAddress` и `scAddress` — такие же, как и в методе `sendDataMessage()`. Остальные три имеют следующее назначение:

- ☐ `destinationPort` — номер порта для доставки сообщения;
- ☐ `data` — тело сообщения, представленное в виде массива байтов;
- ☐ `sentIntent` — объект типа `PendingIntent` для получения результата отправки сообщения.

Деление SMS на фрагменты

При отправке SMS всегда проверяется количество символов в сообщении. Длина SMS не должна превышать 160 символов. Если она превышает максимальную длину, сообщение необходимо разбить на фрагменты длиной, не превышающей максимальную.

Для передачи таких сообщений используется метод `sendMultipartTextMessage()`. Этот метод отличается от `sendTextMessage()` тем, что входные параметры имеют типы `ArrayList<string>` для тела сообщения и `ArrayList<PendingIntent>` для параметров `sentIntent` и `deliveryIntent`.

С помощью метода `sendMultipartTextMessage()` можно передавать сразу весь пакет фрагментов SMS-сообщения. Если в методе использовать параметры типа `PendingIntent`, то можно контролировать успешную передачу для каждого фрагмента SMS-сообщения.

Установка разрешений для работы SMS

Для использования возможности работы с SMS в приложении требуется установка трех различных разрешений в зависимости от того, посылает приложение SMS или получает его:

- ☐ `RECEIVE_SMS` — разрешает приложению отслеживать прием входящих сообщений;
- ☐ `WRITE_SMS` — разрешает приложению создавать исходящее сообщение;
- ☐ `SEND_SMS` — разрешает приложению отправлять сообщение.

Например, для установки разрешения на отправку SMS необходимо записать в файл манифеста приложения `AndroidManifest.xml` следующий код:

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

Приложение для отправки SMS

Рассмотрим теперь создание приложения, способного отправить SMS-сообщение заданному абоненту. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — SmsSender;
- ☐ **Application name** — Send SMS;
- ☐ **Package name** — com.samples.telephony.sendsms;
- ☐ **Create Activity** — SendSmsActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch33_SmsSender.

В файле манифеста приложения AndroidManifest.xml необходимо установить разрешения для возможности написания и отправки SMS-сообщений из приложения: `WRITE_SMS` и `SEND_SMS`. Код файла манифеста приложения представлен в листинге 33.1.

Листинг 33.1. Файл манифеста приложения AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.telephony.sendsms"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity
            android:name=".SendSmsActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.WRITE_SMS" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-sdk android:minSdkVersion="10" />
</manifest>
```

В файле компоновки окна приложения main.xml создайте следующие виджеты:

- ☐ редактируемое текстовое поле `EditText` с идентификатором `number` для ввода номера абонента;
- ☐ редактируемое текстовое поле `EditText` с идентификатором `text` для текста сообщения;

- ❑ командную кнопку `Button` с надписью **Send** и идентификатором `bSend` для отправки SMS;
- ❑ два текстовых поля `TextView` с надписями **Number:** и **Text:**.

Код файла компоновки окна приложения `main.xml` представлен в листинге 33.2.

Листинг 33.2. Файл компоновки окна приложения `main.xml`

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

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Number:"
            android:layout_margin="5px"
            android:textStyle="bold"/>
        <EditText
            android:id="@+id/number"
            android:layout_width="150dip"
            android:layout_height="wrap_content"
            android:text=""
            android:layout_margin="5px"/>
        <Button
            android:id="@+id/bSend"
            android:layout_height="wrap_content"
            android:layout_margin="5px"
            android:layout_width="wrap_content"
            android:text="Send"/>
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="fill_parent">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Text:"
            android:layout_margin="5px"
            android:textStyle="bold"/>
        <EditText
            android:id="@+id/text"
            android:text=""
```

```
        android:layout_width="fill_parent"
        android:layout_margin="5px"
        android:layout_height="fill_parent"/>
</LinearLayout>
```

```
</LinearLayout>
```

В классе окна приложения `SendSmsActivity` в обработчике события `onClick()` кнопки сначала получаем объект `intent` вызовом метода `PendingIntent.getActivity()`, который затем передаем в качестве параметра методу `sendTextMessage()`.

Код класса `SendSmsActivity` представлен в листинге 33.3.

Листинг 33.3. Файл класса главного окна приложения `SendSmsActivity.java`

```
package com.samples.telephony.sendsms;

import android.app.Activity;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.PhoneNumberUtils;
import android.telephony.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class SendSmsActivity extends Activity implements OnClickListener {

    private EditText textSms;
    private EditText textNumber;
    private Button bSend;
    private SmsManager manager;
    private PendingIntent intent;
    @Override
    public void onCreate(final Bundle icle) {

        super.onCreate(icle);
        this setContentView(R.layout.main);

        textNumber = (EditText) findViewById(R.id.number);
        textSms = (EditText) findViewById(R.id.text);
        bSend = (Button) findViewById(R.id.bSend);

        manager = SmsManager.getDefault();
        bSend.setOnClickListener(this);
    }
}
```

```
@Override
public void onClick(View arg0) {
    String dest = textNumber.getText().toString();

    try {
        if (PhoneNumberUtils.isWellFormedSmsAddress(dest)) {
            intent = PendingIntent.getActivity(
                this, 0, new Intent(this, SendSmsActivity.class), 0);

            manager.sendTextMessage("5556", null,
                textSms.getText().toString(), intent, null);

            Toast.makeText(SendSmsActivity.this,
                "SMS sent", Toast.LENGTH_LONG).show();
        }
        else {
            Toast.makeText(SendSmsActivity.this,
                "Error formed SMS adress", Toast.LENGTH_LONG).show();
        }
    }
    catch (Exception e) {
        Toast.makeText(SendSmsActivity.this, e.toString(),
            Toast.LENGTH_LONG).show();
    }
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Протестируйте приложение: в текстовое поле **Number** введите номер абонента (в нашем случае это 5556), напишите текст сообщения в поле **Text** и отправьте его кнопкой **Send**. Сообщение должно прийти на второй эмулятор, как показано на рис. 33.4.

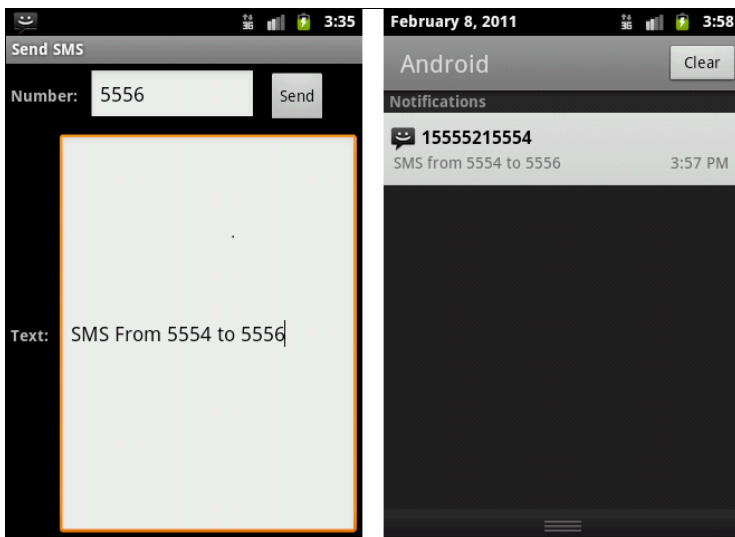


Рис. 33.4. Отправка SMS из приложения

Структура SMS-сообщения

Структуру SMS-сообщения определяет класс `SmsMessage`. Этот класс имеет множество методов для доступа к полям SMS-сообщений:

- ❑ `getMessageBody()` — возвращает тело текстового SMS-сообщения в виде строки;
- ❑ `getDisplayMessageBody()` — возвращает тело SMS- или E-mail-сообщения. Для SMS возвращаемое значение будет таким же, как и для метода `getMessageBody()`;
- ❑ `getOriginatingAddress()` — возвращает адрес отправителя сообщения или `null`, если он недоступен;
- ❑ `getDisplayOriginatingAddress()` — возвращает адрес отправителя SMS или E-mail. В случае SMS-сообщения возвращаемое значение будет таким же, как и для метода `getOriginatingAddress()`;
- ❑ `getServiceCenterAddress()` — возвращает адрес сервисного центра SMS;
- ❑ `getIndexOnIcc()` — возвращает индекс записи сообщения на ICC (Integrated Circuit Card, т. е. на SIM-карте);
- ❑ `getMessageClass()` — возвращает класс сообщения;
- ❑ `getPdu()` — возвращает сообщение формата PDU (Protocol Data Unit) в виде массива байтов. PDU — это протокол передачи SMS-сообщений в сетях GSM;
- ❑ `getProtocolIdentifier()` — возвращает Protocol Identifier, идентификатор протокола PDU. Этот идентификатор используется при маршрутизации SMS-сообщения;
- ❑ `getStatus()` — возвращает сведения о состоянии SMS из SMS-STATUS-REPORT (это элемент протокола PDU);
- ❑ `getTimestampMillis()` — возвращает timestamp сервисного центра SMS в формате `currentTimeMillis()`;
- ❑ `getUserData()` — возвращает часть сообщения с пользовательскими данными без заголовка сообщения;
- ❑ `getStatusOnIcc()` — возвращает статус сообщения на ICC (на SIM-карте). Это значение определяется константами, объявленными в классе `SmsManager`:
 - `STATUS_ON_ICC_READ` — сообщение было получено, открыто и прочитано пользователем;
 - `STATUS_ON_ICC_UNREAD` — сообщение получено, но еще не было открыто;
 - `STATUS_ON_ICC_SEND` — сообщение отправлено;
 - `STATUS_ON_ICC_UNSENT` — сообщение не отправлено;
 - `STATUS_ON_ICC_FREE` — место, занимаемое сообщением на SIM-карте, может быть освобождено для записи другой, более важной информации.

Как вы видите, структура SMS достаточно сложная и содержит не только адрес, текст сообщения и время отправки и получения, а также довольно много других полей, характеризующих это сообщение.

Перехват входящих SMS-сообщений приложением

При необходимости можно создать приложение, реагирующее на получение мобильным устройством SMS-сообщения и обрабатывающее сообщения в теле программы. Для перехвата входящего SMS-сообщения используют объект `BroadcastReceiver`. При получении мобильным устройством SMS-сообщения система генерирует `Broadcast Intent`.

Компонент `Broadcast Receiver` в нашем приложении должен перехватить сгенерированный объект `Broadcast Intent` и обработать его, получив доступ к содержащейся в нем информации. Действие для этого `Broadcast Intent` определено в классе `android.provider.Telephony` следующей константой:

```
public static final String SMS_RECEIVED_ACTION =  
    "android.provider.Telephony.SMS_RECEIVED"
```

Правда, есть одна проблема: класс `android.provider.Telephony` не является частью `Public API Android SDK`, и его нельзя напрямую использовать в приложении. Однако можно использовать для определения действия `SMS_RECEIVED` непосредственно строку:

```
String action = "android.provider.Telephony.SMS_RECEIVED"
```

Действие `SMS_RECEIVED_ACTION` содержит один extra-параметр — `pdu`, представляющий собой массив объектов `pdu` — `Protocol Description Unit` (протокол передачи SMS-сообщений). `Protocol Description Unit` содержит информацию о SMS-сообщении и состоит из большого количества полей. Однако непосредственно с ним не работают, его надо предварительно преобразовать в удобочитаемую форму. Для этого используется класс `SmsMessage`, который содержит статический метод `createFromPdu()` и возвращает объект класса `SmsMessage`:

```
SmsMessage smsMessage = SmsMessage.createFromPdu((byte[]) pdu[i]);
```

Полученный объект `smsMessage` содержит множество полей, характеризующих поля SMS-сообщения, например адрес отправителя, тело сообщения, имя сетевого провайдера, используемый протокол и т. д. Доступ к этим полям можно получить через набор методов `get...()` этого класса, например:

```
String address = smsMessage.getOriginatingAddress();  
String messageBody = smsMessage.getMessageBody();
```

Сейчас мы разработаем приложение, содержащее единственный компонент `Broadcast Receiver`. Наше приложение при получении SMS будет выводить на экран всплывающее уведомление о полученном SMS и информацию, хранящуюся в полях этого сообщения.

Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `SMSReceiver`;
- ☐ **Application name** — `SMS Receiver`;
- ☐ **Package name** — `com.samples.telephony.receiversms`;

- ❑ **Create Activity** — оставляем незаполненным, т. к. мы создаем приложение без графического интерфейса.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch33_SmsReceiver.

В файле манифеста приложения `AndroidManifest.xml` обязательно добавьте разрешения `android.permission.RECEIVE_SMS` и `android.permission.READ_SMS` для возможности получения и чтения входящего SMS-сообщения. Кроме того, создайте также элемент `<receiver>` с атрибутом `android:name="SmsReceiver"` для нашего приемника SMS-сообщений и вложенный элемент `<intent-filter>` — фильтр принимаемых объектов `Intent` с атрибутом `android:name="android.provider.Telephony.SMS_RECEIVED"`.

Файл манифеста приложения представлен в листинге 33.4.

Листинг 33.4. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.telephony.receiversms"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <receiver android:name="SmsReceiver">
            <intent-filter>
                <action
                    android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
    </application>

    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
</manifest>
```

В проект добавим новый `java`-класс, который назовем `SmsReceiver`. Этот класс будет расширением класса `BroadcastReceiver`, который является базовым классом для компонентов `Broadcast Receiver`. В классе `SmsReceiver` определим метод `onReceive`, наследуемый от базового класса `BroadcastReceiver`, в котором будет выводиться всплывающее уведомление о входящем SMS.

Полный код класса `SmsReceiver` приведен в листинге 33.5.

Листинг 33.5. Файл класса `SmsReceiver.java`

```
package com.samples.telephony.receiversms;

import android.content.BroadcastReceiver;
import android.content.Context;
```

```

import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.widget.Toast;

public class SmsReceiver extends BroadcastReceiver {

    private static final String SMS_REC_ACTION =
        "android.provider.Telephony.SMS_RECEIVED";

    @Override
    public void onReceive(final Context context, final Intent intent) {
        if (intent.getAction().equals(SmsReceiver.SMS_REC_ACTION)) {

            StringBuilder sb = new StringBuilder();

            Bundle bundle = intent.getExtras();
            if (bundle != null) {
                Object[] pdu = (Object[]) bundle.get("pdu");
                for (Object pdu : pdu) {
                    SmsMessage smsMessage =
                        SmsMessage.createFromPdu((byte[]) pdu);

                    sb.append("\nAddress: " +
                        smsMessage.getOriginatingAddress());
                    sb.append("\nDisplay Originating: " +
                        smsMessage.getDisplayOriginatingAddress());
                    sb.append("\nDisplay MessageBody: " +
                        smsMessage.getDisplayMessageBody());
                    sb.append("\nMessage Body: " +
                        smsMessage.getMessageBody());
                    sb.append("\nMessage Class: " +
                        smsMessage.getMessageClass());
                    sb.append("\nProtocol Identifier: " +
                        smsMessage.getProtocolIdentifier());
                    sb.append("\nPseudo Subject: " +
                        smsMessage.getPseudoSubject());
                    sb.append("\nService Center Address: " +
                        smsMessage.getServiceCenterAddress());
                    sb.append("\nStatus: " +
                        smsMessage.getStatus());
                    sb.append("\nStatus On ICC: " +
                        smsMessage.getStatusOnIcc());
                }
            }

            Toast.makeText(context, "SMS Received message" + sb.toString(),
                Toast.LENGTH_LONG).show();
        }
    }
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. При получении SMS-сообщения на экран мобильного устройства будет выведено всплывающее уведомление, в котором будет информация о структуре и содержимом этого сообщения. Внешний вид приложения с уведомлением представлен на рис. 33.5.

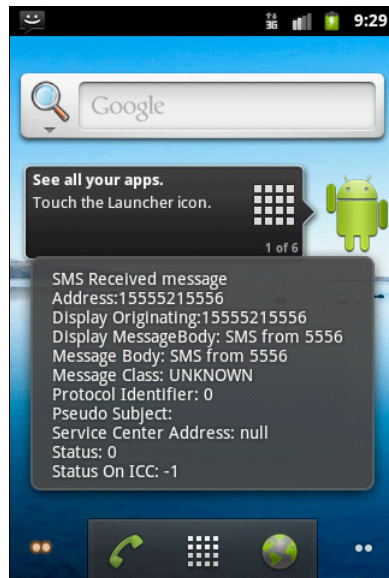


Рис. 33.5. Перехват входящего SMS

Хранение SMS на мобильном устройстве

Все SMS-сообщения, созданные и отправленные на данном мобильном устройстве, а также полученные от других абонентов, сохраняются в локальной базе SMS-сообщений. Эти сообщения, в зависимости от их типа, распределены по определенным каталогам, имеющим стандартные названия и URI.

Вот полный список каталогов SMS-сообщений и URI для каждого каталога SMS:

- ❑ Inbox: `content://sms/inbox;`
- ❑ Sent: `content://sms/sent;`
- ❑ Draft: `content://sms/draft;`
- ❑ Outbox: `content://sms/outbox;`
- ❑ Failed: `content://sms/failed;`
- ❑ Queued: `content://sms/queued;`
- ❑ Undelivered: `content://sms/undelivered;`
- ❑ Conversations: `content://sms/conversations.`

Названия каталогов говорят сами за себя, и нет смысла их описывать. Далее мы рассмотрим доступ к каталогам SMS из программного кода.

Доступ к каталогам SMS

Чтобы получить доступ к SMS-сообщениям, необходимо использовать URI, указывающий на нужный каталог. Например, создать объект `Cursor` и выполнить запрос к каталогу Inbox для входящих SMS можно следующим образом:

```
Uri uri = Uri.parse("content://sms/inbox");
// Создаем объект Cursor, используя запрос без параметров
Cursor cursor = getContentResolver().query(uri, null, null, null, null);
startManagingCursor(cursor);
```

Этот простой запрос без параметров вернет все содержимое указанного каталога.

Чтобы получить доступ к каталогу SMS, необходимо также использовать разрешение `READ_SMS`:

```
<uses-permission android:name="android.permission.READ_SMS"/>
```

Теперь, используя все описанные ранее приемы, создадим приложение для просмотра каталогов SMS-сообщений, сохраненных на мобильном устройстве. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `SmsFoldersInfo`;
- ☐ **Application name** — `SMS folders info`;
- ☐ **Package name** — `com.samples.telephony.smsfoldersinfo`;
- ☐ **Create Activity** — `SmsFolderInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch33_SmsFoldersInfo`.

В файле манифеста приложения `AndroidManifest.xml` необходимо прописать разрешение `android.permission.READ_SMS` для доступа к каталогам SMS. Кроме того, в приложении будут два окна, и нам необходимо объявить соответственно два объекта `Activity`. В главном (`SmsFoldersListActivity`) будет находиться список каталогов SMS-сообщений, второе (`SmsFolderInfoActivity`) будет отображать содержимое полей выбранного каталога SMS.

Код файла манифеста приложения представлен в листинге 33.6.

Листинг 33.6. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.telephony.smsfoldersinfo"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".SmsFoldersListActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```

        <activity android:name=".SmsFolderInfoActivity"
            android:label="@string/app_name">
        </activity>
    </application>

    <uses-permission android:name="android.permission.READ_SMS"/>
</manifest>

```

Файл компоновки для главного окна приложения не нужен, т. к. оно будет построено на основе системного ресурса `android.R.layout.simple_list_item_1` и будет представлять собой меню в виде списка.

Класс `SmsFoldersListActivity` нужен лишь для отображения списка каталогов SMS-сообщений и выбора элемента списка. Код класса представлен в листинге 33.7.

Листинг 33.7. Файл класса главного окна приложения `SmsFoldersListActivity.java`

```

package com.samples.telephony.smsfoldersinfo;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class SmsFoldersListActivity extends ListActivity {

    // Массив строк, представляющий каталоги SMS
    private String[] folders = {
        "inbox", "sent", "draft", "outbox", "failed",
        "queued", "undelivered", "conversations"
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, folders));
    }

    public void onItemClick(AdapterView<?> parent, View v, int pos, long id) {
        Intent intent = new Intent(getApplicationContext(),
            SmsFolderInfoActivity.class);
        // Формируем URI, представляющий каталог SMS
        intent.putExtra("uri", "content://sms/" + folders[pos]);
        this.startActivity(intent);
    }
}

```

Содержимое каталога будет отображаться в виде списка SMS, состоящего из двух полей: номера абонента, пославшего SMS, и текста сообщения. Для отображения этого списка создадим файл компоновки с именем `row.xml`, который будет представлять собой строку этого списка, состоящую из двух колонок.

Код файла компоновки `row.xml` представлен в листинге 33.8.

Листинг 33.8. Файл компоновки для строки списка SMS-сообщений `row.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/address"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:layout_marginLeft="5px"/>

    <TextView
        android:id="@+id/body"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="18sp"
        android:paddingRight="10px"
        android:layout_marginLeft="5px"/>

</LinearLayout>
```

В коде класса `SmsFolderInfoActivity` с помощью объекта `Cursor` мы будем выводить SMS-сообщения, содержащиеся в выбранном каталоге. Код класса представлен в листинге 33.9.

Листинг 33.9. Файл класса `SmsFolderInfoActivity.java`

```
package com.samples.telephony.smsfoldersinfo;

import android.app.ListActivity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.widget.ListAdapter;
import android.widget.SimpleCursorAdapter;

public class SmsFolderInfoActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```



```

// Получаем URI выбранного каталога
Bundle extras = this.getIntent().getExtras();
Uri uri = Uri.parse(extras.getString("uri"));
this.setTitle(uri.toString());

// Создаем объект Cursor, делая запрос к базе данных
Cursor cursor =
    getContentResolver().query(uri, null, null, null, null);
startManagingCursor(cursor);

String[] columns = new String[] { "address", "body" };
int[] rows = new int[] { R.id.address, R.id.body };

ListAdapter adapter = new SimpleCursorAdapter(
    this, R.layout.row, cursor, columns, rows);
setListAdapter(adapter);
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. У нас получилось приложение для навигации по каталогам SMS. С помощью этой программы мы можем заходить в любой каталог и просматривать сохраненные там SMS-сообщения и адреса отправителей.

Внешний вид нашего приложения для просмотра SMS представлен на рис. 33.6.

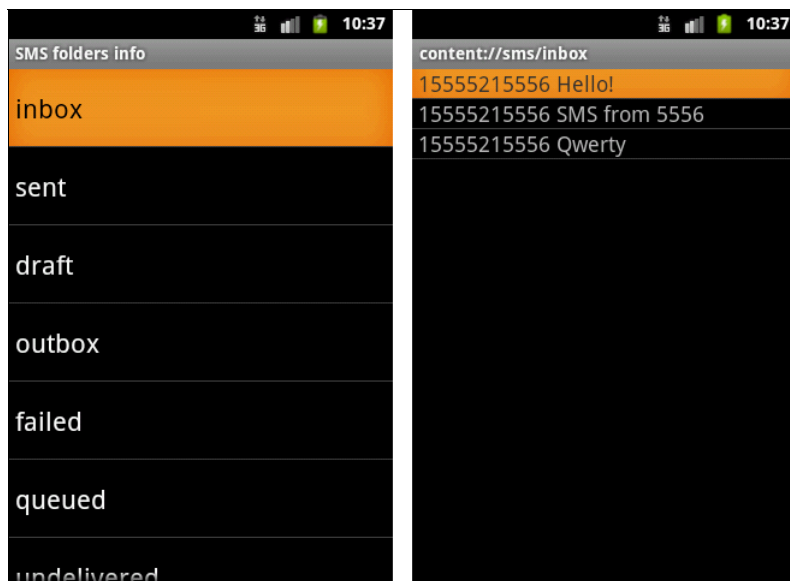


Рис. 33.6. Список SMS из папки **inbox**

Доступ к полям SMS-сообщения

В предыдущем примере приложения для просмотра содержимого SMS мы использовали только два поля: `address` и `body`. На самом деле полей, определяющих структуру SMS-сообщения, гораздо больше. Эти поля не документированы в Android SDK и не являются частью `public API` библиотеки Android.

Поля для таблицы текстовых SMS-сообщений объявлены в интерфейсе `android.provider.Telephony.TextBasedSmsColumns`. Вот их полный перечень и описание:

- ❑ `_id` — идентификатор записи в таблице SMS базы данных SMS- и MMS-сообщений;
- ❑ `thread_id` — идентификатор потока для сообщения;
- ❑ `address` — адрес абонента (его телефонный номер), пославшего сообщение;
- ❑ `person` — идентификатор отправителя SMS или `null`, если идентификатор отсутствует;
- ❑ `date` — дата отправки сообщения (тип `long`);
- ❑ `protocol` — идентификатор протокола SMS;
- ❑ `read` — флаг, указывающий, что данное сообщение прочитано пользователем;
- ❑ `status` — значение `TP-Status` для сообщения или `-1`, если `TP-Status` не был получен. `TP` — это сокращение от `Transfer Protocol` (протокол передачи);
- ❑ `type` — тип сообщения;
- ❑ `reply_path_present` — флаг, показывающий, что установлен `TP-Reply-Path` — протокол передачи обратного адреса;
- ❑ `subject` — тема сообщения, аналогичная полю `subject` при отправке E-mail;
- ❑ `body` — тело сообщения;
- ❑ `service_center` — имя сервисного центра SMS, через который отправлено сообщение;
- ❑ `locked` — флажок для обозначения заблокированного сообщения;
- ❑ `error_code` — код ошибки, 0 в случае, если ошибки нет.

Названия этих полей и, соответственно, данных, хранящихся в них, можно получить в приложении и обработать их, если есть в этом необходимость. Так как мы используем для доступа к базе данных SMS объект `Cursor`, можно при итерации по полям записи, представляющей SMS, вызвать метод `getColumnName()` класса `Cursor`.

Например, прочитать все названия полей таблицы SMS и их содержимое в коде приложения можно следующим образом:

```
Cursor cursor;
...
cursor = getContentResolver().query(uri, null, null, null, null);
startManagingCursor(cursor);
...
// Помещаем курсор на позицию,
// где находится нужное нам сообщение
cursor.moveToPosition(pos);
```

```
// Создаем экземпляр StringBuilder,
// куда будем сохранять полученные данные
StringBuilder data = new StringBuilder();
// Получаем имя поля и данные, хранящиеся в нем
for (int i = 0; i < cursor.getColumnCount(); i++) {
    data.append(cursor.getColumnName(i) + ":\t" +
        cursor.getString(i) + "\n");
}
```

Теперь усовершенствуем наше приложение для чтения каталогов SMS, созданное нами в предыдущем разделе, добавив в него диалоговое окно, в которое будем выводить детальную информацию о выбранном SMS-сообщении.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch33_SmsFoldersInfoDetails.

В код класса `SmsFolderInfoActivity` добавим обработчик события выбора элемента списка `onListItemClick()`, в котором мы будем читать поля выбранного SMS и содержимое этих полей.

Модифицированный код класса `SmsFolderInfoActivity` с необходимыми добавлениями представлен в листинге 33.10.

Листинг 33.10. Файл класса `SmsFolderInfoActivity.java`

```
package com.samples.telephony.smsfoldersdetailsinfo;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.DialogInterface;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class SmsFolderInfoActivity extends ListActivity {

    private Cursor cursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Получаем URI выбранного каталога
        Bundle extras = this.getIntent().getExtras();
        Uri uri = Uri.parse(extras.getString("uri"));
        this.setTitle(uri.toString());
    }
}
```

```

// Создаем объект Cursor, делая запрос к базе данных
cursor = getContentResolver().query(uri, null, null, null, null);
startManagingCursor(cursor);

String[] columns = new String[] { "address", "body" };
int[] rows = new int[] { R.id.address, R.id.body };

ListAdapter adapter = new SimpleCursorAdapter(
    this, R.layout.row, cursor, columns, rows);
setListAdapter(adapter);
}

public void onListItemClick(ListView parent, View v, int pos, long id)
{
    // Устанавливаем курсор в выбранную позицию
    cursor.moveToPosition(pos);
    StringBuilder data = new StringBuilder();

    // Читаем данные из объекта Cursor в StringBuilder
    for (int i = 0; i < cursor.getColumnCount(); i++) {
        data.append(cursor.getColumnName(i) + ":\t" +
            cursor.getString(i) + "\n");
    }

    // Создаем диалоговое окно для вывода данных полей
    // SMS-сообщения
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("SMS Details");
    builder.setMessage(data.toString());
    builder.setPositiveButton("OK",
        new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) { }
        });
    builder.show();
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. Теперь наше приложение способно выводить в диалоговое окно полное описание выбранного SMS-сообщения. Внешний вид приложения представлен на рис. 33.7.

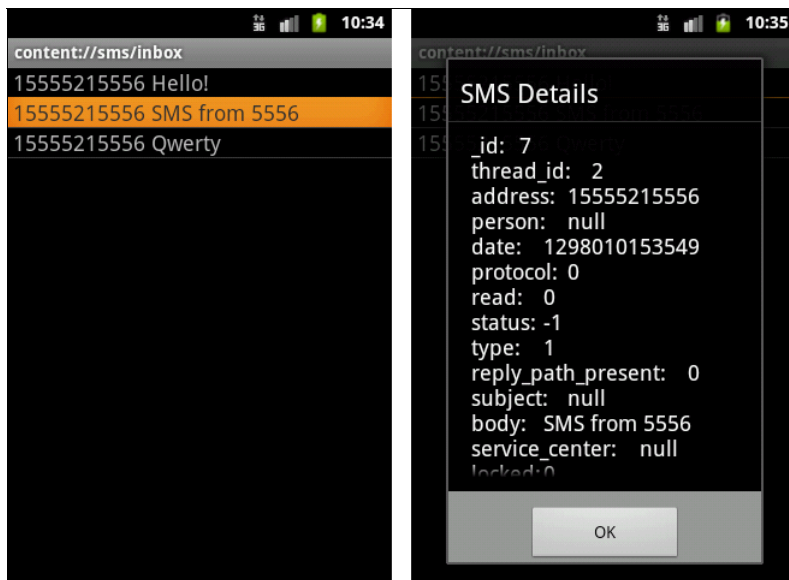


Рис. 33.7. Отображение детальной информации выбранного SMS

Резюме

Система Android предоставляет отличную функциональность для отправки и приема SMS-сообщений, что позволяет создать приложения, которые могут обмениваться информацией с другими мобильными устройствами и их пользователями.

Здесь мы рассмотрели способы отправки и получения SMS-сообщений с использованием классов `SmsManager` и `SmsMessage` из пакета `android.telephony`. Также была рассмотрена структура каталогов SMS-сообщений и программный доступ к базе данных SMS, хранящейся на мобильном устройстве.

На этом мы заканчиваем рассмотрение базовых функций мобильного телефона и переходим к работе с мобильным Интернетом.



ГЛАВА 34

Мобильный Интернет

Возможности мобильных устройств получать доступ к Интернету и работать с сетью точно так же, как и персональные компьютеры, обеспечивают все больший рост их популярности среди пользователей. Разработка сетевых приложений для платформы Android является очень актуальной темой.

Для создания таких приложений Android SDK предоставляет солидную функциональность в виде менеджера сетевых соединений, менеджера закачек и встроенного браузера WebKit и др. В этой главе мы научимся создавать сетевые приложения для работы через мобильный Интернет.

Создание сетевых соединений

Для работы с сетью на платформе Android доступны стандартные библиотеки Java, такие как `java.net`. Кроме того, платформа Android предоставляет собственные специализированные библиотеки: `android.net` и `android.net.http`, специфичные для работы с мобильными сетями.

Сначала мы рассмотрим способы подключения к мобильной сети, определение доступности сети, получение характеристик соединения и создание сетевых соединений в коде приложения.

Менеджер сетевых соединений

Для создания и управления сетевыми соединениями используется класс `ConnectivityManager` из пакета `android.net`. Кроме того, этот класс применяется для мониторинга изменений состояния сети.

Для создания экземпляра класса `ConnectivityManager` используется метод `getSystemService()` класса `Context` с параметром `Context.CONNECTIVITY_SERVICE`:

```
ConnectivityManager manager = (ConnectivityManager) getSystemService(  
    Context.CONNECTIVITY_SERVICE);
```

Для получения информации о сети в классе `ConnectivityManager` содержится набор методов:

- ☐ `getActiveNetworkInfo();`
- ☐ `getNetworkInfo(int networkType);`
- ☐ `getAllNetworkInfo();`

Эти методы, кроме `getAllNetworkInfo()`, возвращают объект типа `NetworkInfo`. Метод `getAllNetworkInfo()` возвращает массив объектов `NetworkInfo`. Объект `NetworkInfo` — это и есть основной класс, определяющий характеристики мобильной сети, которые мы рассмотрим подробнее в следующем разделе.

Характеристики мобильной сети

Класс `NetworkInfo` инкапсулирует параметры мобильной сети Интернет и содержит методы для определения параметров сети. Для определения типа сети применяется следующая группа методов:

- ❑ `getType()` — возвращает целочисленное значение, определяющее тип сети. Эти значения определены набором констант в классе `ConnectivityManager`: `TYPE_MOBILE` (мобильная сеть), `TYPE_WIFI` (сеть Wi-Fi) и т. д.;
- ❑ `getSubtype()` — возвращает целочисленное значение, определяющее тип подсети;
- ❑ `getTypeName()` — возвращает описательное имя типа сети в виде строки, например, "WIFI" или "MOBILE";
- ❑ `getSubtypeName()` — возвращает описательное имя типа подсети.

Для определения доступности сетевого соединения в классе `NetworkInfo` предусмотрен набор методов:

- ❑ `isAvailable()` — проверяет доступность сети. Возвращает `true`, если сеть доступна;
- ❑ `isConnectedOrConnecting()` — возвращает `true`, если сетевое соединение уже установлено или находится в процессе установления;
- ❑ `isConnected()` — возвращает `true`, если установлено сетевое соединение и возможна передача данных через него;
- ❑ `isRoaming()` — возвращает `true`, если для данной сети мобильное устройство находится в роуминге.

Этот класс необходимо использовать в любом приложении Android, которое работает с сетевыми соединениями.

Получение информации о сети в приложении

Чтобы проверить наличие и состояние мобильной сети, создадим приложение, использующее рассмотренные ранее в этой главе классы. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `NetworkStateInfo`;
- ❑ **Application name** — `Network state info`;
- ❑ **Package name** — `com.samples.network.networkstateinfo`;
- ❑ **Create Activity** — `NetworkStateInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch34_NetworkStateInfo`.

В файле манифеста приложения `AndroidManifest.xml` необходимо объявить разрешения `android.permission.ACCESS_NETWORK_STATE` и `android.permission.INTERNET` для доступа приложения к информации о мобильной сети.

Код файла манифеста приложения `AndroidManifest.xml` представлен в листинге 34.1.

Листинг 34.1. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.network.networkstateinfo"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".NetworkStateInfoActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission
        android:name="android.permission.INTERNET" />
    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
</manifest>
```

Код для файла компоновки окна приложения простой — он содержит только виджет `TextView` для вывода информации.

В коде программы мы получаем объект класса `ConnectivityManager`. Затем, используя вызов метода `getAllNetworkInfo()`, получаем массив объектов `NetworkInfo`, из которого получаем информацию о конкретной сети.

Код класса `NetworkStateInfoActivity` главного окна приложения представлен в листинге 34.2.

Листинг 34.2. Файл класса главного окна приложения `NetworkStateInfoActivity.java`

```
package com.samples.network.networkstateinfo;

import android.app.Activity;
import android.content.Context;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
```

```

import android.os.Bundle;
import android.widget.TextView;

public class NetworkStateInfoActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        TextView text = (TextView)findViewById(R.id.text);

        ConnectivityManager cm = (ConnectivityManager) getSystemService(
            Context.CONNECTIVITY_SERVICE);

        NetworkInfo[] ni = cm.getAllNetworkInfo();

        for (int i = 0; i < ni.length; i++) {
            text.append("Type:\t" + ni[i].getTypeName());
            text.append("\n\tAvailable:\t" + ni[i].isAvailable());
            text.append("\n\tConnected:\t" + ni[i].isConnected());
            text.append("\n\tExtra:\t" + ni[i].getExtraInfo() + "\n");
        }
    }
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. Внешний вид приложения и выводимая им информация о сетевых соединениях представлены на рис. 34.1.

Как вы можете убедиться из информации о сетевых соединениях, полученной приложением, эмулятор Android поддерживает 5 типов сетевых соединений, и все сети, кроме Wi-Fi, доступны. Если разрабатываемое вами приложение должно работать с сетью, его без проблем можно отлаживать на эмуляторе Android. Имитация сети Wi-Fi не поддерживается эмулятором, ее использование возможно только на реальном мобильном устройстве и будет рассмотрено в следующей главе.

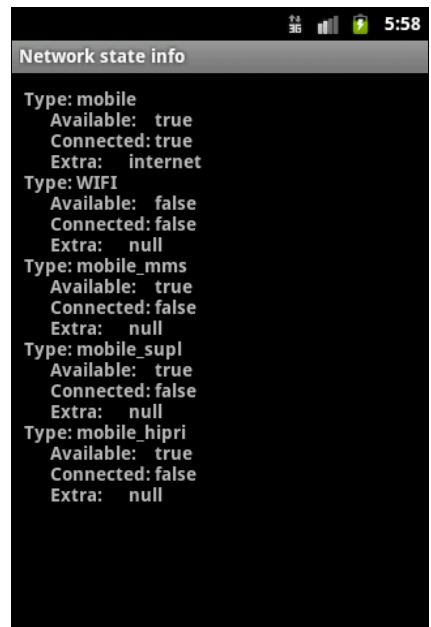


Рис. 34.1. Отображение информации о состоянии сети Интернет

Мониторинг сетевого трафика

Обычно услуги мобильного Интернета, предоставляемые операторами сотовой сети, являются платными, и их стоимость зависит от объема информации, переданной или полученной пользователем через мобильную сеть. Поэтому отслеживание трафика через мобильную сеть является актуальной темой.

Получение информации о трафике

Для мониторинга сетевого трафика в библиотеке Android SDK предусмотрен класс `TrafficStats` из пакета `android.net`. Методы класса `TrafficStats` позволяют приложению получать информацию о количестве байтов и пакетов, переданных или полученных через мобильную сеть.

Например, если требуется получить данные по трафику через мобильный Интернет, в классе `TrafficStats` используются следующие методы:

- ❑ `getMobileRxBytes()` — возвращает общее количество байтов, полученных через мобильный интерфейс с момента создания подключения;
- ❑ `getMobileRxPackets()` — возвращает общее количество пакетов, полученных через мобильный интерфейс;
- ❑ `getMobileTxBytes()` — возвращает общее количество байтов, переданных через мобильный интерфейс;
- ❑ `getMobileTxPackets()` — возвращает общее количество пакетов, переданных через мобильный интерфейс.

Класс `TrafficStats` позволяет также оценивать общий объем трафика, проходящий через все сетевые интерфейсы, если, например, кроме мобильного Интернета, использовалось соединение Wi-Fi.

Для мониторинга сетевого трафика в классе `TrafficStats` используется следующий набор методов:

- ❑ `getTotalRxBytes()` — возвращает общее количество байтов, полученных через все интерфейсы сети;
- ❑ `getTotalRxPackets()` — возвращает общее количество пакетов, полученных через все интерфейсы сети;
- ❑ `getTotalTxBytes()` — возвращает общее количество байтов, посланных через все интерфейсы сети;
- ❑ `getTotalTxPackets()` — возвращает общее количество пакетов, посланных через все интерфейсы сети.

К мобильному устройству может быть подключено и использоваться сразу несколько сетевых соединений, предоставляемых разными операторами. В классе `TrafficStats` есть набор методов для получения данных по трафику для каждой из сетей:

- ❑ `getUidRxBytes()` — возвращает число байтов, полученных через сеть с заданным UID;
- ❑ `getUidTxBytes()` — возвращает число байтов, посланных через сеть с заданным UID.

Этим методам в качестве входного параметра надо передавать идентификатор сетевого соединения.

Кроме определения общего количества байтов, переданных или полученных через соединение, существует также набор методов, определяющих общее количество пакетов, но эти методы применяются редко, т. к. фактором, определяющим стоимость сетевого трафика оператора мобильной сети, обычно является общее количество байт, проходящих через сетевое соединение.

Приложение для мониторинга сетевого трафика

Сейчас мы создадим приложение, использующее возможности класса `TrafficStats` для мониторинга сетевого трафика. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `TrafficStatsInfo`;
- ☐ **Application name** — `Network Traffic info`;
- ☐ **Package name** — `com.samples.network.trafficstatsinfo`;
- ☐ **Create Activity** — `TrafficStatsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch34_TrafficStatsInfo`.

В файле манифеста приложения `AndroidManifest.xml` объявите разрешения `android.permission.ACCESS_NETWORK_STATE` и `android.permission.INTERNET` для доступа приложения к информации о мобильной сети (так же как и в предыдущем приложении, см. листинг 34.1).

В коде класса `TrafficStatsActivity` мы воспользуемся методами класса `TrafficStats`. Код класса `TrafficStatsActivity` главного окна приложения представлен в листинге 34.3.

Листинг 34.3. Файл класса главного окна приложения `TrafficStatsActivity.java`

```
package com.samples.network.trafficstatsinfo;

import android.app.Activity;
import android.net.TrafficStats;
import android.os.Bundle;
import android.widget.TextView;

public class TrafficStatsActivity extends Activity {

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

        final TextView text = (TextView) findViewById(R.id.text);

        text.append("Total:");
        text.append("\n\tRX Bytes:\t" + TrafficStats.getTotalRxBytes());
```

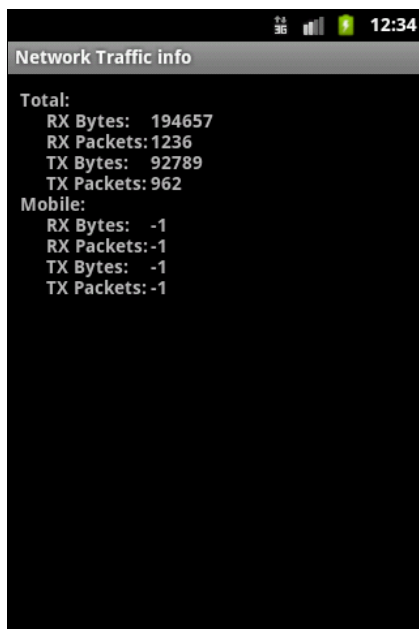
```
text.append("\n\tRX Packets:\t" +
    TrafficStats.getTotalRxPackets());
text.append("\n\tTX Bytes:\t" + TrafficStats.getTotalTxBytes());
text.append("\n\tTX Packets:\t" +
    TrafficStats.getTotalTxPackets());

text.append("\nMobile:");
text.append("\n\tRX Bytes:\t" + TrafficStats.getMobileRxBytes());
text.append("\n\tRX Packets:\t" +
    TrafficStats.getMobileRxPackets());
text.append("\n\tTX Bytes:\t" + TrafficStats.getMobileTxBytes());
text.append("\n\tTX Packets:\t" +
    TrafficStats.getMobileTxPackets());
}
```

Выполните компиляцию проекта и запустите его на эмуляторе Android. Внешний вид приложения и выводимая в него информация представлены на рис. 34.2.

Если у вас на компьютере есть сетевое подключение, можете в эмуляторе запустить встроенный браузер и походить по ссылкам, а затем снова запустить приложение, которое покажет уже другие результаты по переданным данным.

Рис. 34.2. Мониторинг сетевого трафика в приложении



Встроенный браузер

Платформа Android комплектуется встроенным браузером WebKit. Этот браузер обеспечивает отображение веб-страниц на устройствах Android с учетом особенностей, присущих мобильным устройствам, которые существенно отличаются от требований, предъявляемых к персональным компьютерам.

На мобильных устройствах в первую очередь имеет большое значение размер и разрешение экрана, хотя уже многие сайты адаптированы под просмотр на мобильных устройствах. На рис. 34.3 представлен пример отображения веб-страницы в браузере мобильного устройства.

При загрузке веб-страниц в браузер WebKit содержимое страницы, не адаптированной под мобильные устройства, масштабируется и отображается так, чтобы вся страница целиком помещалась на экране. Браузер позволяет такую страницу прокручивать и увеличивать масштаб, обеспечивая доступ к ее содержимому.

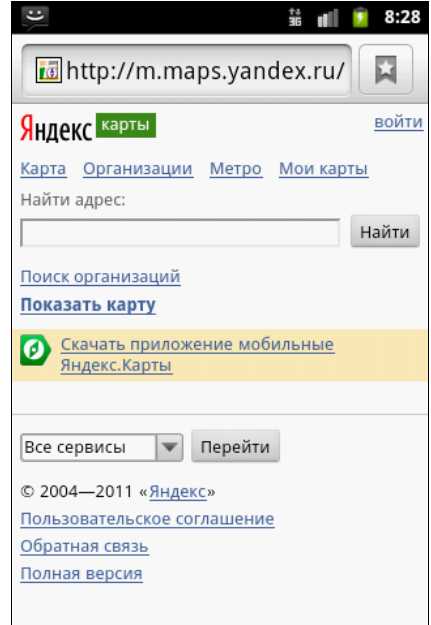


Рис. 34.3. Встроенный браузер

Виджет *WebView*

В разрабатываемых приложениях вы также можете использовать встроенный браузер. Это виджет `WebView` из пакета `android.webkit`. В этом пакете представлено множество классов и интерфейсов, которые можно использовать для создания веб-браузера в приложениях.

Класс `WebView` представляет собой основу для создания встроенного в приложение веб-браузера. `WebView` использует движок на основе браузера WebKit для отображения веб-страниц и включает множество методов для навигации, масштабирования, выполнения текстового поиска, конфигурации пользовательских настроек и безопасности и много другой функциональности, требуемой для создания полноценного веб-браузера.

Использование виджета *WebView*

Для загрузки контента в классе `WebView` предусмотрен метод `loadUrl()`. Есть два варианта метода `loadUrl()`. Первый вариант принимает во входном параметре строку с URL и загружает веб-страницу с этим URL. Второй вариант метода `loadUrl()` загружает веб-страницу с URL, заданным в первом параметре, и дополнительным заголовком, передаваемом во втором параметре.

Сейчас мы рассмотрим использование виджета `WebView` в приложении и загрузку в него веб-страниц. Для этого приложения создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `WebKit`;
- ❑ **Application name** — `WebKit browser`;

- ❑ **Package name** — `com.samples.network.webkit`;
- ❑ **Create Activity** — `WebKitActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch34_WebKit`.

В файле манифеста приложения `AndroidManifest.xml` для доступа к Интернету потребуется разрешение `android.permission.INTERNET`. Код файла манифеста приложения представлен в листинге 34.4.

Листинг 34.4. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.network.webkit"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".WebKitActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-sdk android:minSdkVersion="10" />
</manifest>
```

В файле компоновки главного окна приложения `main.xml` будет содержаться только виджет `WebView`. Код файла `main.xml` приведен в листинге 34.5.

Листинг 34.5. Файл компоновки окна приложения `main.xml`

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

    <WebView
        android:id="@+id/browser"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>
</LinearLayout>
```

В коде класса главного окна приложения просто используем вызов метода `loadUrl()` и передадим в него в качестве параметра строку URL, указывающую, например, на домашнюю страницу Google. Код класса главного окна приложения `WebKitActivity` представлен в листинге 34.6.

Листинг 34.6. Файл класса окна приложения `WebKitActivity.java`

```
package com.samples.network.webkit;

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

public class WebKitActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        WebView myBrowser=(WebView)findViewById(R.id.browser);
        myBrowser.getSettings().setJavaScriptEnabled(true);

        String url = "http://google.com";
        myBrowser.loadUrl(url);
    }
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. При запуске приложения в браузер `WebView` будет загружаться домашняя страница Google. Внешний вид приложения представлен на рис. 34.4.

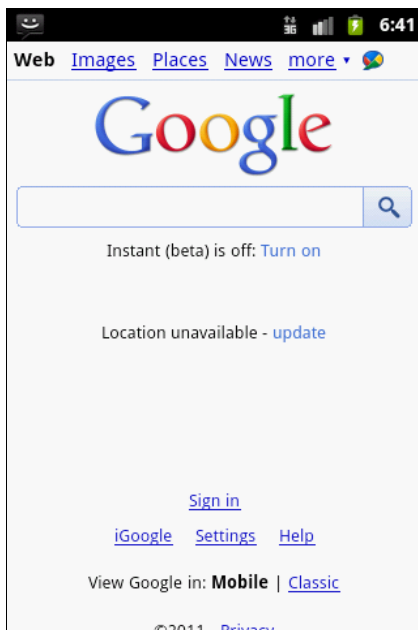


Рис. 34.4. Виджет `WebView` в приложении

Загрузка данных в виджет *WebView*

В браузер *WebView* очень легко загружать мобильный контент. Для этого в классе *WebView* предусмотрены два метода:

- ❑ `loadData()`;
- ❑ `loadDataWithBaseUrl()`.

Для метода `loadData()` входными параметрами являются строка, определяющая содержимое страницы, тип MIME и кодировка. Например, так можно создать веб-страницу и загрузить ее в объект *WebView*:

```
WebView webKit;  
...  
String html "<html><body><h3>This is web page content</h3></body></html>";  
webKit.loadData(html, "text/html", "UTF-8");
```

Метод `loadDataWithBaseUrl()` загружает данные в *WebView*, используя URL как базовый для данного контента.

Можете модифицировать код предыдущего проекта (или создать новый), и чтобы попробовать эти методы на практике. Код класса главного окна приложения представлен в листинге 34.7.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch34_WebKit_v2`.

Листинг 34.7. Файл класса окна приложения *WebKitActivity.java*

```
package com.samples.web.webkit2;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.webkit.WebView;  
  
public class WebKitActivity extends Activity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        WebView webKit = (WebView) findViewById(R.id.browser);  
  
        String html = "<html><body>" +  
            "<a href='\"http://translate.google.com/?hl=en&tab=iT#\"'>" +  
            "Go to Google Translate Service</a></body></html>";  
  
        webKit.loadData(html, "text/html", "UTF-8");  
    }  
}
```

Запустите проект на эмуляторе Android. В окне приложения будет отображаться веб-ссылка на ресурс Google Translate Service, по которой можно перейти на саму веб-страницу, как показано на рис. 34.5.

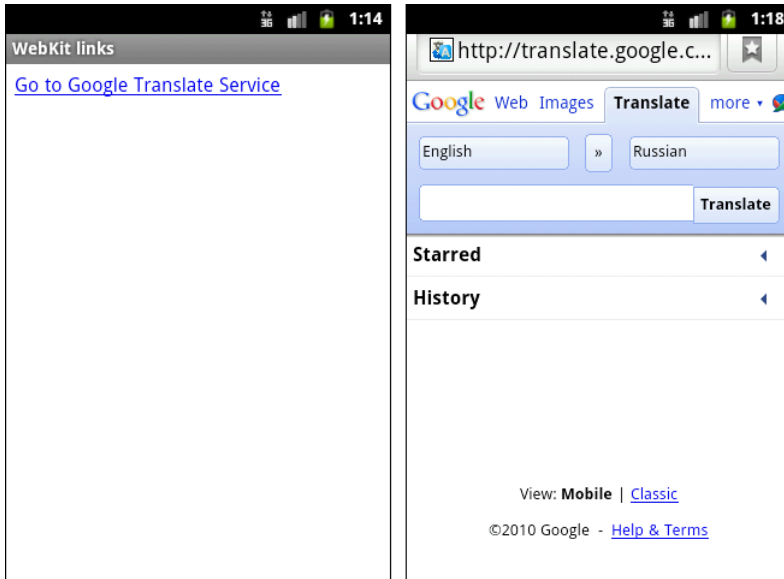


Рис. 34.5. Использование виджета WebView для загрузки данных

Сохранение пользовательских настроек

По умолчанию виджет WebView не разрешает выполнение скриптов JavaScript на странице, предоставляет шрифт маленького размера и другие настройки, которые пользователь может захотеть поменять.

Для доступа к настройкам браузера используется класс `WebSettings`. Этот класс содержит большое количество методов для чтения и установки параметров настроек браузера WebView:

- ❑ для установки параметров отображения текста и шрифтов: `setTextSize()`, `setFixedFontFamily()`, `setCursiveFontFamily()`, `setMinimumLogicalFontSize()`, `setMinimumFontSize()`, `setDefaultFixedFontSize()`, `setDefaultFontSize()`;
- ❑ для установки масштаба: `setDefaultZoom()`, `setDisplayZoomControls()`, `setBuiltInZoomControls()`, `setSupportZoom()`;
- ❑ для установки параметров кэша: `setCacheMode()`, `setAppCacheEnabled()` и т. д.

Пользовательские настройки браузера можно сохранять в системе, используя функциональность класса `Preference`, так же как и для всех стандартных системных настроек мобильного устройства.

Сейчас мы создадим приложение, в котором будут реализованы функциональности, предоставляемые виджетом `WebView`, и возможности для сохранения пользовательских настроек в системе.

Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — WebKitSettings;
- ☐ **Application name** — Browser with menu;
- ☐ **Package name** — com.samples.web.webkitmenu;
- ☐ **Create Activity** — WebKitActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch34_WebKitSettings.

Меню приложения будет состоять из шести опций:

- ☐ **URL bar** — отображение/скрытие поля с URL;
- ☐ **Reload** — обновление страницы;
- ☐ **Back** — навигация назад;
- ☐ **Forward** — навигация вперед;
- ☐ **Settings** — доступ к окну пользовательских настроек;
- ☐ **Exit** — выход из приложения.

Также мы создадим окно для изменения и сохранения настроек. У нас будет несколько пользовательских настроек, определяющих режим работы браузера и загрузки содержимого на страницу:

- ☐ установка URL стартовой страницы;
- ☐ разрешение загрузки графики на страницу;
- ☐ разрешение на использование скриптов JavaScript на странице;
- ☐ блокирование всплывающих окон.

В файле манифеста приложения AndroidManifest.xml потребуется использовать разрешение `android.permission.INTERNET`. Кроме того, у нас будет окно пользовательских настроек, которое назовем `WebKitPreferencesActivity`. Для этого добавим в файл манифеста приложения еще один элемент `<activity>` с атрибутом `android:name=".WebKitPreferencesActivity"`.

Код файла манифеста приложения AndroidManifest.xml представлен в листинге 34.8.

Листинг 34.8. Файл манифеста приложения AndroidManifest.xml

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

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".WebKitActivity"
            android:label="@string/app_name">
```

```

<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=".WebKitPreferencesActivity"
    android:label="@string/app_name">
</activity>
</application>

<uses-permission android:name="android.permission.INTERNET" />
<uses-sdk android:minSdkVersion="10" />
</manifest>

```

В файле компоновки главного окна приложения `main.xml`, кроме виджета `WebView`, будет поле `EditText` для ввода URL и кнопка для загрузки URL в браузер. Файл компоновки окна `main.xml` представлен в листинге 34.9.

Листинг 34.9. Файл компоновки окна приложения `main.xml`

```

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

    <LinearLayout
        android:id="@+id/url_bar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:visibility="visible"
        android:layout_gravity="center">
        <EditText
            android:id="@+id/text_url"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="http://" android:layout_gravity="center"/>
        <Button
            android:id="@+id/button_load"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

            android:text="@string/tx_load"
            android:layout_gravity="center"/>
    </LinearLayout>

    <WebView
        android:id="@+id/browser"

```

```

    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>

```

```

</LinearLayout>

```

В нашем приложении будет значительное количество строк для меню и пользовательских настроек, поэтому их целесообразно вынести в файл строковых ресурсов `strings.xml`, находящийся в каталоге `res/values/`.

Код файла `strings.xml` с добавленными строковыми ресурсами для меню и пользовательских настроек представлен в листинге 34.10.

Листинг 34.10. Файл строковых ресурсов `strings.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Browser with menu</string>

    <string name="tx_url">URL:</string>
    <string name="tx_load">Load</string>

    <string name="mn_urlbar">URL bar</string>
    <string name="mn_back">&lt;&lt;</string>
    <string name="mn_refresh">Reload</string>
    <string name="mn_forward">>></string>
    <string name="mn_pref">Settings</string>
    <string name="mn_exit">Exit</string>

    <string name="hd_options">Options</string>

    <string name="pk_url">url</string>
    <string name="pr_url">Startup URL</string>
    <string name="sm_url">Set startup URL</string>

    <string name="pk_images">images</string>
    <string name="pr_images">Load images</string>
    <string name="sm_images">
        Enable/disable images for faster page loading</string>

    <string name="hd_security">Security</string>

    <string name="pk_jscript">jscript</string>
    <string name="pr_jscript">Enable Java Script</string>
    <string name="sm_jscript">Enable/disable Java Script in browser</string>

    <string name="pk_popup">popup</string>
    <string name="pr_popup">Block pop-up windows</string>
    <string name="sm_popup">
        Enable/disable pop-up windows in browser</string>
</resources>

```

В коде класса `WebKitActivity` главного окна приложения, помимо создания меню, необходимо будет реализовать функциональность для изменения пользовательских настроек браузера в обработчике события `onResume()`.

Код класса `WebKitActivity` приложения представлен в листинге 34.11.

Листинг 34.11. Файл класса окна приложения `WebKitActivity.java`

```
package com.samples.web.webkitmenu;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TableLayout;
import android.widget.Toast;

public class WebKitActivity extends Activity {
    private static final int IDM_URLBAR = 101;
    private static final int IDM_REFRESH = 102;
    private static final int IDM_BACK = 103;
    private static final int IDM_FORWARD = 104;
    private static final int IDM_SETTINGS = 105;
    private static final int IDM_EXIT = 106;

    private LinearLayout layoutBar;
    private WebView browser;
    private EditText textUrl;
    private Button buttonUrl;

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

        layoutBar = (LinearLayout) findViewById(R.id.url_bar);
        browser = (WebView) findViewById(R.id.browser);
        textUrl = (EditText) findViewById(R.id.text_url);
        buttonUrl = (Button) findViewById(R.id.button_load);
```

```
buttonUrl.setOnClickListener(buttonUrlOnClick);
browser.setWebViewClient(new WebViewClient());

}

@Override
public void onResume() {
    super.onResume();

    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(this);

    // Стартовая страница
    String url = prefs.getString(getString(R.string.pk_url), "http://");
    browser.loadUrl(url);

    // Разрешение загрузки графики на страницу
    boolean allowImages =
        prefs.getBoolean(getString(R.string.pk_images), true);

    // Разрешение использования Java-скриптов
    boolean allowJavaScript =
        prefs.getBoolean(getString(R.string.pk_jscript), true);

    // Разрешение отображать всплывающие окна
    boolean allowPopup =
        prefs.getBoolean(getString(R.string.pk_popup), false);

    // Устанавливаем новые настройки браузера
    WebSettings settings = browser.getSettings();
    settings.setBlockNetworkImage(allowImages);
    settings.setJavaScriptEnabled(allowJavaScript);
    settings.setJavaScriptCanOpenWindowsAutomatically(allowPopup);

    textUrl.setText(url);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, IDM_URLBAR, 0, R.string.mn_urlbar);
    menu.add(0, IDM_SETTINGS, 0, R.string.mn_pref);
    menu.add(0, IDM_EXIT, 0, R.string.mn_exit);
    menu.add(0, IDM_BACK, 0, R.string.mn_back);
    menu.add(0, IDM_REFRESH, 0, R.string.mn_refresh);
    menu.add(0, IDM_FORWARD, 0, R.string.mn_forward);

    return super.onCreateOptionsMenu(menu);
}
```

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);

    switch(item.getItemId()) {
    case IDM_URLBAR:
        if (layoutBar.getVisibility() == View.VISIBLE) {
            layoutBar.setVisibility(View.GONE);
        }
        else {
            layoutBar.setVisibility(View.VISIBLE);
        }
        break;
    case IDM_REFRESH:
        browser.reload();
        break;
    case IDM_BACK:
        if(browser.canGoBack())
            browser.goBack();
        break;
    case IDM_FORWARD:
        if(browser.canGoForward())
            browser.goForward();
        break;
    case IDM_SETTINGS:
        Intent i = new Intent();
        i.setClass(this, WebKitPreferencesActivity.class);
        startActivity(i);
        break;
    case IDM_EXIT:
        this.finish();
        break;
    }

    return true;
}

private Button.OnClickListener buttonUrlOnClick =
    new Button.OnClickListener() {

        @Override
        public void onClick(View v) {
            browser.loadUrl(textUrl.getText().toString());
        }
    };
}

```

Для окна пользовательских настроек потребуется создать отдельный XML-файл компоновки. Этот файл должен размещаться в каталоге `res/xml/`. Вложенный каталог `xml`

при создании проекта Android в IDE Eclipse автоматически не создается, его необходимо создать вручную и добавить в него новый XML-файл с именем `preferences.xml`.

Файл `preferences.xml` будет определять внешний вид окна пользовательских настроек. Код этого файла представлен в листинге 34.12.

Листинг 34.12. Файл компоновки окна пользовательских настроек `preferences.xml`

```
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory
        android:title="@string/hd_options">
        <EditTextPreference
            android:key="@string/pk_url"
            android:title="@string/pr_url"
            android:summary="@string/sm_url"
            android:defaultValue="http://"
            android:dialogTitle="@string/pr_url"/>
        <CheckBoxPreference
            android:key="@string/pk_images"
            android:title="@string/pr_images"
            android:summary="@string/sm_images"
            android:defaultValue="true"/>
    </PreferenceCategory>

    <PreferenceCategory
        android:title="@string/hd_security">
        <CheckBoxPreference
            android:key="@string/pk_jscript"
            android:title="@string/pr_jscript"
            android:summary="@string/sm_jscript"
            android:defaultValue="true"/>
        <CheckBoxPreference
            android:key="@string/pk_popup"
            android:title="@string/pr_popup"
            android:summary="@string/sm_popup"
            android:defaultValue="false"/>
    </PreferenceCategory>

</PreferenceScreen>
```

Класс окна пользовательских настроек `WebKitPreferencesActivity` очень простой. Этот класс должен наследоваться от класса `PreferenceActivity`, и в нем необходимо определить единственный метод `onCreate()`, в теле которого сделать вызов метода `addPreferencesFromResource()` для загрузки окна пользовательских настроек приложения.

Код класса главного окна приложения представлен в листинге 34.13.

Листинг 34.13. Файл класса окна пользовательских настроек WebKitPreferencesActivity.java

```
package com.samples.web.webkitmenu;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class WebKitPreferencesActivity extends PreferenceActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.preferences);
    }
}
```

Теперь скомпилируйте проект и запустите его на эмуляторе Android. Внешний вид нашего приложения с открытым меню представлен на рис. 34.6.

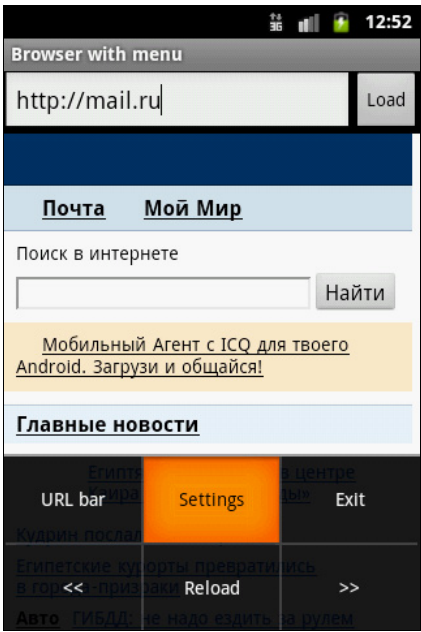


Рис. 34.6. Браузер на основе виджета WebView с меню

При выборе опции **Settings** откроется окно пользовательских настроек, представленное на рис. 34.7.

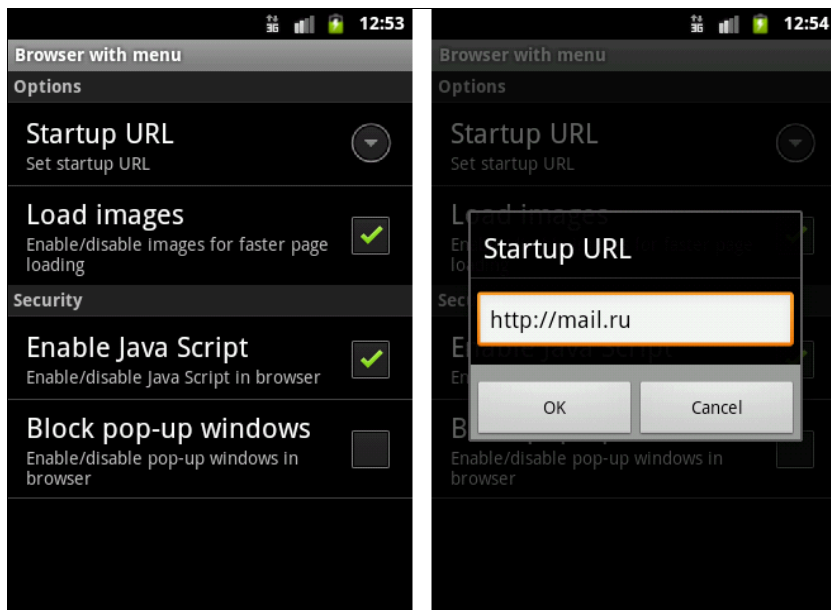


Рис. 34.7. Окно пользовательских настроек браузера

Резюме

В этой главе мы познакомились с функциональностью, предоставляемой платформой Android, для создания и управления соединениями через мобильный Интернет. Также мы рассмотрели использование встроенного браузера, предоставляемого Android, в своих приложениях.

В следующей главе мы будем изучать особенности создания и управления соединениями мобильного устройства через Wi-Fi и создание приложений, использующих коммуникацию через Wi-Fi.



ГЛАВА 35

Управление Wi-Fi-соединениями

В этой главе рассматриваются сети Wi-Fi и взаимодействие с ними мобильного устройства Android.

Приложения, использующие Wi-Fi-соединение, могут взаимодействовать непосредственно с другими компьютерами или с мобильными устройствами беспроводной сети или подключиться к существующей сети через предоставляемую этой сетью точку доступа.

Обычно схема Wi-Fi-сети содержит одну или несколько точек доступа. Также возможно подключение двух клиентов в режиме Ad-hoc, когда точка доступа не используется, а клиенты соединяются напрямую через свои сетевые Wi-Fi-адаптеры.

Управление соединением Wi-Fi

Необходимая функциональность для работы с Wi-Fi в Android SDK обеспечивается пакетом `android.net.wifi`. В этом пакете класс `WifiManager` предоставляет базовую функциональность для управления Wi-Fi-соединениями непосредственно из кода приложения.

Менеджер Wi-Fi-соединений

Объект `WifiManager`, так же как и остальные системные менеджеры, создается путем вызова метода `getSystemService()`:

```
WifiManager manager = getSystemService(Context.WIFI_SERVICE);
```

Созданный объект `WifiManager` можно использовать для подключения и конфигурации Wi-Fi-сети, сканирования точек доступа и других операций по управлению сетевыми соединениями, специфичными для сети Wi-Fi.

В классе `WifiManager` также определен большой набор методов для управления соединением и получения базовых данных о Wi-Fi-соединении, которые мы будем рассматривать на протяжении этой главы.

Разрешения

Помимо разрешений, которые мы использовали в предыдущей главе при управлении и взаимодействии через Интернет, для работы приложения с соединениями по сети Wi-Fi необходимы дополнительные разрешения:

- ❑ `ACCESS_WIFI_STATE` — для разрешения приложению получать информацию о Wi-Fi-соединении;
- ❑ `CHANGE_WIFI_STATE` — для разрешения приложению изменять состояние Wi-Fi-соединения.

Состояние соединения

Для управления соединением в классе `WifiManager` определен метод `setWifiEnabled()`. Используя его, можно включать или выключать соединение, передавая в этот метод в качестве параметра булево значение — `true` для включения соединения:

```
manager.setWifiEnabled(true);
```

или `false` — для выключения соединения:

```
manager.setWifiEnabled(false);
```

Для определения состояния соединения в классе `WifiManager` существует несколько констант. Состояние Wi-Fi-соединения определяется следующими значениями:

- ❑ `WIFI_STATE_ENABLING`;
- ❑ `WIFI_STATE_ENABLED`;
- ❑ `WIFI_STATE_DISABLING`;
- ❑ `WIFI_STATE_DISABLED`;
- ❑ `WIFI_STATE_UNKNOWN`.

Как вы видите по названиям этих констант, существуют константы `WIFI_STATE_ENABLING` и `WIFI_STATE_DISABLING` для отображения промежуточных состояний подключения к Wi-Fi, т. к. процесс установления соединения может занять достаточно длительное время.

Отслеживание состояния соединения

Изменение состояния подключения мобильного устройства к сети Wi-Fi можно отслеживать в программном коде. Дело в том, что при изменении состояния сетевого соединения Wi-Fi менеджер генерирует объекты `Intent`. Тип этого объекта `Intent` определяется одной из строковых констант в классе `WifiManager`:

- ❑ `WIFI_STATE_CHANGED_ACTION` — действие `Broadcast Intent`, указывающее на изменение состояния Wi-Fi-соединения. Это действие имеет два дополнения. Первый Extra-параметр `EXTRA_WIFI_STATE` возвращает текущее состояние. Второй Extra-параметр `EXTRA_PREVIOUS_WIFI_STATE` возвращает значение предыдущего состояния соединения в случае, если оно было доступно;
- ❑ `NETWORK_STATE_CHANGED_ACTION` — действие `Broadcast Intent`, указывающее на изменение состояния сетевого соединения. Первый Extra-параметр `EXTRA_NETWORK_INFO` обес-

печивает новое состояние в форме объекта `NetworkInfo`. Если новое состояние является установленным соединением, то второй Extra-параметр `EXTRA_BSSID` возвращает идентификатор BSSID точки доступа Wi-Fi. BSSID (Basic Service Set Identifier) — идентификатор сети Wi-Fi, о нем будет рассказано далее в этой главе;

- ❑ `SUPPLICANT_CONNECTION_CHANGE_ACTION` — действие Broadcast Intent, указывающее на изменение состояния соединения с клиентом, который запрашивал соединение: соединение было установлено (и теперь можно выполнять операции Wi-Fi) или потеряно. Единственный Extra-параметр `EXTRA_SUPPLICANT_CONNECTED` обеспечивает чтение состояния соединения;
- ❑ `SUPPLICANT_STATE_CHANGED_ACTION` — действие Broadcast Intent, указывающее на изменение состояния соединения с точкой доступа (например, в процессе работы произошел обрыв соединения). Это действие дополнительно предоставляет новое состояние в виде объекта `SupplicantState`, которое определяет текущее состояние для запрашивающего соединения клиента. Действие содержит два Extra-параметра: `EXTRA_NEW_STATE`, возвращающее новое состояние, и `EXTRA_SUPPLICANT_ERROR`, который представляет описание кода ошибки.

Например, создать приемник для отслеживания изменений состояния соединения Wi-Fi можно следующим образом:

```
BroadcastReceiver receiver = new BroadcastReceiver() {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        int wifiState = intent.getIntExtra(  
            WifiManager.EXTRA_WIFI_STATE,  
            WifiManager.EXTRA_PREVIOUS_WIFI_STATE);  
  
        switch(wifiState) {  
            case WifiManager.WIFI_STATE_ENABLING:  
                ...  
                break;  
            case WifiManager.WIFI_STATE_ENABLED:  
                ...  
                break;  
            case WifiManager.WIFI_STATE_DISABLING:  
                ...  
                break;  
            case WifiManager.WIFI_STATE_DISABLED:  
                ...  
                break;  
            case WifiManager.WIFI_STATE_UNKNOWN:  
                ...  
                break;  
        }  
    }  
}
```

Чтобы запустить мониторинг состояния соединения Wi-Fi, надо зарегистрировать приемник с помощью метода `registerReceiver()` следующим образом:

```
registerReceiver(
    receiver, new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));
```

Для остановки мониторинга необходимо вызвать метод `unregisterReceiver()`, передав ему в качестве параметра объект `BroadcastReceiver`:

```
unregisterReceiver(receiver);
```

Управление подключением Wi-Fi и отслеживание состояния соединения из приложения

Сейчас мы создадим небольшое приложение, управляющее Wi-Fi-соединением и отслеживающее изменения состояния соединения.

ПРИМЕЧАНИЕ

К сожалению, эмулятор Android не обеспечивает симулирование Wi-Fi-соединения, и приложение должно тестироваться на реальном устройстве. Кроме того, у вас должна быть работоспособная сеть Wi-Fi хотя бы с одной точкой доступа.

Для нашего приложения создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `ManageWiFiConnection`;
- ☐ **Application name** — `Manage Wi-Fi connection`;
- ☐ **Package name** — `com.samples.network.wifimanagement`;
- ☐ **Create Activity** — `WiFiChangeActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch35_ManageWiFiConnection`.

В файле манифеста приложения `AndroidManifest.xml` нам потребуется объявить разрешение `android.permission.CHANGE_WIFI_STATE`. Код файла манифеста представлен в листинге 35.1.

Листинг 35.1. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.network.wifimanagement"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name="WiFiChangeActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
```



```
<uses-permission
    android:name="android.permission.CHANGE_WIFI_STATE" />

</manifest>
```

В файле компоновки главного окна приложения `main.xml` создадим две кнопки для включения и выключения соединения и текстовое поле для вывода информации о соединении. Файл компоновки `main.xml` представлен в листинге 35.2.

Листинг 35.2. Файл компоновки окна приложения `main.xml`

```
<?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">

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent">

        <Button
            android:id="@+id/bEnable"
            android:layout_height="wrap_content"
            android:text="Enable Wi-Fi"
            android:layout_width="fill_parent"
            android:layout_weight="1"
            android:layout_margin="5px"/>

        <Button
            android:id="@+id/bDisable"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:text="Disable Wi-Fi"
            android:layout_weight="1"
            android:layout_margin="5px"/>

    </LinearLayout>

    <TextView
        android:id="@+id/text"
        android:text=""
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:layout_margin="5px"/>

</LinearLayout>
```

В коде класса `WiFiChangeActivity` главного окна приложения нам потребуется объект `BroadcastReceiver` для отслеживания изменений состояния сети Wi-Fi. В обработке

события `onClick()` кнопок будет производиться включение и выключение сетевого соединения.

Код класса `WiFiChangeActivity` представлен в листинге 35.3.

Листинг 35.3. Файл класса окна приложения `WiFiChangeActivity.java`

```
package com.samples.network.wifimanagement;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class WiFiChangeActivity extends Activity
    implements View.OnClickListener {

    private TextView text;
    private WifiManager manager;

    // Приемник для перехвата изменения состояния соединения сети Wi-Fi
    private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            int wifiState = intent.getIntExtra(
                WifiManager.EXTRA_WIFI_STATE,
                WifiManager.WIFI_STATE_UNKNOWN);
            text.append("\n\t");

            switch(wifiState){
            case WifiManager.WIFI_STATE_ENABLING:
                text.append("Wi-Fi state enabling");
                break;
            case WifiManager.WIFI_STATE_ENABLED:
                text.append("Wi-Fi state enabled");
                break;
            case WifiManager.WIFI_STATE_DISABLING:
                text.append("Wi-Fi state disabling");
                break;
            case WifiManager.WIFI_STATE_DISABLED:
                text.append("Wi-Fi state disabled");
                break;
            case WifiManager.WIFI_STATE_UNKNOWN:
                text.append("Wi-Fi state unknown");
                break;
```

```
        default:
            text.append("Not defined");
            break;
    }
}

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

    text = (TextView)findViewById(R.id.text);

    Button bEnable = (Button)findViewById(R.id.bEnable);
    Button bDisable = (Button)findViewById(R.id.bDisable);

    text.append("Current Wi-Fi state:");
    bEnable.setOnClickListener(this);
    bDisable.setOnClickListener(this);

    manager = (WifiManager) getSystemService(Context.WIFI_SERVICE);

    // Регистрация приемника для прослушивания изменения
    // состояния соединения
    this.registerReceiver(this.receiver,
        new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bEnable:
            // Включение Wi-Fi
            manager.setWifiEnabled(true);
            text.append("\nStart enable Wi-Fi...");
            break;
        case R.id.bDisable:
            // Выключение Wi-Fi
            manager.setWifiEnabled(false);
            text.append("\nStart disable Wi-Fi...");
            break;
    }
}
}
```

Выполнять тестирование приложений, использующих Wi-Fi, необходимо на реальном мобильном устройстве. Приложение будет последовательно записывать все изменения

статуса Wi-Fi-соединения при последовательном нажатии пользователем кнопок **Enable Wi-Fi** и **Disable Wi-Fi**.

Внешний вид и работа приложения на мобильном устройстве при изменении состояния подключения представлены на рис. 35.1.

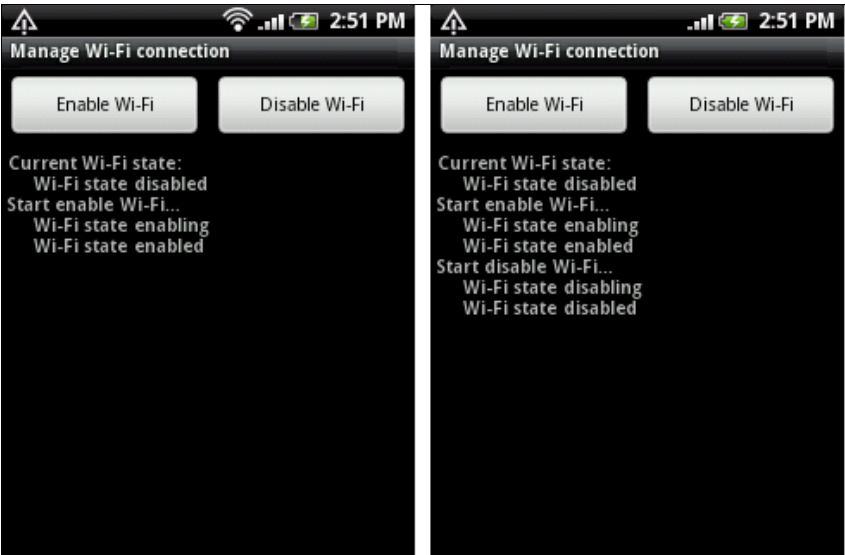


Рис. 35.1. Приложение, управляющее Wi-Fi-соединением на мобильном устройстве

Хотя эмулятор Android не обеспечивает симулирование Wi-Fi-соединения, на нем доступны все функции Wi-Fi. В принципе запустить приложение, работающее с Wi-Fi-соединением, можно и на эмуляторе, однако он не обеспечит вам реальное соединение.

Чтобы убедиться в этом, можете запустить наше приложение на эмуляторе. Никаких ошибок или исключений генерироваться не будет, но и открытия соединения Wi-Fi также происходить не будет (рис. 35.2).

Однако использовать эмулятор Android удобно для предварительной отладки приложений, чтобы устранить часть ошибок, не связанных с управлением соединениями Wi-Fi. Затем можно произвести уже окончательную отладку приложения на реальном мобильном устройстве.

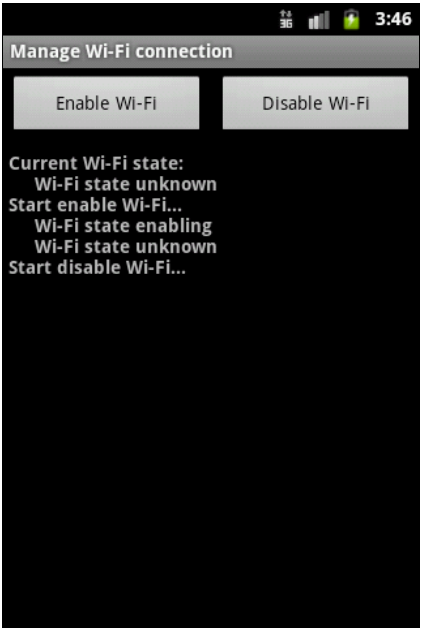


Рис. 35.2. Запуск приложения для управления Wi-Fi-соединением в эмуляторе Android

Управление настройками Wi-Fi-соединения

Система Android предоставляет стандартные Activity для управления настройками конфигурации Wi-Fi-соединения. В классе `Settings` определены три константы для вызова настроек:

- ❑ `ACTION_WIFI_SETTINGS` — для отображения окна настроек для включения/выключения Wi-Fi-соединения, добавления новых соединений;
- ❑ `ACTION_WIFI_IP_SETTINGS` — для отображения окна настройки для IP-адреса, шлюза;
- ❑ `ACTION_WIRELESS_SETTINGS` — для отображения окна для общих настроек беспроводных соединений: Wi-Fi, Bluetooth, мобильного Интернета.

Кроме того, в классе `WifiManger` определена константа для вызова окна настроек `ACTION_PICK_WIFI_NETWORK`, но она аналогична константе `ACTION_WIFI_SETTINGS` из класса `Settings`.

Вызвать стандартный Activity для настроек можно обычным способом с помощью объекта `Intent`:

```
Intent intent = new Intent(Settings.ACTION_WIFI_SETTINGS);
startActivity(intent);
```

Эти окна настроек можно вызывать из приложения, чтобы не создавать излишнюю функциональность в коде приложения.

Сейчас мы рассмотрим эти настройки и их вызов из кода приложения. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `WiFiSettings`;
- ❑ **Application name** — `Standard WiFi Settings`;
- ❑ **Package name** — `com.samples.network.wifisettings`;
- ❑ **Create Activity** — `WiFiSettingsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch35_WiFiSettings`.

В файл манифеста приложения `AndroidManifest.xml` добавьте разрешение `android.permission.CHANGE_WIFI_STATE`, как в листинге 35.1 предыдущего примера. Файл компоновки `main.xml` нам не потребуется, т. к. мы будем использовать для создания окна приложения класс, производный от `ListActivity`.

Код файла класса `WiFiSettingsActivity` представлен в листинге 35.4.

Листинг 35.4. Файл класса окна приложения `WiFiSettingsActivity.java`

```
package com.samples.network.wifisettings;

import android.app.Activity;
import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.provider.Settings;
import android.view.View;
```

```
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;

public class WiFiSettingsActivity extends ListActivity {
    private String[] actions = {
        Settings.ACTION_WIFI_IP_SETTINGS,
        Settings.ACTION_WIFI_SETTINGS,
        Settings.ACTION_WIRELESS_SETTINGS
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        try {
            setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, actions));
        }
        catch (Exception e) {
            Toast.makeText(this, e.toString(), 3000).show();
        }
    }

    public void onItemClick(ListView parent, View v, int pos, long id)
    {
        try {
            Intent intent = new Intent(actions[pos]);
            startActivity(intent);
        }
        catch (Exception e) {
            Toast.makeText(this, e.toString(), 3000).show();
        }
    }
}
```

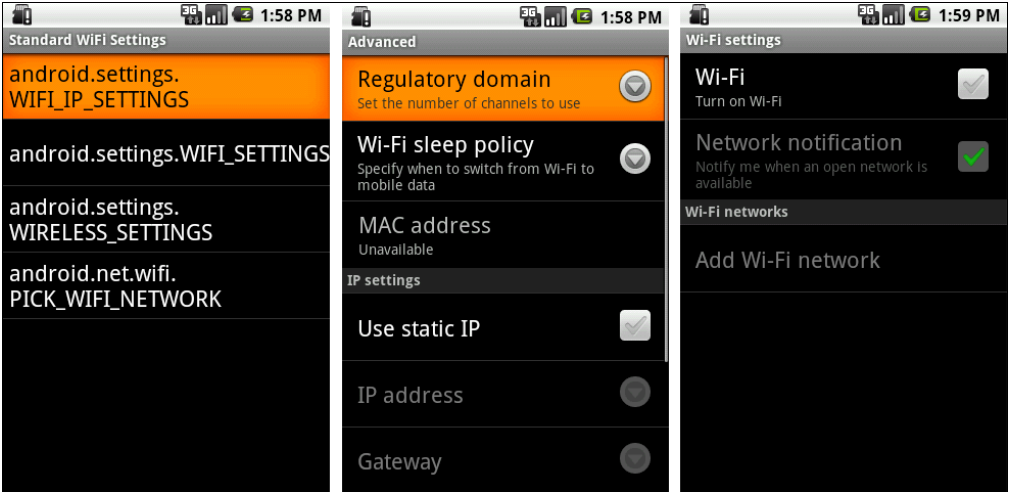


Рис. 35.3. Управление настройками Wi-Fi-соединения

Скомпилируйте проект и запустите его на эмуляторе Android. Приложение будет открывать стандартные окна настроек для управления соединением и конфигурацией сети Wi-Fi. Внешний вид приложения представлен на рис. 35.3.

ПРИМЕЧАНИЕ

Если вы изменяете конфигурацию сети в коде приложения, необходимо вызвать метод `saveConfiguration()` для сохранения настроек сети в памяти телефона.

Характеристики соединения

Кроме управления соединением, в приложении необходимо также иметь возможность получения информации о Wi-Fi-соединении.

Сеть Wi-Fi и ее точки доступа характеризуются специальными идентификаторами. Каждая сеть использует уникальное сетевое имя для идентификации сети. Это имя называется идентификатором обслуживания сети или идентификатором SSID (Service Set Identifier). Когда вы устанавливаете адаптер беспроводной сети, необходимо указать SSID. Если вы хотите подключиться к существующей сети, вы должны использовать имя этой сети. Идентификатор SSID представляет собой строку длиной до 32 символов и может содержать буквы и цифры.

В сети Wi-Fi точка доступа постоянно передает свой идентификатор сети. Если мобильное устройство находит несколько точек доступа с одинаковыми SSID, оно будет выбирать точку доступа с большим уровнем сигнала.

Для получения идентификаторов сети в классе `WifiInfo` определен набор методов:

- ❑ `getSSID()` — возвращает основной идентификатор сети Wi-Fi. Все устройства в беспроводной сети должны использовать один SSID;
- ❑ `getBSSID()` — возвращает BSSID (Basic Service Set Identifier, идентификатор основного пакета услуг) текущей точки доступа. BSSID обычно используется для идентификации компьютеров, объединенных в беспроводную сеть и обменивающихся информацией непосредственно друг с другом, без использования точек доступа.

В классе `WifiInfo` также есть методы `getRssi()` для определения уровня сигнала RSSI (см. главу 31, разд. "Изменение уровня сигнала") и `getLinkSpeed()`, возвращающий текущую скорость передачи данных по сети. Единицу измерения скорости передачи данных определяет строковая константа `LINK_SPEED_UNITS` в классе `WifiInfo`, которая имеет значение "Mbps".

IP-адресация

IP-адрес, присвоенный службой DHCP мобильному устройству при соединении с сетью Wi-Fi, а также остальные характеристики сетевой конфигурации можно получить, используя метод `getDhcpInfo()` класса `WifiManager`.

Метод `getDhcpInfo()` возвращает объект класса `DhcpInfo` из пакета `android.net`. Класс `DhcpInfo` содержит набор полей, характеризующих данное подключение:

- ❑ `ipAddress;`
- ❑ `gateway;`
- ❑ `dns1;`

- ☐ dns2;
- ☐ netmask;
- ☐ serverAddress.

Однако нужно учесть, что эти поля имеют тип `integer`, и для получения IP-адреса в виде октетов эти значения необходимо самостоятельно конвертировать в коде приложения в более привычную для пользователей форму.

Получение информации о сети Wi-Fi в приложении

Теперь попробуем использовать эти классы в практическом приложении. Это приложение будет подключаться к сети Wi-Fi и выводить информацию о конфигурации. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `WiFiInfo`;
- ☐ **Application name** — `Wi-Fi info`;
- ☐ **Package name** — `com.samples.network.wifiinfo`;
- ☐ **Create Activity** — `WiFiInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch35_WiFiInfo`.

В файле манифеста приложения `AndroidManifest.xml` объявите разрешения `ACCESS_NETWORK_STATE`, `CHANGE_WIFI_STATE` и `ACCESS_WIFI_STATE`, как показано в листинге 35.5.

Листинг 35.5. Файл манифеста приложения `AndroidManifest.xml`

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

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".WiFiInfoActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission
        android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission
        android:name="android.permission.CHANGE_WIFI_STATE" />
```



```
<uses-permission
    android:name="android.permission.ACCESS_WIFI_STATE"/>
</manifest>
```

В файле компоновки главного окна приложения `main.xml` будут расположены виджеты `CheckBox` с именем `cbEnable` для включения и выключения соединения и `TextView` с именем `text` для отображения информации о Wi-Fi-соединении. Файл компоновки `main.xml` представлен в листинге 35.6.

Листинг 35.6. Файл компоновки окна приложения `main.xml`

```
<?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">

    <CheckBox
        android:id="@+id/cbEnable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enable Wi-Fi"
        android:layout_margin="5px"
        android:textSize="18sp"/>

    <TextView
        android:id="@+id/text"
        android:text=""
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:layout_margin="5px"/>

</LinearLayout>
```

В коде класса `WiFiInfoActivity` главного окна приложения получение информации о сети мы реализуем в методе `printWifiInfo()`, который будет вызываться при установлении соединения (при статусе сети `WIFI_STATE_ENABLED`).

Код класса `WiFiInfoActivity` главного окна приложения представлен в листинге 35.7.

Листинг 35.7. Файл класса окна приложения `WiFiInfoActivity.java`

```
package com.samples.network.wifiinfo;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
import android.content.IntentFilter;
import android.net.DhcpInfo;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;
import android.widget.CompoundButton.OnCheckedChangeListener;

public class WifiInfoActivity extends Activity
    implements OnCheckedChangeListener {

    private TextView text;
    private CheckBox cbEnable;
    private WifiManager manager;

    // Приемник для отслеживания состояния соединения
    private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            int wifiState = intent.getIntExtra(
                WifiManager.EXTRA_WIFI_STATE,
                WifiManager.WIFI_STATE_UNKNOWN);
            // Изменение статуса сети Wi-Fi
            switch(wifiState){
                case WifiManager.WIFI_STATE_ENABLING:
                    text.setText("Wi-Fi state enabling");
                    break;
                case WifiManager.WIFI_STATE_ENABLED:
                    text.setText("Wi-Fi state enabled");
                    text.append(printWifiInfo());
                    break;
                case WifiManager.WIFI_STATE_DISABLING:
                    text.setText("Wi-Fi state disabling");
                    break;
                case WifiManager.WIFI_STATE_DISABLED:
                    text.setText("Wi-Fi state disabled");
                    break;
                case WifiManager.WIFI_STATE_UNKNOWN:
                    text.setText("Wi-Fi state unknown");
                    break;
            }
        }
    };

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

```

text = (TextView)findViewById(R.id.text);
cbEnable = (CheckBox)findViewById(R.id.cbEnable);

manager = (WifiManager)getSystemService(Context.WIFI_SERVICE);

this.registerReceiver(this.receiver,
    new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));

cbEnable.setChecked(manager.isWifiEnabled());
cbEnable.setOnCheckedChangeListener(this);
}

@Override
public void onCheckedChanged(CompoundButton view, boolean isChecked) {
    manager.setWifiEnabled(isChecked);
}

// Вывод информации на экран
private String printWifiInfo() {
    StringBuilder sb = new StringBuilder();
    WifiInfo wifiInfo = manager.getConnectionInfo();

    DhcpInfo dhcpInfo = manager.getDhcpInfo();

    sb.append("\nWi-Fi Information:");
    sb.append("\n\tMAC Address:\t" + wifiInfo.getMacAddress());
    sb.append("\n\tSS ID:\t" + wifiInfo.getSSID());
    sb.append("\n\tBSS ID:\t" + wifiInfo.getBSSID());

    sb.append("\n\tLink speed:\t" + wifiInfo.getLinkSpeed());
    sb.append("\n\tLink speed units:\t" + WifiInfo.LINK_SPEED_UNITS);
    sb.append("\n\tRSSI:\t" + wifiInfo.getRssi());
    sb.append("\n\tHidden SSID:\t" + wifiInfo.getHiddenSSID());
    sb.append("\n\tNetwork ID:\t" + wifiInfo.getNetworkId());

    sb.append("\n\nDHCP Information");
    sb.append("\n\tIP address:\t" +
        convertIpAddress(dhcpInfo.ipAddress));
    sb.append("\n\tDNS 1:\t" + convertIpAddress(dhcpInfo.dns1));
    sb.append("\n\tDNS 2:\t" + convertIpAddress(dhcpInfo.dns2));
    sb.append("\n\tGateway:\t" + convertIpAddress(dhcpInfo.gateway));
    sb.append("\n\tLease duration:\t" + dhcpInfo.leaseDuration);
    sb.append("\n\tDescribe contents:\t" + dhcpInfo.describeContents());

    return sb.toString();
}

// Метод для конвертирования IP-адреса в октетную форму
private String convertIpAddress(int ipAddress) {
    int ip0, ip1, ip2, ip3, tmp;

```

```

        ip3 = ipAddress / 0x1000000;
        tmp = ipAddress % 0x1000000;
        ip2 = tmp / 0x10000;
        tmp %= 0x10000;
        ip1 = tmp / 0x100;
        ip0 = tmp % 0x100;

        return String.format("%d.%d.%d.%d", ip0, ip1, ip2, ip3);
    }
}

```

Скомпилируйте проект и запустите его на мобильном устройстве Android. Внешний вид приложения представлен на рис. 35.4.



Рис. 35.4. Получение информации о Wi-Fi-соединении

Конфигурация Wi-Fi-соединения

Конфигурацию сети определяет объект класса `WifiConfiguration`. В классе `WifiManager` существует метод `getConfiguredNetworks()`. Он возвращает список всех сетевых конфигураций в виде объекта `List<WifiConfiguration>`:

```

manager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
List<WifiConfiguration> configs = manager.getConfiguredNetworks();

```

В классе `WifiConfiguration` содержится набор полей, определяющих конфигурацию сети Wi-Fi:

- ☐ `SSID` — идентификатор SSID;
- ☐ `BSSID` — идентификатор BSSID;
- ☐ `hiddenSSID` — скрытый идентификатор SSID. Точка доступа не передает его в сеть, однако если он известен, то может использоваться приложением при запросе на поиск конкретной сети;
- ☐ `networkId` — идентификатор сети, который использует клиент, запрашивающий соединение, чтобы идентифицировать эту точку доступа;

- ❑ `status` — текущий статус этого сетевого соединения;
- ❑ `priority` — приоритет, который определяет предпочтение при выборе точки доступа для клиента, запрашивающего защищенный доступ Wi-Fi.

У класса `WifiConfiguration` имеется также набор методов, возвращающих информацию о поддержке различных типов алгоритмов и протоколов безопасности для данной сетевой конфигурации:

- ❑ `allowedProtocols()` — возвращает множество поддерживаемых протоколов безопасности;
- ❑ `allowedAuthAlgorithms()` — возвращает множество поддерживаемых протоколов аутентификации;
- ❑ `allowedGroupCiphers()` — возвращает множество поддерживаемых протоколов шифрования;
- ❑ `allowedKeyManagement()` — возвращает множество протоколов управления ключами шифрования;
- ❑ `wepKeys()` — возвращает ключи WEP (Wired Equivalent Privacy, алгоритм для обеспечения безопасности сетей Wi-Fi). Их может быть до 4 штук;
- ❑ `wepTxKeyIndex()` — индекс ключа WEP по умолчанию. Его значение может быть от 0 до 3, в зависимости от количества ключей;
- ❑ `allowedPairwiseCiphers()` — возвращает множество поддерживаемых протоколов попарного шифрования для WPA. Алгоритм WPA (Wi-Fi Protected Access, алгоритм защищенного доступа для сетей Wi-Fi) представляет собой усовершенствование алгоритма защиты беспроводных сетей WEP. В WPA поддерживается шифрование по стандарту AES (Advanced Encryption Standard, усовершенствованный стандарт шифрования);
- ❑ `preSharedKey()` — ключ для использования аутентификации WPA с помощью предустановленного ключа Pre-Shared Key (WPA-PSK).

Теперь посмотрим, как можно получить конфигурацию сети Wi-Fi в приложении. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `WifiConfiguration`;
- ❑ **Application name** — `Wi-Fi configuration`;
- ❑ **Package name** — `com.samples.network.wificonfig`;
- ❑ **Create Activity** — `WifiConfigActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch35_WifiConfiguration`.

В файле манифеста приложения `AndroidManifest.xml` поставьте необходимые разрешения, как в листинге 35.5. Файл компоновки главного окна приложения `main.xml` будет аналогичен файлу компоновки предыдущего примера, приведенного в листинге 35.6.

В коде класса `WifiConfigActivity` при состоянии сети `WIFI_STATE_ENABLED` мы будем читать ее конфигурацию. В классе `WifiConfigActivity` добавлен закрытый метод

`printWifiConfig()`, в теле которого приложение получает конфигурацию сети Wi-Fi и выводит ее на экран мобильного устройства (листинг 35.8).

Листинг 35.8. Файл класса окна приложения `WifiConfigActivity.java`

```
package com.samples.network.wificonfig;

import java.util.List;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.WifiConfiguration;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;
import android.widget.CompoundButton.OnCheckedChangeListener;

public class WifiConfigActivity extends Activity
    implements OnCheckedChangeListener {

    private TextView text;
    private CheckBox cbEnable;
    private WifiManager manager;

    // Приемник для отслеживания состояния соединения
    private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            int wifiState = intent.getIntExtra(WifiManager.EXTRA_WIFI_STATE,
                WifiManager.WIFI_STATE_UNKNOWN);

            switch(wifiState) {
            case WifiManager.WIFI_STATE_ENABLING:
                text.setText("Wi-Fi state enabling");
                break;
            case WifiManager.WIFI_STATE_ENABLED:
                text.setText("Wi-Fi state enabled");
                // Получаем конфигурацию сети
                printWifiConfig();
                break;
            case WifiManager.WIFI_STATE_DISABLING:
                text.setText("Wi-Fi state disabling");
                break;
```

```

        case WifiManager.WIFI_STATE_DISABLED:
            text.setText("Wi-Fi state disabled");
            break;
        case WifiManager.WIFI_STATE_UNKNOWN:
            text.setText("Wi-Fi state unknown");
            break;
    }
}

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

    text = (TextView)findViewById(R.id.text);
    cbEnable = (CheckBox)findViewById(R.id.cbEnable);

    manager = (WifiManager)getSystemService(Context.WIFI_SERVICE);

    this.registerReceiver(this.receiver,
        new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));

    cbEnable.setChecked(manager.isWifiEnabled());
    cbEnable.setOnCheckedChangeListener(this);
}

@Override
public void onCheckedChanged(CompoundButton view, boolean isChecked) {
    // Управление подключением
    manager.setWifiEnabled(isChecked);
}

// Получение конфигурации и вывод данных на экран
private void printWifiConfig() {
    List<WifiConfiguration> configs = manager.getConfiguredNetworks();

    for (WifiConfiguration config : configs) {
        text.append("\nSS ID:\t" + config.SSID);
        text.append("\nPre-shared key:\t" +
            config.preSharedKey);
        text.append("\nAuthentication protocols:\t" +
            config.allowedAuthAlgorithms);
        text.append("\nSecurity Protocols:\t" +
            config.allowedProtocols);
        text.append("\nGroup ciphers:\t" +
            config.allowedGroupCiphers);
        text.append("\nKey Management:\t" +
            config.allowedKeyManagement);
    }
}

```

```

text.append("\nPairwise Ciphers:\t" +
            config.allowedPairwiseCiphers);
text.append("\nDefault WEP key index:\t" +
            config.wepTxKeyIndex);
text.append("\nWEP keys: ");

String[] weps = config.wepKeys;
for (int i = 0; i < weps.length; i++) {
    text.append(weps[i] + " ");
}
}
}
}

```

Скомпилируйте проект и запустите его на мобильном устройстве. Внешний вид приложения представлен на рис. 35.5.

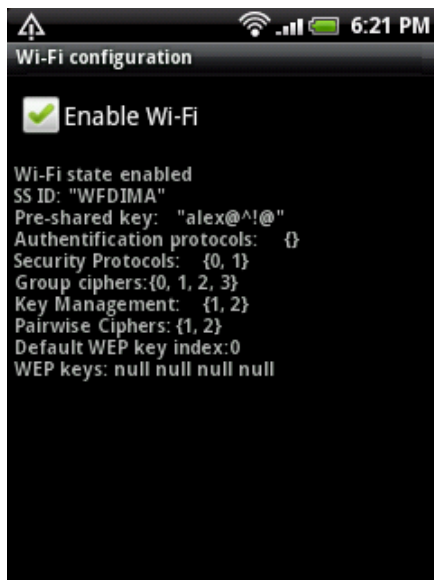


Рис. 35.5. Вывод конфигурации сети Wi-Fi

Сканирование точек доступа

Сканирование точек доступа необходимо для определения наиболее подходящей по уровню сигнала точки для подсоединения мобильного устройства к сети Wi-Fi. Каждая точка доступа передает свой идентификатор сети SSID, используя сигнальные пакеты, которые генерируются каждые 100 мс. Если при сканировании сети обнаруживается несколько точек доступа с одинаковыми идентификаторами сети, выбирается точка доступа с большим уровнем сигнала.

Для сканирования точек доступа используется метод `startScan()`. Этот метод возвращает список объектов типа `ScanResult`. Этот тип определяет набор параметров, характеризующих каждую отсканированную точку доступа.

В классе для этих целей используется ряд полей:

- ❑ `SSID`, `BSSID` — уже описанные ранее идентификаторы сети Wi-Fi;
- ❑ `capabilities` — строка с описанием протокола аутентификации, ключа, схемы шифрования. Это описание выводится в сокращенном виде, например, следующим образом: `"[WPA-PSK-CCMP]"`;
- ❑ `frequency` — рабочая частота канала в МГц;
- ❑ `level` — уровень сигнала в dBm.

Поскольку процесс сканирования занимает достаточно длительное время, при реализации сканирования в коде программы лучше применять асинхронное сканирование с использованием объекта `Intent`. Кроме того, для отслеживания изменений при сканировании нам также понадобится объект `BroadcastReceiver`.

Затем нам потребуется создать новый объект класса `IntentFilter` с входным параметром `WifiManager.SCAN_RESULTS_AVAILABLE_ACTION` и зарегистрировать объект `BroadcastReceiver` с использованием метода `registerReceiver()`, например, таким образом:

```
WifiManager manager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
IntentFilter filter = new IntentFilter(
    WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
registerReceiver(this.scanReceiver,
    new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));
```

Кроме регистрации приемника, необходимо также вызвать метод `startScan()` класса `WifiManager` для запуска сканирования сети:

```
manager.startScan();
```

Для объекта `BroadcastReceiver`, предназначенного для сканирования сети Wi-Fi, надо определить метод `onReceive()`, в котором можно получать обновляемую конфигурацию, используя вызов метода `getScanResults()` класса `WifiManager`:

```
BroadcastReceiver scanReceiver = new BroadcastReceiver()
{
    @Override
    public void onReceive(Context context, Intent intent) {

        List<ScanResult> results = manager.getScanResults();
        for (ScanResult result : results) {
            // Читаем результаты сканирования
            ...
        }
    }
}
```

Сейчас мы реализуем приложение для сканирования сетевых точек доступа. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `ScanWifiAccessPoint`;
- ❑ **Application name** — `Scan Wi-Fi network`;

- ❑ **Package name** — com.samples.network.scanwifi;
- ❑ **Create Activity** — WiFiScanActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch35_ScanWifiAccessPoint.

В файле манифеста приложения AndroidManifest.xml поставьте необходимые разрешения, описанные ранее в этой главе.

В файл компоновки окна приложения main.xml, помимо виджетов CheckBox и TextView, добавьте дополнительную кнопку с атрибутом android:id="@+id/bStart и надписью **Scan Wi-Fi** для запуска процесса сканирования. Код файла main.xml представлен в листинге 35.9.

Листинг 35.9. Файл компоновки окна приложения main.xml

```
<?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">

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/linearLayout1"
        android:orientation="horizontal"
        android:gravity="left|center">

        <CheckBox
            android:textSize="18sp"
            android:layout_height="wrap_content"
            android:id="@+id/cbEnable"
            android:layout_margin="5px"
            android:layout_width="wrap_content"
            android:text="Enable Wi-Fi"/>

        <Button
            android:id="@+id/bStart"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="5px"
            android:layout_weight="1"
            android:text="Scan Wi-Fi"/>

    </LinearLayout>

    <TextView
        android:id="@+id/text"
        android:text=""
```

```
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:textStyle="bold"  
android:layout_margin="5px"/>
```

```
</LinearLayout>
```

В коде класса главного окна приложения разместим дополнительный объект типа `BroadcastReceiver`, который назовем `scanReceiver`. С его помощью мы будем получать список доступных точек сканирования. Также добавим обработчик события нажатия кнопки **Scan Wi-Fi** для запуска сканирования сети.

Код класса главного окна приложения `WiFiScanActivity` представлен в листинге 35.10.

Листинг 35.10. Файл класса окна приложения `WiFiScanActivity.java`

```
package com.samples.network.scanwifi;  
  
import java.util.List;  
  
import android.app.Activity;  
import android.content.BroadcastReceiver;  
import android.content.Context;  
import android.content.Intent;  
import android.content.IntentFilter;  
import android.net.wifi.ScanResult;  
import android.net.wifi.WifiManager;  
import android.os.Bundle;  
import android.view.View;  
import android.view.View.OnClickListener;  
import android.widget.Button;  
import android.widget.CheckBox;  
import android.widget.CompoundButton;  
import android.widget.TextView;  
import android.widget.CompoundButton.OnCheckedChangeListener;  
  
public class WiFiScanActivity extends Activity  
    implements OnCheckedChangeListener, OnClickListener {  
  
    private TextView text;  
    private CheckBox cbEnable;  
    private Button bStart;  
  
    private WifiManager manager;  
  
    // Приемник для отслеживания состояния соединения  
    private BroadcastReceiver receiver = new BroadcastReceiver() {  
        @Override  
        public void onReceive(Context context, Intent intent) {  
            int wifiState = intent.getIntExtra(WifiManager.EXTRA_WIFI_STATE,  
                WifiManager.WIFI_STATE_UNKNOWN);
```

```

switch(wifiState){
case WifiManager.WIFI_STATE_ENABLING:
    text.setText("Wi-Fi state enabling");
    break;
case WifiManager.WIFI_STATE_ENABLED:
    text.setText("Wi-Fi state enabled.");
    break;
case WifiManager.WIFI_STATE_DISABLING:
    text.setText("Wi-Fi state disabling");
    break;
case WifiManager.WIFI_STATE_DISABLED:
    text.setText("Wi-Fi state disabled");
    break;
case WifiManager.WIFI_STATE_UNKNOWN:
    text.setText("Wi-Fi state unknown");
    break;
}
}
};

// Приемник для сканирования точек доступа
private BroadcastReceiver scanReceiver = new BroadcastReceiver(){
    @Override
    public void onReceive(Context context, Intent intent) {
        List<ScanResult> results = manager.getScanResults();
        text.setText("Scan result: " + results.size() + " points");

        // Выводим результаты сканирования на экран
        for (ScanResult result : results) {
            text.append("\nSS ID:\t" + result.SSID);
            text.append("\n\tLevel:\t" + result.level + " dBm");
            text.append("\n\tFrequency:\t" + result.frequency + " MHz");
            text.append("\n\tCapabilities:\t" + result.capabilities);
        }
    }
};

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

    text = (TextView)findViewById(R.id.text);
    cbEnable = (CheckBox)findViewById(R.id.cbEnable);
    bStart = (Button)findViewById(R.id.bStart);

    manager = (WifiManager) getSystemService(Context.WIFI_SERVICE);

    this.registerReceiver(this.receiver,
        new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));

```

```
cbEnable.setChecked(manager.isWifiEnabled());
cbEnable.setOnCheckedChangeListener(this);
bStart.setOnClickListener(this);
}

@Override
public void onCheckedChanged(CompoundButton view, boolean isChecked) {
    manager.setWifiEnabled(isChecked);
}

@Override
public void onClick(View arg0) {
    // Запускаем сканирование сети
    this.registerReceiver(this.scanReceiver,
        new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
    manager.startScan();
}
}
```

Скомпилируйте проект и запустите его на мобильном устройстве. При нажатии кнопки **Scan Wi-Fi** запустится процесс сканирования сети и в текстовом поле будет отображен список доступных точек подключения к сети Wi-Fi с параметрами этих точек, как представлено на рис. 35.6.

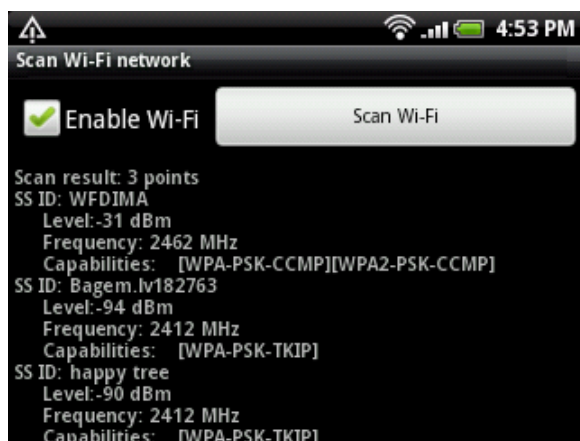


Рис. 35.6. Сканирование точек доступа

Мониторинг уровня сигнала и скорости передачи данных в приложении

Уровень сигнала при беспроводном соединении изменяется в зависимости от взаимного расположения сетевой точки доступа и мобильного устройства. Так как мобильное устройство, подключенное к сети Wi-Fi, постоянно перемещается вместе с пользовате-

лем и может попасть в зону неустойчивого приема или совсем выйти из зоны доступа сети Wi-Fi, актуально в приложении постоянно отслеживать изменение уровня сигнала.

Приложение может контролировать текущую силу сигнала связанной сети Wi-Fi. Для этого надо создать объект `BroadcastReceiver` и определить в нем метод `onReceive()`:

```
BroadcastReceiver rssiReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        WifiInfo info = manager.getConnectionInfo();
        ...
    }
};
```

Затем нам надо зарегистрировать этот объект `BroadcastReceiver` с `Intent`-фильтром, передав фильтру в качестве параметра значение `RSSI_CHANGED_ACTION`, например, таким образом:

```
registerReceiver(rssiReceiver,
    new IntentFilter(WifiManager.RSSI_CHANGED_ACTION));
```

Уровень сигнала можно получить, вызвав метод `getRssi()`, а скорость передачи данных — вызвав `getLinkSpeed()`. Эти методы мы уже использовали в листинге 35.7, когда измеряли характеристики сети в момент создания подключения. Метод `getRssi()` возвращает уровень сигнала RSSI в виде целочисленного значения, которое можно использовать для оценки качества приема и передачи данных. Для этого в классе `WifiManager` определен набор методов:

- ❑ `calculateSignalLevel()` — возвращает относительный уровень сигнала;
- ❑ `compareSignalLevel()` — сравнивает уровни двух сигналов.

В метод `calculateSignalLevel()` передаются два параметра: уровень сигнала в виде целого числа, полученный вызовом метода `WifiInfo.getRssi()`, и параметр, определяющий масштабирование уровня измерения сигнала (количество уровней сигнала), например:

```
WifiManager.calculateSignalLevel(info.getRssi(), 5);
```

Число 5 в этом примере означает наличие пяти уровней для измерения сигнала: если метод `calculateSignalLevel()` вернет 5, это означает максимальный уровень сигнала. От уровня сигнала зависит и скорость передачи данных. Ее изменение также можно отслеживать в приложении с помощью метода `getLinkSpeed()`.

Теперь реализуем отслеживание изменения уровня сигнала в практическом приложении. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `MonitoringRSSI`;
- ❑ **Application name** — `Monitoring RSSI`;
- ❑ **Package name** — `com.samples.network.monitoringrssi`;
- ❑ **Create Activity** — `MonitoringRssiActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch35_MonitoringRSSI`.

В файл манифеста приложения `AndroidManifest.xml` не забудьте добавить соответствующие разрешения.

Код класса окна приложения `MonitoringRssiActivity` в целом похож на класс `WifiConfigActivity` из листинга 35.8 проекта `WiFiConfiguration`, за исключением того, что при подключении соединения создается и регистрируется дополнительный объект `rssiReceiver` типа `BroadcastReceiver`. Он будет отслеживать изменение уровня сигнала и отображать каждое изменение в окне приложения (теперь у нас в приложении будут два приемника: один для отслеживания состояния соединения, другой — для отслеживания уровня сигнала и скорости передачи данных). В методе `onReceive()` объекта `rssiReceiver` при открытии и закрытии соединения мы также добавим функциональность для запуска и остановки мониторинга изменения уровня сигнала.

Код класса окна приложения `MonitoringRssiActivity` приведен в листинге 35.11.

Листинг 35.11. Файл класса окна приложения `MonitoringRssiActivity.java`

```
package com.samples.network.monitoringrssi;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.WifiInfo;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;
import android.widget.CompoundButton.OnCheckedChangeListener;

public class MonitoringRssiActivity extends Activity
    implements OnCheckedChangeListener {

    private TextView text;
    private CheckBox cbEnable;
    private WifiManager manager;

    // Приемник для отслеживания состояния соединения
    private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            int wifiState = intent.getIntExtra(WifiManager.EXTRA_WIFI_STATE,
                WifiManager.WIFI_STATE_UNKNOWN);

            switch(wifiState) {
            case WifiManager.WIFI_STATE_ENABLING:
                text.setText("Wi-Fi state enabling");
                break;
            
```

```

        case WifiManager.WIFI_STATE_ENABLED:
            text.setText("Wi-Fi state enabled");
            // Запускаем мониторинг уровня сигнала
            startMonitoringRssi();
            break;
        case WifiManager.WIFI_STATE_DISABLING:
            text.setText("Wi-Fi state disabling");
            break;
        case WifiManager.WIFI_STATE_DISABLED:
            text.setText("Wi-Fi state disabled");
            // Останавливаем мониторинг уровня сигнала
            stopMonitoringRssi();
            break;
        case WifiManager.WIFI_STATE_UNKNOWN:
            text.setText("Wi-Fi state unknown");
            break;
    }
}

// Приемник для отслеживания уровня сигнала
private BroadcastReceiver rssiReceiver = new BroadcastReceiver(){
    @Override
    public void onReceive(Context context, Intent intent) {
        WifiInfo info = manager.getConnectionInfo();

        // Выводим идентификатор сети,
        // уровень сигнала и скорость передачи данных
        text.append("\nChange signal in " + info.getSSID());
        text.append("\n\tSignal level:\t" +
            WifiManager.calculateSignalLevel(info.getRssi(), 5));
        text.append("\n\tLink speed:\t" + info.getLinkSpeed() +
            " " + WifiInfo.LINK_SPEED_UNITS);
    }
};

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

    text = (TextView)findViewById(R.id.text);
    cbEnable = (CheckBox)findViewById(R.id.cbEnable);

    manager = (WifiManager)getSystemService(Context.WIFI_SERVICE);

    this.registerReceiver(this.receiver,
        new IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION));

    cbEnable.setChecked(manager.isWifiEnabled());
    cbEnable.setOnCheckedChangeListener(this);
}

```



```
@Override
public void onCheckedChanged(CompoundButton view, boolean isChecked) {
    manager.setWifiEnabled(isChecked);
}

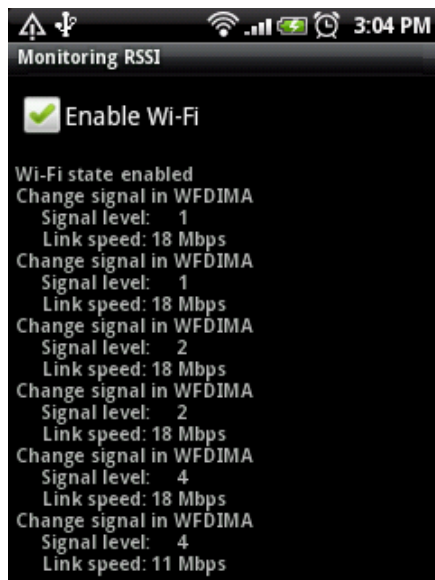
// Запуск мониторинга уровня сигнала
private void startMonitoringRssi() {
    this.registerReceiver(rssiReceiver,
        new IntentFilter(WifiManager.RSSI_CHANGED_ACTION));
}

// Остановка мониторинга уровня сигнала
private void stopMonitoringRssi() {
    if (this.rssiReceiver.isInitialStickyBroadcast())
        this.unregisterReceiver(rssiReceiver);
}
}
```

Скомпилируйте проект и запустите его на мобильном устройстве. Изменение уровня сигнала достаточно легко тестировать, т. к. при нахождении точки подключения в здании на уровень сигнала влияют, кроме удаленности от точки подключения, различные строительные конструкции.

Внешний вид приложения для отслеживания уровня сигнала Wi-Fi-соединения представлен на рис. 35.7.

Рис. 35.7. Отслеживание изменения мощности сигнала Wi-Fi-соединения



Резюме

В этой главе мы рассмотрели подключение и конфигурирование соединения Wi-Fi в коде приложения. Мы также научились создавать приложения, которые могут осуществлять сканирование точек доступа сетевого соединения Wi-Fi и подключение к ним мобильного устройства, читать характеристики сети, отслеживать изменение уровня сигнала и управлять конфигурацией сети Wi-Fi.

Далее мы переходим к новой большой теме — определение местоположения и использование сетевых сервисов Google Maps и Geocoding.



ГЛАВА 36

Определение местоположения

Одно из главных преимуществ сотовых телефонов — мобильность, что позволяет использовать их для навигации, определения своего положения на местности.

В этой главе вы узнаете, как использовать сетевые сервисы определения местоположения и библиотеки Android SDK в приложениях. Android обеспечивает библиотеки API, которые приложение может использовать, чтобы определять местоположение устройства и отслеживать его перемещение.

Использование Google API в эмуляторе

Для работы с сервисами определения местоположения и картами необходимо использовать эмулятор Android с поддержкой Google API. Если у вас нет экземпляра Android Virtual Device с Google API, создайте новый или обновите существующий, как показано на рис. 36.1.

Экземпляр Android Virtual Device с поддержкой библиотек Google API нам понадобится для работы с примерами в этой главе, а также в следующих главах этой части.

Сервисы и провайдеры местоположения

Для создания приложений, предназначенных для работы с сервисами определения местоположения, в Android SDK предусмотрен пакет `android.location`. Этот пакет содержит ряд классов, предоставляющих требуемую функциональность для работы с местоположением.

Класс `LocationManager` является основным классом для доступа к сервисам местоположения. Экземпляр этого класса, так же как и другие менеджеры, рассмотренные ранее, создается через вызов метода `getSystemService()`:

```
LocationManager manager = (LocationManager) getSystemService(  
    Context.LOCATION_SERVICE);
```

Получив экземпляр `LocationManager` в приложении, мы можем воспользоваться богатой функциональностью, которую предоставляют методы этого класса.

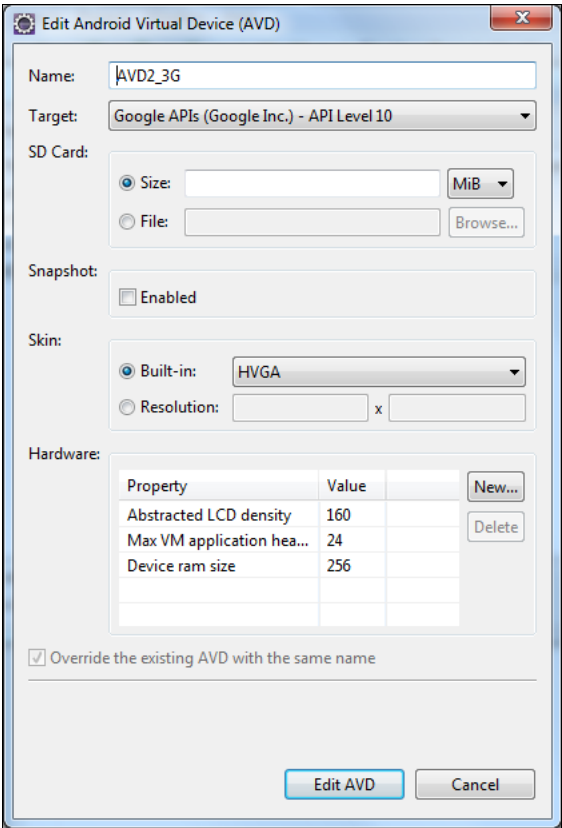


Рис. 36.1. Добавление в эмулятор поддержки Google API

Типы провайдеров местоположения

В настоящее время на мобильном устройстве для определения его местоположения используются несколько систем глобального позиционирования. Наиболее распространенным провайдером является GPS (Global Positioning System) — американская система глобального позиционирования, которая позволяет в любом месте Земли определить местоположение объекта. Кроме GPS, существуют и другие системы, пока обладающие меньшими возможностями, чем GPS: ГЛОНАСС — российская и Galileo — европейская система глобального позиционирования.

Принцип работы всех систем глобального позиционирования заключается в определении местоположения путем измерения расстояний от мобильного устройства до спутников. Расстояние вычисляется по времени задержки приема сигнала от спутников до мобильного устройства.

Кроме GPS, мобильное устройство на платформе Android также поддерживает определение местоположения с помощью сетевого провайдера, используя сигнал от станций сотовой связи. Еще один вариант определения местоположения — через точки доступа Wi-Fi-соединений. Однако следует иметь в виду, что сетевые провайдеры местоположения менее надежны, чем спутниковые системы глобального позиционирования. Для

этих типов провайдеров в классе `LocationManager` определены константы: `GPS_PROVIDER` и `NETWORK_PROVIDER`.

Для определения провайдеров местоположения, используемых в конкретном месте, где находится мобильное устройство, в классе `LocationManager` существует несколько методов:

- ❑ `getProvider()` — возвращает объект класса `LocationProvider` (об этом классе будет рассказано позже), который представляет собой детальную информацию, описывающую данного провайдера. В качестве параметра этому методу передается строка с именем провайдера;
- ❑ `getAllProviders()` — возвращает список имен всех известных провайдеров местоположения;
- ❑ `getProviders()` — возвращает список имен всех провайдеров местоположения. В метод передается булев параметр `enabledOnly`, чтобы можно было получить список имен провайдеров местоположения, которые доступны на данный момент для мобильного устройства;
- ❑ `getProviders(Criteria criteria, boolean enabledOnly)` — возвращает список имен провайдеров местоположения, которые удовлетворяют заданным критериям. Критерии представляют собой объект класса `Criteria`, о котором будет рассказано позже в этой главе.

Провайдер местоположения представляется классом `LocationProvider`. Этот класс описывает характеристики и особенности конкретного провайдера местоположения и имеет следующие методы:

- ❑ `getName()` — возвращает имя провайдера;
- ❑ `getAccuracy()` — возвращает константу, определяющую точность этого провайдера;
- ❑ `getPowerRequirement()` — возвращает требуемое значение мощности сигнала, необходимое для устойчивой работы с этим провайдером;
- ❑ `hasMonetaryCost()` — возвращает `true`, если использование этого провайдера является платной услугой;
- ❑ `meetsCriteria()` — возвращает `true`, если этот провайдер удовлетворяет критериям, которые передаются в качестве параметра этому методу;
- ❑ `requiresCell()` — возвращает `true`, если провайдер требует доступа к сотовой сети, например использует идентификатор станции сотовой связи;
- ❑ `requiresNetwork()` — возвращает `true`, если провайдер требует доступа к мобильной сети Интернет;
- ❑ `requiresSatellite()` — возвращает `true`, если провайдер требует доступа к спутниковым системам позиционирования, например GPS;
- ❑ `supportsAltitude()` — возвращает `true`, если провайдер в состоянии предоставить высотную информацию;
- ❑ `supportsSpeed()` — возвращает `true`, если провайдер в состоянии предоставить информацию о скорости перемещения мобильного устройства.

Часть этих методов совпадает с методами класса `Criteria`, определяющего критерии (требования к провайдеру местоположения: точность определения координат, под-

держку различных режимов и другие его параметры) для возможности использования данного провайдера в приложении. Класс `Criteria` используется для определения наиболее подходящего провайдера местоположения в случае, если доступно сразу несколько провайдеров, и будет подробнее рассматриваться далее в этой главе.

Разрешения для работы с провайдерами местоположения

Чтобы приложение могло запрашивать провайдера и обновлять местоположение мобильного устройства в зависимости от типа провайдера местоположения, который будет использоваться, в приложение необходимо добавить разрешения `ACCESS_FINE_LOCATION` для провайдеров `GPS_PROVIDER` и `ACCESS_COARSE_LOCATION`. Разрешение `ACCESS_COARSE_LOCATION` используется для провайдеров с низкой точностью измерения, типа `NETWORK_PROVIDER`. Разрешение `ACCESS_FINE_LOCATION` используется для обоих типов провайдеров. Можно использовать в манифесте одновременно оба разрешения. Без этих разрешений при запуске ваше приложение сгенерирует исключение.

Приложение для поиска доступных провайдеров

Сейчас мы используем рассмотренные классы для написания простого приложения, которое будет осуществлять поиск провайдеров и проверять их доступность. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `Location_FindProviders`;
- ☐ **Application name** — `Location Providers info`;
- ☐ **Package name** — `com.samples.locationproviders`;
- ☐ **Create Activity** — `LocationProvidersActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch036_Location_FindProviders`.

В файл манифеста приложения `AndroidManifest.xml` добавьте разрешение `android.permission.ACCESS_FINE_LOCATION` для возможности доступа к сервисам местоположения. Код файла манифеста приложения `AndroidManifest.xml` представлен в листинге 36.1.

Листинг 36.1. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.locationproviders"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".LocationProvidersActivity"
            android:label="@string/app_name">
```

```

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-sdk android:minSdkVersion="7" />
</manifest>

```

Файл компоновки главного окна приложения будет содержать элемент `TextView` с идентификатором `text` для вывода информации о провайдерах местоположения. Он аналогичен таким же файлам компоновки, которые мы использовали в приложениях в предыдущих главах.

В коде класса `LocationProvidersActivity` главного окна приложения в обработчике события `onCreate()` мы получаем объект `LocationManager`, с помощью метода `getAllProviders()` получаем список провайдеров и выводим его на экран. Код класса `LocationProvidersActivity` представлен в листинге 36.2.

Листинг 36.2. Файл класса окна приложения `LocationProvidersActivity.java`

```

package com.samples.locationproviders;

import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class LocationProvidersActivity extends Activity {

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

        TextView text = (TextView) this.findViewById(R.id.text);

        LocationManager manager = (LocationManager) getSystemService(
            Context.LOCATION_SERVICE);
        text.append("List of Location providers\n");

        List<String> providers = manager.getAllProviders();
        for (int i = 0; i < providers.size(); i++) {
            String provider = providers.get(i);

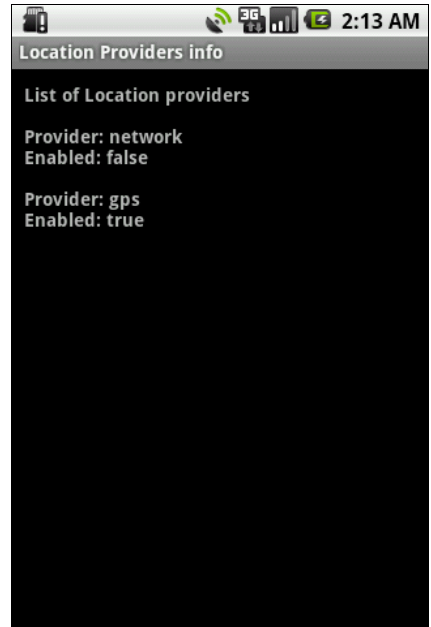
```

```
text.append("\nProvider: " + provider);
text.append("\nEnabled: " +
    manager.isProviderEnabled(provider) + "\n");
}
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Внешний вид приложения для поиска провайдеров местоположения представлен на рис. 36.2.

Как вы видите, для эмулятора доступен провайдер GPS, и, в принципе, можно тестировать приложения для работы с сервисами местоположения на эмуляторе мобильного устройства Android.

Рис. 36.2. Отображение провайдеров местоположения



Определение лучшего провайдера

У мобильных устройств Android может быть одновременный доступ к нескольким разным провайдерам местоположения. Возможности, предоставляемые провайдерами, могут отличаться. У одного провайдера может быть лучшая точность, чем у другого. Некоторые провайдеры могут быть бесплатными, другие — платными. Поэтому в приложениях для навигации необходимо предусмотреть функциональность для определения лучшего провайдера местоположения.

Критерии для определения лучшего провайдера

Когда для пользователя мобильного устройства доступны сразу несколько провайдеров местоположения, можно делать выбор наиболее подходящего провайдера, основываясь на наборе критериев — точность, скорость, стоимость и т. д.

В классе `LocationProvider` есть метод `getBestProvider()`, который возвращает имя провайдера, наиболее полно удовлетворяющего требуемым критериям. Критерии передаются в качестве первого параметра и представляют собой объект класса `Criteria`, который определяет характеристики провайдера.

Класс `Criteria` содержит набор методов, с помощью которых можно читать или задавать характеристики провайдера. Некоторые из методов аналогичны методам класса `LocationProvider`, например, пара методов `getAccuracy()` и `setAccuracy()`, соответственно возвращающих и устанавливающих константу, определяющую точность этого провайдера.

Константа, характеризующая точность провайдера, имеет всего два значения, которые определены в классе `Criteria`:

- ☐ `ACCURACY_FINE` — высокий уровень точности определения местоположения;
- ☐ `ACCURACY_COARSE` — низкий уровень точности определения местоположения.

В классе `Criteria` также есть и более специфические методы для получения характеристик точности определения местоположения:

- ☐ `getHorizontalAccuracy()` — возвращает точность определения горизонтальных координат: географической широты и долготы;
- ☐ `getVerticalAccuracy()` — возвращает точность определения высоты объекта над уровнем моря;
- ☐ `getBearingAccuracy()` — возвращает точность определения пеленгации;
- ☐ `getSpeedAccuracy()` — возвращает точность определения скорости перемещения.

Эти методы работают с набором целочисленных значений, характеризующих точность и определяемых следующими константами:

- ☐ `ACCURACY_HIGH`;
- ☐ `ACCURACY_MEDIUM`;
- ☐ `ACCURACY_LOW`;
- ☐ `NO_REQUIREMENT`.

В классе `Criteria` есть пара методов для определения требования к мощности принимаемого от провайдера сигнала:

- ☐ `getPowerRequirement()`;
- ☐ `setPowerRequirement(int level)`.

Эти методы характеризуют уровень мощности (высокий, средний, низкий), который определяется тремя целочисленными константами:

- ☐ `POWER_HIGH`;
- ☐ `POWER_MEDIUM`;
- ☐ `POWER_LOW`.

Класс `Criteria` также содержит набор булевых методов для чтения и определения требований к характеристикам провайдера местоположения:

- ☐ `isAltitudeRequired()` — поддержка провайдером определения высоты;
- ☐ `isBearingRequired()` — поддержка провайдером пеленгации;
- ☐ `isCostAllowed()` — использование провайдера является платной услугой;
- ☐ `isSpeedRequired()` — поддержка провайдером определения скорости перемещения.

Класс `Criteria` помогает определить и использовать в приложениях наиболее подходящего провайдера местоположения, если в данном регионе доступны сразу несколько провайдеров. Например, это можно сделать следующим способом:

```
// Получаем экземпляр locationManager
LocationManager manager = (LocationManager) getSystemService(
    Context.LOCATION_SERVICE);

// Создаем объект Criteria
Criteria criteria = new Criteria();
// Задаем требуемые характеристики провайдера
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
criteria.setAltitudeRequired(true);
criteria.setBearingRequired(true);
criteria.setCostAllowed(true);
criteria.setSpeedRequired(true);

// Определяем лучшего провайдера
String bestProvider = manager.getBestProvider(criteria, true);
```

Поиск и определение лучшего провайдера в приложении

Рассмотрим теперь практическое использование класса `Criteria`. Создадим приложение, способное определять лучшего провайдера, удовлетворяющего заданным критериям функциональности и точности определения местоположения. Например, зададим требуемую точность провайдера `ACCURACY_FINE` и уровень мощности `POWER_LOW`.

Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `Location_BestProviders`;
- ☐ **Application name** — `Find best providers`;
- ☐ **Package name** — `com.samples.location.findbestproviders`;
- ☐ **Create Activity** — `FindBestProvidersActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch036_Location_BestProviders`.

В файл манифеста приложения `AndroidManifest.xml` добавьте разрешения `android.permission.ACCESS_FINE_LOCATION` и `android.permission.INTERNET`, как показано в листинге 36.3.

Листинг 36.3. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.locationproviders">
```

```

    android:versionCode="1"
    android:versionName="1.0">
    ...
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
        android:name="android.permission.INTERNET" />
</manifest>

```

Файл компоновки главного окна приложения будет содержать элемент `TextView` с идентификатором `text` для вывода информации. Он аналогичен файлам компоновки, которые мы использовали в приложениях в предыдущих главах.

В коде класса `FindBestProvidersActivity` главного окна приложения используем объект класса `Criteria` для установки требований к провайдеру местоположения и с помощью метода `getBestProvider()` определяем наиболее подходящего провайдера. Код класса `FindBestProvidersActivity` представлен в листинге 36.4.

Листинг 36.4. Файл класса окна приложения `FindBestProvidersActivity.java`

```

package com.samples.location.findbestproviders;

import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.location.Criteria;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class FindBestProvidersActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        final TextView text = (TextView)findViewById(R.id.text);

        LocationManager manager = (LocationManager) getSystemService(
            Context.LOCATION_SERVICE);

        text.append("List of Location providers\n");

        // Определяем всех провайдеров местоположения
        List<String> providers = manager.getAllProviders();
        for (int i = 0; i < providers.size(); i++) {
            String provider = providers.get(i);
            text.append("\nProvider: " + provider);
            text.append("\nEnabled: " +
                manager.isProviderEnabled(provider) + "\n");
        }
    }
}

```

```

// Создаем объект Criteria и задаем
// требуемые характеристики провайдера
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
criteria.setAltitudeRequired(true);
criteria.setBearingRequired(true);
criteria.setCostAllowed(true);
criteria.setSpeedRequired(true);

// Определяем лучшего провайдера
String bestProvider = manager.getBestProvider(criteria, true);
text.append("\nBest provider: " + bestProvider);
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. Приложение при запуске выполнит поиск провайдеров местоположения и выведет на экран эмулятора информацию об имени провайдера и его доступности. Внешний вид приложения представлен на рис. 36.3.

Правда, в данном случае выбирать было не из чего: доступен всего один провайдер — GPS, но, конечно, существуют места, где возможно использование нескольких провайдеров местоположения.

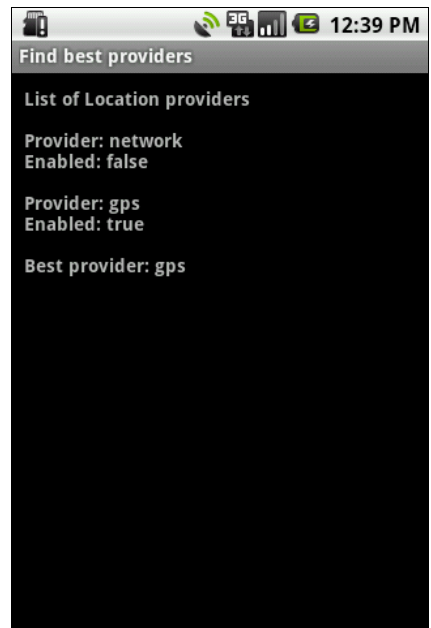


Рис. 36.3. Приложение для определения лучшего провайдера местоположения

Использование эмулятора Android для тестирования приложений

Если вы создаете приложение, работающее с сервисами местоположения, то для отладки необязательно использовать реальное мобильное устройство, в большинстве случаев можно обойтись эмулятором Android. Как вы видели на примере приложения из предыдущего раздела, эмулятор способен моделировать GPS-провайдер местоположения.

При запуске приложения на реальном мобильном устройстве провайдер будет всегда возвращать одни и те же координаты, если вы остаетесь на месте и не перемещаетесь вместе с мобильным устройством. При тестировании приложения, конечно, удобнее оставаться дома, но иметь возможность менять свои координаты.

При тестировании приложения в представлении **Emulator Control** (при доступе из IDE Eclipse) можно устанавливать виртуальные координаты долготы и широты местоположения. На вкладке **Manual** этого представления есть два текстовых поля:

- ❑ **Longitude** — географическая долгота. Для западного полушария значение долготы имеет положительное значение, для восточного полушария — отрицательное;
- ❑ **Latitude** — географическая широта. Для северного полушария значение широты имеет положительное значение, для южного полушария — отрицательное.

Кнопкой **Send** можно отправлять координаты на эмулятор, обновляя географические координаты в приложении (рис. 36.4).

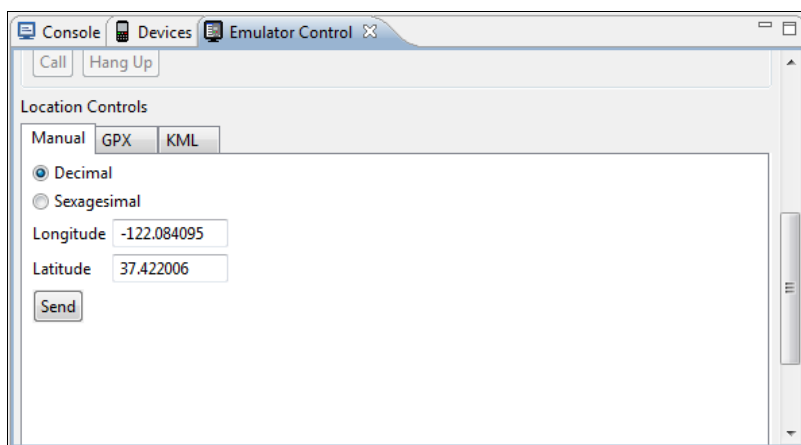


Рис. 36.4. Задание координат в представлении **Emulator Control**

Так как в начале главы мы уже создали эмулятор с поддержкой Google API, то этот вариант виртуального мобильного устройства уже имеет встроенное приложение Maps, использующее сетевой сервис Google Maps для отображения заданных координат на карте, как представлено на рис. 36.5.

Эмулятор Android может моделировать сервисы местоположения, но, конечно, имеет некоторые ограничения. Например, эмулятор не может соединиться с реальной спутниковой системой глобального позиционирования. Также, конечно, нельзя использовать метод триангуляции с помощью базовых станций сотовой связи или точек доступа сетей Wi-Fi.



Рис. 36.5. Отображение местоположения на карте

Определение координат

В классе `LocationManager` есть метод `getLastKnownLocation()`, который возвращает координаты последнего известного местоположения мобильного устройства, полученные от конкретного провайдера, имя которого передается в этот метод в качестве входного параметра.

Эти координаты представляют собой объект класса `Location`, который служит для описания физического местоположения мобильного устройства. В классе `Location` определено множество методов для чтения и установки характеристик, определяющих местоположение мобильного устройства. Вот, например, основные методы для определения физических координат:

- ❑ `getLatitude()` — возвращает географическую широту;
- ❑ `getLongitude()` — возвращает географическую долготу;
- ❑ `getAltitude()` — возвращает высоту при условии, что провайдер поддерживает измерение высоты;
- ❑ `getAccuracy()` — возвращает точность измерения координат в метрах.

С помощью класса `Location` можно также задавать физические координаты местоположения. Для этого в классе `Location` существуют методы, аналогичные перечисленным, только с приставкой `set`.

Обновление местоположения

Поскольку мобильный телефон может постоянно перемещаться вместе со своим пользователем, в приложении необходимо предусмотреть возможность обновления координат, а также возможность изменения состояния доступа к провайдеру местоположения.

Для отслеживания этих изменений в пакете `android.location` предусмотрен интерфейс `LocationListener`. В этом интерфейсе объявлено несколько методов, которые необходимо переопределить в приложении:

- ❑ `onLocationChanged(Location location)` — вызывается при изменении координат местоположения, новое местоположение в виде объекта `Location` передается как параметр;
- ❑ `onProviderDisabled(String provider)` — вызывается, когда провайдер местоположения становится недоступным. В качестве параметра передается имя провайдера;
- ❑ `onProviderEnabled(String provider)` — вызывается, когда провайдер местоположения вновь становится доступным. В качестве параметра передается имя провайдера;
- ❑ `onStatusChanged(String provider, int status, Bundle extras)` — вызывается при любом изменении статуса провайдера. В качестве параметров передаются имя провайдера, статус провайдера, определяющий его доступность, и Extra-параметр, содержащий количество спутников, используемых для определения местоположения. Статус провайдера определяется тремя константами:
 - `AVAILABLE` — провайдер полностью доступен;
 - `TEMPORARY_UNAVAILABLE` — провайдер временно недоступен;
 - `OUT_OF_SERVICE` — провайдер недоступен.

Отслеживание изменения статуса провайдера местоположения актуально для мобильных устройств. На уровень сигнала оказывают влияние перемещение мобильного телефона, нахождение в помещениях, электромагнитные поля и даже изменение погоды (например, облачность может снизить уровень сигнала от спутников системы глобального позиционирования). Если провайдеров в данной местности несколько, то при недоступности одного из них приложение может переключиться на использование другого провайдера, который имеет более низкое качество определения местоположения и не был выбран при поиске лучшего провайдера, тем не менее, он может быть использован в качестве запасного провайдера местоположения.

Создать экземпляр `LocationListener` в коде приложения для прослушивания изменений местоположения и состояния провайдера можно следующим образом:

```
LocationListener listener = new LocationListener() {

    public void onLocationChanged(Location loc) {
        // Действия при изменении местоположения
    }

    @Override
    public void onProviderDisabled(String arg0) {
        // Действия при недоступности провайдера
    }

    @Override
    public void onProviderEnabled(String arg0) {
        // Действия при получении доступа к провайдеру
    }

    @Override
    public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
        // Действия при изменении состояния провайдера
    }
};
```

Чтобы приложение могло отслеживать изменения местоположения и состояния провайдера, объект `LocationListener` необходимо зарегистрировать. Для запуска мониторинга изменений местоположения в классе `LocationManager` предусмотрено два метода:

- ❑ `requestLocationUpdates()` — регистрирует объект `LocationListener` для постоянного прослушивания изменений;
- ❑ `requestSingleUpdate()` — регистрирует объект `LocationListener` для одноразового прослушивания изменений.

Эти методы перегружены и имеют несколько отличающихся наборов входных параметров, но каждый из методов обязательно принимает одним из параметров экземпляр класса `LocationListener`. Например, так можно зарегистрировать созданный ранее объект `LocationListener`:

```
LocationManager manager;
...
manager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER, 0, 0, listener);
```

Приложение для мониторинга изменений координат и состояния провайдера

Используя описанным ранее способом интерфейс `LocationListener`, создадим приложение для отслеживания изменений местоположения и состояния провайдера. Для этого создайте в IDE Eclipse новый проект Android (можете просто изменить предыдущий проект) и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `Location_GetCoordinates`;
- ☐ **Application name** — `Get coordinates`;
- ☐ **Package name** — `com.samples.location.getcoordinates`;
- ☐ **Create Activity** — `GetCoordinatesActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch036_Location_GetCoordinates`.

В файл манифеста приложения `AndroidManifest.xml` добавьте разрешения, как вы уже делали ранее для приложений в этой главе (см. листинг 36.3). Файл компоновки главного окна приложения будет содержать единственный элемент `TextView` с идентификатором `text` для вывода информации.

В классе `GetCoordinatesActivity` будет использоваться объект `LocationListener` для отслеживания изменений в координатах местоположения и обновления вывода координат на экран мобильного устройства.

Файл класса окна приложения `GetCoordinatesActivity` представлен в листинге 36.5.

Листинг 36.5. Файл класса окна приложения `GetCoordinatesActivity.java`

```
package com.samples.location.getcoordinates;

import android.app.Activity;
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class GetCoordinatesActivity extends Activity {

    private LocationManager manager;
    private TextView text;

    private LocationListener locListaner = new LocationListener() {

        public void onLocationChanged(Location loc) {
            // Выводим на экран новые координаты
            printLocation(argLocation);
        }
    }
}
```



```
@Override
public void onProviderDisabled(String arg0) {
    // Выводим сообщение о недоступности провайдера
    printLocation(null);
}

@Override
public void onProviderEnabled(String arg0) { }

@Override
public void onStatusChanged(String arg0, int arg1, Bundle arg2) { }
};

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

    text = (TextView)findViewById(R.id.text);

    manager = (LocationManager) getSystemService(
        Context.LOCATION_SERVICE);
    manager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER, 0, 0, locListaner);

    Location loc = manager.getLastKnownLocation(
        LocationManager.GPS_PROVIDER);
    printLocation(loc);
}

// Вывод координат на экран
private void printLocation(Location loc) {
    if (loc != null)
    {
        text.setText("Longitude:\t" + loc.getLongitude() +
            "\nLatitude:\t" + loc.getLatitude());
    }
    else {
        text.setText("Location unavailable");
    }
}
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Внешний вид приложения представлен на рис. 36.6.

Протестируйте приложение, используя для этого представление **Emulator Control** в IDE Eclipse, изменяя в текстовых полях **Longitude** и **Latitude** долготу и широту местоположения. При этом должно происходить обновление значения текущих географических координат в программе.



Рис. 36.6. Приложение для отображения координат

Резюме

В этой главе мы рассмотрели только общие основы навигации, изучили поиск и выбор провайдеров местоположения, критерии для выбора лучшего провайдера и отслеживание изменений местоположения мобильного устройства.

В следующей главе мы займемся сетевым сервисом Geocoding, который позволяет находить соответствие между координатами местоположения мобильного устройства и физическим адресом.



ГЛАВА 37

Сервис Geocoding

Сетевой сервис Geocoding позволяет проводить соответствие между географическими координатами карты и физического адреса объекта: например, соотносить координаты карты долготы/широты с названием населенного пункта, улицей, номером дома, почтовым индексом и др. Сервис Geocoding предоставляет необходимую информацию из своей базы данных.

С этой сетевой службой можно взаимодействовать из приложения, установленного на мобильном устройстве. Полученную информацию можно отображать в текстовом виде или на карте. Однако при использовании этого сервиса необходимо учитывать, что база данных Geocoding очень зависит от региона, для некоторых районов Земли эта база пока еще не заполнена.

Использование Geocoding

Сервис Geocoding может использоваться в двух режимах:

- ☐ Forward Geocoding (прямой Geocoding) — позволяет находить физические координаты широты и долготы для заданного места;
- ☐ Reverse Geocoding (обратный Geocoding) — позволяет находить адрес для заданных координат широты и долготы.

Оба режима Geocoding возвращают список объектов класса `Address`. В каждом объекте `Address` будет содержаться столько информации, сколько имеется в базе данных сервиса Geocoding для данного места. Объект `Address` может включать в себя географическую широту, долготу, номер дома и другую информацию.

Объект класса `Address` имеет множество методов для описания объекта, находящегося в данном месте, и представляет собой набор строк, характеризующих этот географический объект. Вот наиболее часто используемые методы этого класса:

- ☐ `getCountryName()` — возвращает название страны, например "Russia";
- ☐ `getCountryCode()` — возвращает код страны для данного адреса, например "RU";
- ☐ `getAdminArea()` — возвращает название административного субъекта (области для РФ или штата для США);
- ☐ `getSubAdminArea()` — возвращает название административного подразделения (области для РФ или графства для США);

- ❑ `getFeatureName()` — возвращает дополнительное название объекта, например "Эрмитаж";
- ❑ `getLocality()` — возвращает название местности, где расположен данный объект;
- ❑ `getPostalCode()` — возвращает почтовый код адреса, например "94110";
- ❑ `getAddressLine()` — возвращает строку адреса: название города, улицы, номер дома;
- ❑ `getPhone()` — возвращает телефонный номер.

Если необходима более детализированная информация о государстве или языке, используемом в этой местности, можно использовать вызов метода `getLocale()`. Этот метод возвращает объект класса `Locale`, с помощью которого можно получить как полное название языка или региона, так и его кодировку: 2-буквенный код (alpha-2) или 3-буквенный код (alpha-3) по стандарту ISO. Для получения этой информации в классе `Locale` имеется набор методов:

- ❑ `getCountry()` — возвращает код государства;
- ❑ `getDisplayCountry()` — возвращает название государства;
- ❑ `getDisplayLanguage()` — возвращает название языка;
- ❑ `getDisplayName()` — возвращает строку, содержащую название языка, государства или региона;
- ❑ `getISO3Country()` — возвращает 3-буквенный код государства по стандарту ISO;
- ❑ `getISO3Language()` — возвращает 3-буквенный код языка по стандарту ISO;
- ❑ `getLanguage()` — возвращает название языка для данного региона.

Все перечисленные методы могут возвращать `null` (для методов класса `Address`) или пустую строку (для методов класса `Locale`), если в базе данных сервиса Geocoding отсутствует запрашиваемая информация или она является неполной. Обязательно учитывайте это при выводе информации пользователю.

Reverse Geocoding

Сначала мы рассмотрим Reverse Geocoding, т. е. получение адреса путем ввода физических координат широты и долготы. Чтобы найти адрес для заданных координат, необходимо в метод `getFromLocation()` передать значения широты и долготы, а также целое число, представляющее собой максимальное число выводимых адресов для данного места. Этот метод вернет список возможных адресов соответствия (список может быть достаточно длинным, особенно для США, поэтому вводят ограничение по количеству выводимых адресов):

```
Geocoder geocoder = new Geocoder(getApplicationContext());
LocationManager manager = (LocationManager) getSystemService(
    Context.LOCATION_SERVICE);
Location loc = manager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
if (loc != null)
{
    // Получаем список адресов и ограничиваем кол-во выводимых адресов 10-ю
    List<Address> list = geocoder.getFromLocation(
        loc.getLatitude(), loc.getLongitude(), 10);
```

```
for (int i = 0; i < list.size(); i++) {  
    // Получаем объект Address, представляющий конкретный адрес  
    // в базе данных сервиса Geocoding  
    Address address = list.get(i);  
    ...  
}  
}
```

Если сервис Geocoding не в состоянии найти адрес для указанных координат в своей базе данных, он возвратит `null`.

Точность и степень детализации адреса зависят от качества информации, находящейся в базе данных сервиса. В результате полнота и качество полученной информации может широко варьироваться в зависимости от страны и местности.

Создадим приложение, использующее сервис Geocoding и осуществляющее поиск адреса по заданным координатам. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — ReverseGeocoding;
- ☐ **Application name** — Reverse Geocoding;
- ☐ **Package name** — com.samples.location.reversegeocoding;
- ☐ **Create Activity** — GeocoderActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch37_ReverseGeocoding.

Поскольку Geocoding является сетевым сервисом, для приложения требуется использовать разрешение для доступа в сеть. Для этого в файл манифеста приложения `AndroidManifest.xml` добавьте разрешение `android.permission.INTERNET`. Не забудьте также добавить разрешения `android.permission.ACCESS_COARSE_LOCATION` и `android.permission.ACCESS_FINE_LOCATION` для работы с сервисами местоположения.

Файл разметки окна приложения `main.xml` будет содержать только виджет `TextView` с идентификатором `text` для вывода адреса.

В коде класса главного окна приложения используется объект `LocationListener` для отслеживания изменений координат, который мы уже изучили в предыдущей главе. В методе `printLocation()` мы получаем список доступных адресов для данной точки и выводим их на экран мобильного устройства. Код класса `GeocoderActivity` представлен в листинге 37.1.

Листинг 37.1. Файл класса окна приложения `GeocoderActivity.java`

```
package com.samples.location.reversegeocoding;  
  
import java.io.IOException;  
import java.util.List;  
  
import android.app.Activity;  
import android.content.Context;  
import android.location.Address;
```

```
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class GeocoderActivity extends Activity {

    private LocationManager manager;
    private TextView text;
    private Geocoder geocoder;

    private LocationListener listener = new LocationListener() {
        public void onLocationChanged(Location argLocation) {
            printLocation(argLocation);
        }

        @Override
        public void onProviderDisabled(String arg0) {
            printLocation(null);
        }

        @Override
        public void onProviderEnabled(String arg0) {}

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle arg2) {}
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        text = (TextView)findViewById(R.id.text);

        manager = (LocationManager) getSystemService(
            Context.LOCATION_SERVICE);
        manager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER, 0, 0, listener);
        Location loc = manager.getLastKnownLocation(
            LocationManager.GPS_PROVIDER);

        // Создаем экземпляр класса Geocoder
        geocoder = new Geocoder(getApplicationContext());

        printLocation(loc);
    }
}
```

```
// Вывод информации о местоположении на экран
private void printLocation(Location loc) {
    if (loc != null) {
        text.setText("Longitude:\t" + loc.getLongitude() +
            "\nLatitude:\t" + loc.getLatitude());

        try {
            // Получаем список адресов
            List<Address> list = geocoder.getFromLocation(
                loc.getLatitude(), loc.getLongitude(), 20);

            for (int i = 0; i < list.size(); i++) {
                Address address = list.get(i);

                // Выводим информацию на экран
                text.append("\nAddress# " + i +
                    "\n\tLocality: " + address.getLocality() +
                    "\n\tCountryName: " + address.getCountryName() +
                    "-" + address.getCountryCode() +
                    "\n\tFeatureName: " + address.getFeatureName() +
                    "\n\tPostalCode: " + address.getPostalCode()
                );
            }
        } catch (IOException e) {
            Toast.makeText(getApplicationContext(),
                e.toString(), Toast.LENGTH_LONG).show();
        }
    }
    else {
        text.setText("Location unavailable");
    }
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Если в представлении **Emulator Control** не менять заданные по умолчанию координаты (**Longitude:** -122.084095 и **Latitude:** 37.422005), мы увидим адрес штаб-квартиры Google в США. Приложение выводит список из нескольких адресов с различными вариантами представления этого местоположения, как показано на рис. 37.1.

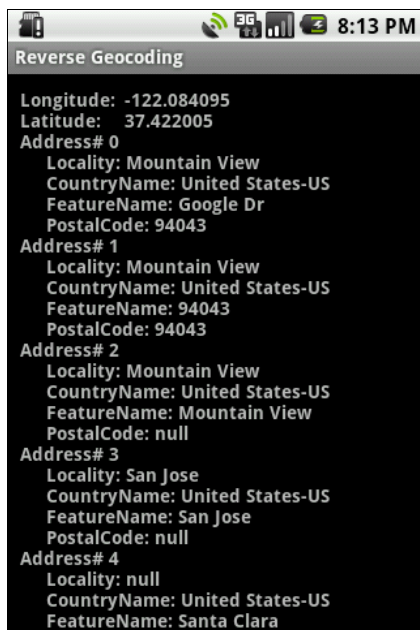


Рис. 37.1. Reverse Geocoding — поиск адреса по заданным координатам

Поэкспериментируйте с приложением, задавая разные координаты. Кстати, обратите внимание на количество выводимой информации для разных регионов мира: наиболее полная информация для адреса будет представлена для США и Европы, включая и европейскую часть России, для экзотических и пустынных регионов список возвращаемых адресов может быть вообще пустым.

Отображение местоположения на карте

Теперь мы подошли к более интересному моменту: как отобразить место с заданными координатами на карте? Это сделать очень просто, вы уже это делали в предыдущих приложениях в этой книге, когда вызывали стандартные Activity (например, для набора телефонного номера). Android предоставляет стандартный Activity с картой. Для его вызова необходимо создать объект Intent с параметром Intent.ACTION_VIEW и передать его в метод startActivity(), который отобразит Activity с картой сервиса Google Maps. Однако для отображения карты с заданными координатами местоположения потребуется еще Extra-параметр — URI этого местоположения.

URI для местоположения имеет следующий формат:

```
geo: <latitude>,<longitude>
```

Таким образом, вызвать карту сервиса Google Maps и отобразить на ней заданное местоположение можно так:

```
// Формируем URI из географических координат
Uri uri = Uri.parse(String.format("geo:%f,%f",
    loc.getLatitude(), loc.getLongitude()));

// Создаем объект Intent и вызываем Activity с картой
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

Теперь реализуем это в приложении. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — ReverseGeocodingWithMap;
- ☐ **Application name** — ReverseGeocoding with map;
- ☐ **Package name** — com.samples.location.reversegeocodingmap;
- ☐ **Create Activity** — GeocoderActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch37_ReverseGeocodingWithMap.

В файле компоновки окна, кроме виджета TextView с идентификатором text, будет кнопка для вызова стандартного Activity, содержащего карту. Назначьте для кнопки идентификатор bMap и надпись **Show map**. Код для файла компоновки main.xml представлен в листинге 37.2.

Листинг 37.2. Файл компоновки окна приложения main.xml

```
<?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">

<LinearLayout
    android:layout_height="wrap_content"
    android:layout_width="fill_parent">

    <Button
        android:id="@+id/bMap"
        android:layout_height="wrap_content"
        android:text="Show map"
        android:layout_width="fill_parent"
        android:layout_weight="1"/>
</LinearLayout>

<TextView
    android:id="@+id/text"
    android:text=""
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textStyle="bold"/>

</LinearLayout>
```

В коде класса `GeocoderActivity` главного окна приложения в теле обработчика события нажатия кнопки `bMap_onClick()` будет формироваться `URI` для заданных координат местоположения и вызываться объект `Intent` с параметром `Intent.ACTION_VIEW` для отображения карты с выбранным местоположением. Код класса `GeocoderActivity` представлен в листинге 37.3.

Листинг 37.3. Файл класса окна приложения `GeocoderActivity.java`

```
package com.samples.location.reversegeocodingmap;

import java.io.IOException;
import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
```

```
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class GeocoderActivity extends Activity {

    private LocationManager manager;
    private TextView text;
    private Geocoder geocoder;
    private Button bMap;
    private Location currentLocation;

    private LocationListener listener = new LocationListener() {

        public void onLocationChanged(Location argLocation) {
            currentLocation = argLocation;
            printLocation(currentLocation);
        }

        @Override
        public void onProviderDisabled(String arg0) {
            printLocation(null);
        }

        @Override
        public void onProviderEnabled(String arg0) {}

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle arg2) {}
    };

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

        text = (TextView) findViewById(R.id.text);
        bMap = (Button) findViewById(R.id.bMap);

        bMap.setOnClickListener(bMap_onClick);

        manager = (LocationManager) getSystemService(
            Context.LOCATION_SERVICE);
        manager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER, 0, 0, listener);
        currentLocation = manager.getLastKnownLocation(
            LocationManager.GPS_PROVIDER);

        // Создаем объект Geocoder
        geocoder = new Geocoder(getApplicationContext());
    }
}
```

```

        printLocation(currentLocation);
    }

    public OnClickListener bMap_onClick = new OnClickListener() {
        @Override
        public void onClick(View v) {
            if (currentLocation != null) {
                // Создаем URI для заданных координат
                Uri uri = Uri.parse(String.format("geo:%f,%f",
                    currentLocation.getLatitude(),
                    currentLocation.getLongitude()));

                // Создаем объект Intent и вызываем Activity с картой
                Intent intent = new Intent(Intent.ACTION_VIEW, uri);
                startActivity(intent);
            }
        }
    };

    // Вывод информации о местоположении на экран
    private void printLocation(Location loc) {
        if (loc != null) {
            text.setText("Longitude:\t" + loc.getLongitude() +
                "\nLatitude:\t" + loc.getLatitude());

            try {
                List<Address> list = geocoder.getFromLocation(
                    loc.getLatitude(), loc.getLongitude(), 20);
                for (int i = 0; i < list.size(); i++) {
                    Address address = list.get(i);
                    text.append("\nAddress# " + i +
                        "\n\tLocality: " + address.getLocality() +
                        "\n\tCountryName: " + address.getCountryName() +
                        "-" + address.getCountryCode() +
                        "\n\tFeatureName: " + address.getFeatureName() +
                        "\n\tPostalCode: " + address.getPostalCode()
                    );
                }
            }
            catch (IOException e) {
                Toast.makeText(getApplicationContext(),
                    e.toString(), Toast.LENGTH_LONG).show();
            }
        }
        else {
            text.setText("Location unavailable");
        }
    }
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. Теперь, кроме списка адресов для заданного местоположения, при нажатии кнопки **Show map** приложение отобразит это местоположение на карте, как показано на рис. 37.2.

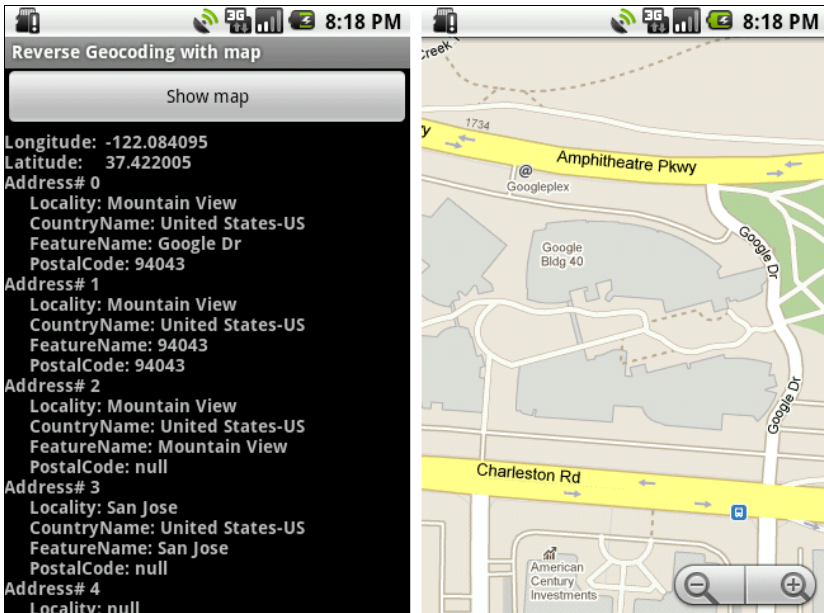


Рис. 37.2. Reverse Geocoding с отображением места на карте

Forward Geocoding

Теперь мы рассмотрим Forward Geocoding, с помощью которого можно задавать адрес или название места и отображать для него координаты широты и долготы. Сделать запрос для Forward Geocoding позволяет вызов метода `getFromLocationName()` класса `Geocoder`. Этот метод, так же как и метод `getFromLocation()`, рассмотренный ранее, возвращает список объектов `Address` и позволяет задавать ограничение для количества выводимых адресов.

Сделать запрос можно следующим образом:

```
Geocoder geocoder = new Geocoder(getApplicationContext());
List<Address> locations =
    geocoder.getFromLocationName("Moscow", 10);
```

Далее, сделав проход по списку в цикле, можно получить весь набор результатов запроса в виде пар широта/долгота:

```
for (int i = 0; i < size; i++) {
    Address loc = locations.get(i);

    // Получаем широту и долготу
    int latitude = loc.getLatitude();
    int longitude = loc.getLongitude();
}
```

Теперь попробуем реализовать Forward Geocoding в приложении. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — ForwardGeocodingWithMap;
- ☐ **Application name** — Forward Geocoding;
- ☐ **Package name** — com.samples.location.geocoderwithmap;
- ☐ **Create Activity** — GeocoderActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch37_FowardGeocodingWithMap.

В файл компоновки окна приложения main.xml добавим следующие виджеты:

- ☐ EditText — текстовое поле для ввода запроса (editSearch);
- ☐ Button — кнопку для запуска поиска (bSearch);
- ☐ ListView — список для вывода результатов запроса (list).

Код файла main.xml представлен в листинге 37.4.

Листинг 37.4. Файл компоновки окна приложения main.xml

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

    <EditText
        android:id="@+id/editSearch"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Enter place name or address" />

    <Button
        android:id="@+id/bSearch"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Search Geocode" />

    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

</LinearLayout>
```

В классе GeocoderActivity в обработчике события нажатия кнопки bSearch_onClick() напишем создание запроса и вывод результатов поиска в список. Для обработки события выбора элемента списка реализуем метод onItemClick(), в котором мы будем вызывать стандартный Activity для отображения найденного местоположения на карте.

Код класса `GeocoderActivity` представлен в листинге 37.5.

Листинг 37.5. Файл класса `GeocoderActivity.java`

```
package com.samples.location.geocoderwithmap;

import java.io.IOException;
import java.util.List;

import android.app.ListActivity;
import android.content.Intent;
import android.location.Address;
import android.location.Geocoder;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;

public class GeocoderActivity extends ListActivity {

    private Geocoder geocoder;
    private EditText editSearch;
    private Button bSearch;
    private List<Address> locations;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        editSearch = (EditText) findViewById(R.id.editSearch);
        bSearch = (Button) findViewById(R.id.bSearch);

        // Создаем объект Geocoder
        geocoder = new Geocoder(getApplicationContext());

        bSearch.setOnClickListener(bSearch_onClick);
    }

    // Обработка события нажатия на кнопку
    public OnClickListener bSearch_onClick = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // Читаем введенный запрос из текстового поля
            String placeName = editSearch.getText().toString();
```

```
try {
    // Получаем объект Location
    locations = geocoder.getLocationName(placeName, 10);

    int size = locations.size();

    if (size == 0) {
        editSearch.setText("");
        editSearch.setHint("No results. Enter new place");
    }

    // Создаем список результатов запроса
    // и выводим его на экран
    String[] list = new String[size];

    for (int i = 0; i < size; i++) {
        Address loc = locations.get(i);
        list[i] = "Latitude: " + loc.getLatitude() +
            "\nLongitude" + loc.getLongitude();
    }

    setListAdapter(
        new ArrayAdapter<String>(getApplicationContext(),
            android.R.layout.simple_list_item_1, list));
    }
    catch (IOException e) {
        Toast.makeText(getApplicationContext(),
            e.toString(), Toast.LENGTH_LONG).show();
    }
}

// Обработка события выбора элемента
// из списка результатов поиска
public void onItemClick(
    ListView parent, View v, int position, long id) {

    Address loc = locations.get(position);
    Uri uri = Uri.parse(String.format("geo:%f,%f",
        loc.getLatitude(), loc.getLongitude()));
    Intent geoMap = new Intent(Intent.ACTION_VIEW, uri);
    startActivity(geoMap);
}
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Введите в текстовое поле, например, "Lund". Сервис Geocoding выдаст несколько вариантов местоположения этого города с разными координатами в совершенно разных точках Земли. Можно уточнить запрос, набрав в текстовом поле "Lund, Sweden", тогда результатом запроса

к сервису будет всего одно местоположение. Внешний вид приложения с отображенными результатами запроса представлен на рис. 37.3.

Если выбрать первый элемент в списке, отобразится карта с городом Lund, расположенным в Швеции, как показано на рис. 37.4 (для тех, кто не в курсе, в этом городе расположено производство Sony Ericsson).

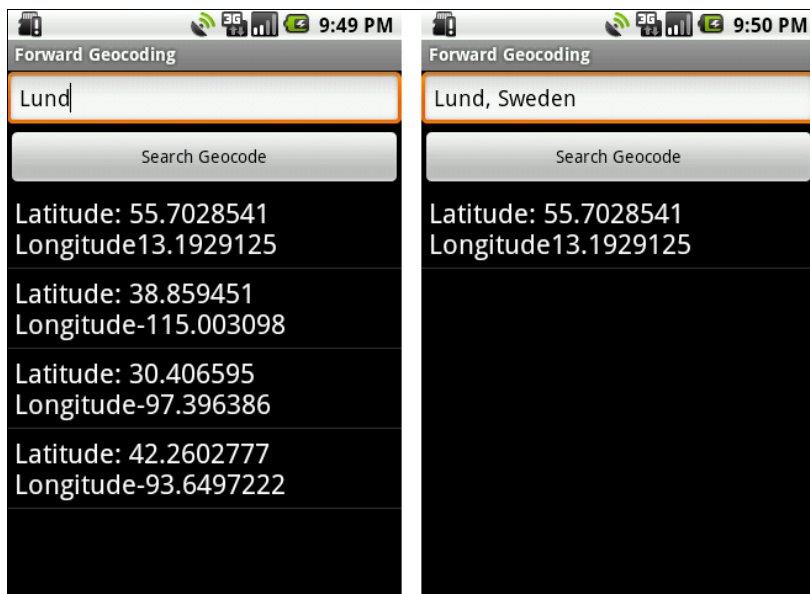


Рис. 37.3. Forward Geocoding — отображение списка возможных координат

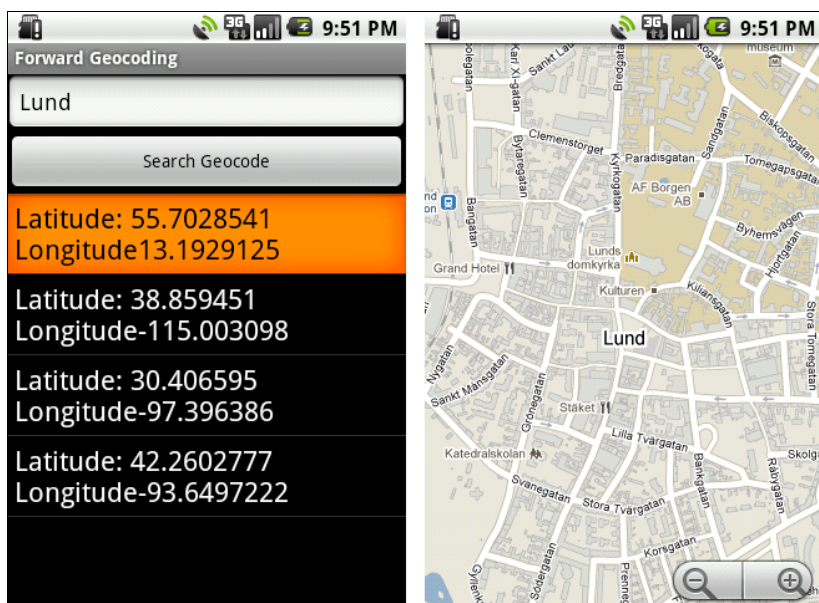


Рис. 37.4. Forward Geocoding с отображением выбранных координат на карте

Резюме

Используя сервис Geocoding вместе с картами сервиса Google Map, вы можете получать информацию о географических координатах объектов и детальную информацию о физическом адресе, которые можно использовать во многих предметных областях.

Пока мы подключали карты с использованием стандартного объекта `Intent`, т. е. вызывали внешний `Activity` с картой Google Map. В следующей главе мы рассмотрим встраивание карт сервиса Google Map в собственные приложения.



ГЛАВА 38

Использование карт Google Maps в приложениях

До сих пор мы использовали стандартный Activity с картой. Однако гораздо удобнее в разрабатываемых приложениях применять встроенные карты сервиса Google Maps. Google предоставляет бесплатную библиотеку для использования карт в своих приложениях. Описание этой библиотеки доступно по адресу:

<https://developers.google.com/maps/documentation/android/>

Однако чтобы получить возможность встраивать карты Google Maps в свои приложения, необходимо зарегистрироваться на сайте Google и получить ключ Maps API Key.

Получение ключа Maps API Key

Процедура регистрации на сайте Google простая и бесплатная. После регистрации вы можете получить ключ для использования карт Google Maps в ваших приложениях.

Ключи сохраняются в специальном хранилище ключей — *keystore*. Это каталог на вашем компьютере, путь к которому вы можете увидеть, если в IDE Eclipse в главном меню выберите **Window | Preferences**. В открывшемся диалоговом окне **Preferences** на панели слева откройте узел **Android | Build**. При этом в правой части окна **Preferences** отобразится панель **Build**, на которой в текстовом поле **Default debug keystore** будет указан путь к каталогу для хранения ключей, как показано на рис. 38.1.

Пока в этом хранилище ключей нет. Для того чтобы получить ключ, надо создать для него сертификат. Сертификат генерируется инструментом *keytool*, входящим в состав Android SDK. Запустите командную строку и введите следующую команду:

```
keytool -list -keystore путь_к_каталогу_keystore
```

Инструмент сгенерирует вам сертификат (рис. 38.2).

Теперь, имея сертификат, можно получить ключ Maps API Key. Для этого зайдите на страницу:

<https://developers.google.com/maps/documentation/android/maps-api-signup/>

На этой странице находятся инструкции по получению Maps API Key, как представлено на рис. 38.3.

После того как вы согласились с условиями использования ключа, введите в текстовое поле **My certificate's MD5 fingerprint** свой сертификат, нажмите кнопку **Generate API Key**, и вам будет сгенерирован ключ.

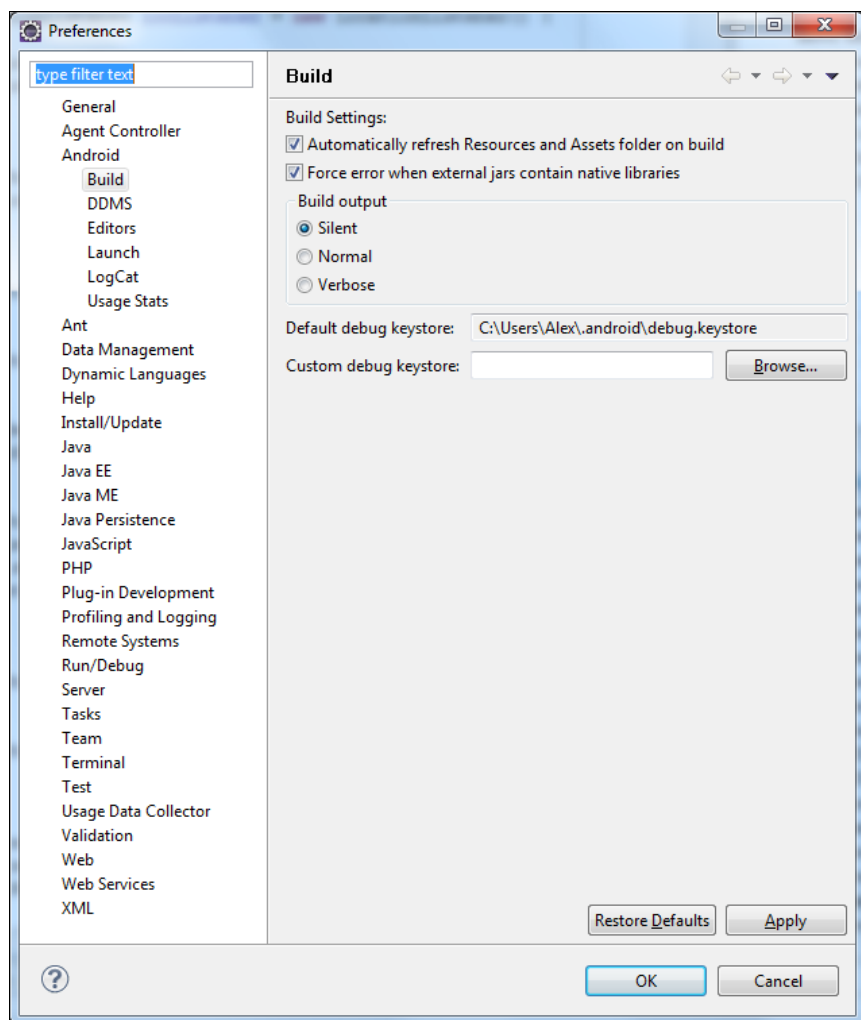


Рис. 38.1. Путь к хранилищу ключей

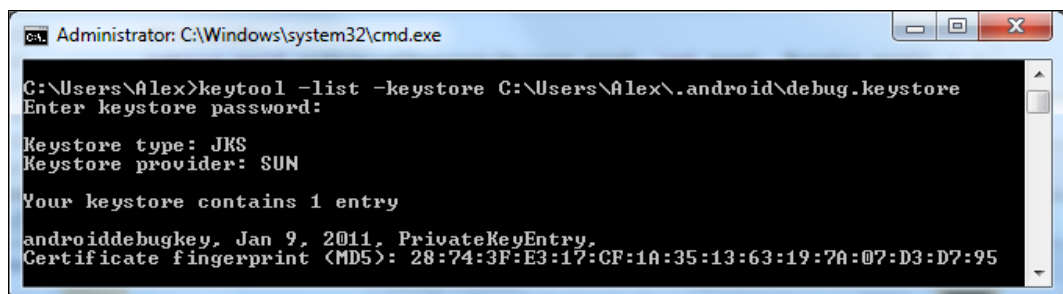


Рис. 38.2. Получение сертификата ключа

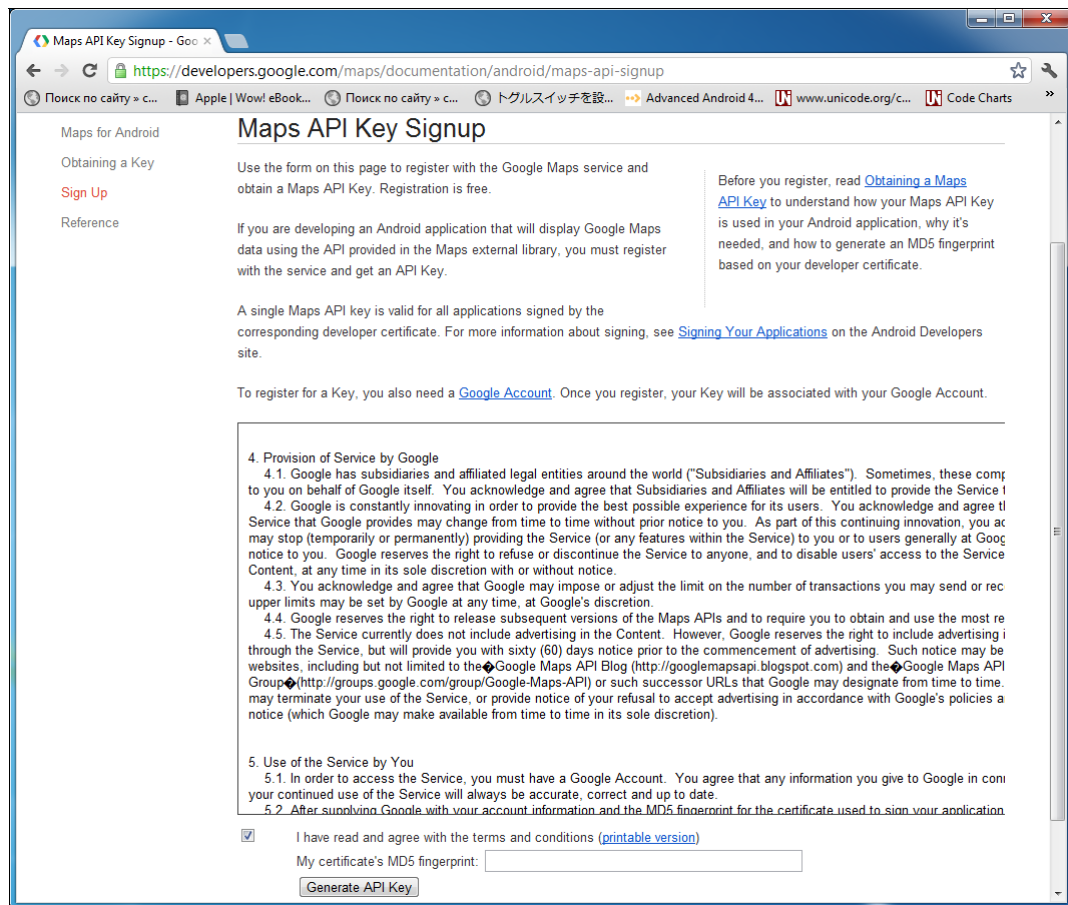


Рис. 38.3. Страница для генерации ключа

После получения ключа вы можете использовать его во всех своих приложениях, которые будут иметь встроенные карты Google Maps.

Базовые классы

Библиотека Google API для работы с картами Google Maps представлена пакетом `com.google.android.maps`. Этот пакет состоит из полутора десятков классов и интерфейсов. Наиболее важными из них являются четыре класса:

- ☐ `MapActivity` — базовый класс для создания Activity со встроенной картой Google Maps;
- ☐ `MapView` — виджет, отображающий карту Google Maps;
- ☐ `MapController` — класс для управления режимами отображения карты (масштабирование, перемещение изображения и др.);
- ☐ `GeoPoint` — класс, представляющий географические координаты в виде пары широты-долготы.

При разработке приложений со встроенными картами Google Maps мы будем активно использовать функциональность, предоставляемую этими классами, поэтому рассмотрим их более подробно.

Так как пакет `com.google.android.maps` является внешней библиотекой и не входит в состав Android SDK, чтобы использовать его в приложении, в файл `AndroidManifest.xml` необходимо добавить ссылку:

```
<uses-library android:name="com.google.android.maps" />
```

Виджет *MapView*

Для отображения карты используется специализированный виджет `MapView`, который представлен в библиотеке `com.google.android.maps` одноименным классом. Карта может быть отображена в различных режимах. Эти режимы устанавливает набор методов класса `MapView`:

- ☐ `setStreetView()` — управление режимом Street View. Это режим панорамного отображения улиц с малой высоты (около 2—3 метров);
- ☐ `setTraffic()` — устанавливает режим отображения трафика на карте;
- ☐ `setSatellite()` — устанавливает карту в режим Satellite и накладывает на нее аэрофотоснимки дорог и названий объектов.

Управление режимом определяется булевым параметром, передаваемым в метод, т. е. значение `true` будет означать включение режима.

В классе есть также группа методов для проверки состояния перечисленных режимов во время выполнения программы:

- ☐ `isStreetView();`
- ☐ `isSatellite();`
- ☐ `isTraffic();`

Для получения значения масштаба карты в классе `MapView` имеются соответствующие методы:

- ☐ `getZoomLevel()` — возвращает текущее значение масштаба карты;
- ☐ `getMaxZoomLevel()` — возвращает максимальный уровень масштабирования для данного местоположения.

В классе `MapView` не предусмотрена функциональность для изменения масштаба. Для этого используется класс `MapController`. Однако в `MapView` предусмотрена возможность отображения встроенного элемента управления масштабированием. Для этого используется метод `setBuiltInZoomControls()`, который будет рассматриваться далее в этой главе.

Для использования карт в приложении вам потребуется вставлять для каждого элемента `MapView` в файл компоновки атрибут `android:apiKey`, которому должен быть присвоен полученный ранее ключ Maps API Key:

```
<com.google.android.maps.MapView  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:apiKey="YOUR_API_KEY"/>
```

Класс *MapView*

Виджет `MapView` может быть размещен только в `Activity`, который наследуется от класса `MapView`.

```
public class MapViewActivity extends MapActivity {  
    ...  
}
```

В производном классе надо создать экземпляр `MapView` в теле метода `onCreate()`, так же как и любой другой виджет.

ПРИМЕЧАНИЕ

В одном процессе желательно использовать только один экземпляр `MapView`, т. е. этот класс не является потокобезопасным.

При создании приложения, работающего с картами Google Maps, нам также понадобятся еще два важных класса: `MapController` и `GeoPoint`

Класс *MapController*

Класс `MapController` используется для управления масштабированием и общим видом объекта `MapView`. С помощью методов этого класса мы можем изменять программно масштаб карты, прокручивать карту и делать много других полезных вещей.

Далее перечислены основные методы для управления режимами отображения и масштабирования карты, которые содержит класс `MapController`:

- ❑ `setCenter()` — устанавливает местоположение, заданное во входном параметре, в центр карты. Параметром является объект `GeoPoint`, который мы рассмотрим в следующем разделе;
- ❑ `animateTo()` — запускает анимацию, т. е. создает анимированное перемещение изображения на карте к заданной точке, которая передается в качестве параметра в виде объекта `GeoPoint`;
- ❑ `stopAnimation()` — останавливает анимацию (обновление или перемещение изображения), которая происходит в данный момент на карте.

Класс `MapController` также содержит несколько методов для управления масштабированием:

- ❑ `setZoom()` — задает масштаб карты. Параметром для этого метода служит целое число в диапазоне от 1 до 21, которое определяет масштаб: чем больше число, тем крупнее масштаб карты;
- ❑ `zoomIn()` — увеличивает масштаб изображения на карте;
- ❑ `zoomOut()` — уменьшает масштаб изображения на карте;
- ❑ `zoomInFixing(int xPixel, int yPixel)` — увеличивает масштаб изображения на карте и одновременно фиксирует точку с заданными координатами в пикселах, если требуется, чтобы данная точка не ушла за пределы видимой части карты;
- ❑ `zoomOutFixing()` — уменьшает масштаб изображения с фиксацией заданной точки на карте;

- `zoomToSpan()` — пытается отрегулировать масштаб карты таким образом, чтобы отображалась заданная область, определяемая параметрами для широты и долготы, которые передаются в качестве входных параметров.

Все перечисленные методы возвращают значение типа `boolean`. Если масштаб карты находится в допустимом диапазоне, методы всегда будут возвращать `true`. Однако если при изменении масштабирования мы достигли верхнего или нижнего предела и пытаемся дальше увеличить (или уменьшить) масштаб, методы будут возвращать значение `false`, которое указывает на невозможность дальнейшего изменения масштаба изображения на карте.

Чтобы получить экземпляр класса `MapController` в коде программы, нужно вызвать метод `getController()` класса `MapView` следующим образом:

```
MapView map = (MapView)findViewById(R.id.map);
MapController controller = map.getController();
```

Объект `MapController` в программном коде обычно используют для установки масштаба карты:

```
controller.setZoom(15);
```

Параметр для метода `setZoom()` задает начальный масштаб изображения на карте. Впоследствии масштаб можно изменять программно с помощью перечисленных ранее методов или посредством встроенного элемента управления масштабированием. Возможности по управлению масштабом изображения на карте и использование элемента управления масштабом мы рассмотрим несколько позже в этой главе.

Класс *GeoPoint*

Класс `GeoPoint` представляет широту и долготу в виде целого числа. В принципе, этот класс похож на класс `Location`, который мы применяли в предыдущих главах, но использует другие единицы измерения.

Единицей измерения координат в классе `GeoPoint` служит 10^{-6} градуса (микроградус), т. е. при работе с этим классом надо масштабировать координаты — умножать географические координаты на 1 000 000 (или делить при обратном преобразовании).

Для чтения значений географических координат в классе `GeoPoint` реализовано два метода:

- `getLatitudeE6()` — возвращает значение географической широты в микроградусах;
- `getLongitudeE6()` — возвращает значение географической долготы в микроградусах.

В приложении объект класса `GeoPoint` нужен для отображения местоположения на карте. Экземпляр `GeoPoint` передается в качестве параметра методам `animateTo()` и `setCenter()` объекта `MapController`.

Однако необходимо учитывать, если мы используем в приложении объект `LocationListener` для отслеживания изменений местоположения, нам необходимо осуществлять конвертацию объекта `Location`, который мы получаем при обновлении местоположения, в объект `GeoPoint`, например, следующим образом:


```

MapController controller;
...
LocationListener locListener = new LocationListener() {

    @Override
    public void onLocationChanged(Location location) {
        double lat = location.getLatitude();
        double lon = location.getLongitude();

        GeoPoint point = new GeoPoint((int)(lat * 1E6), (int)(lon * 1E6));
        controller.animateTo(point);
    }
}

```

Использование *MapView* в приложении

Получив ключ Maps API Key, мы можем заняться разработкой приложений с использованием встроенной карты Google Maps. Сначала создадим простое приложение, использующее *MapView*. Это приложение мы будем совершенствовать, добавляя к нему дополнительную функциональность, на протяжении всей этой главы.

Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — *MapView*;
- ☐ **Application name** — *Embedded Google map*;
- ☐ **Package name** — *com.samples.location.mapview*;
- ☐ **Create Activity** — *MapViewActivity*.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге *Ch38_MapView*.

В файл манифеста приложения *AndroidManifest.xml* добавьте разрешения *android.permission.INTERNET*, *android.permission.ACCESS_COARSE_LOCATION* и *android.permission.ACCESS_FINE_LOCATION*. Также не забудьте добавить ссылку на библиотеку *com.google.android.maps*.

Код файла манифеста представлен в листинге 38.1.

Листинг 38.1. Файл манифеста приложения *AndroidManifest.xml*

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.location.mapview">

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">

        <activity
            android:name=".MapViewActivity"
            android:label="@string/app_name">

```

```

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

<uses-permission
    android:name="android.permission.INTERNET"/>
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"/>

<uses-library android:name="com.google.android.maps"/>
</manifest>

```

В файле компоновки главного окна приложения `main.xml` у нас будет два виджета: `TextView` для отображения координат и `MapView` для загрузки карты. Для виджета `MapView` не забудьте добавить полученный ключ в атрибут `android:apiKey`.

Файл компоновки `main.xml` представлен в листинге 38.2.

Листинг 38.2. Файл компоновки окна приложения `main.xml`

```

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

    <TextView
        android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"/>

    <com.google.android.maps.MapView
        android:id="@+id/map"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="YOUR_API_KEY"/>

</LinearLayout>

```

В коде класса `MapViewActivity` главного окна приложения для работы с картами будем использовать объекты `MapView` и `MapController`, работа с которыми была описана ранее в этой главе.

Код класса главного окна приложения представлен в листинге 38.3.

Листинг 38.3. Файл класса окна приложения MapViewActivity.java

```
package com.samples.location.mapview;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;

import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class MapViewActivity extends MapActivity {
    private MapController controller;
    private TextView text;
    private MapView map;

    private final LocationListener locListener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            printLocation(location);
        }

        @Override
        public void onProviderDisabled(String arg0) {
            printLocation(null);
        }

        @Override
        public void onProviderEnabled(String arg0) { }

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle arg2) { }
    };

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

```

map = (MapView)findViewById(R.id.map);
text = (TextView)findViewById(R.id.text);

controller = map.getController();

// Устанавливаем масштаб карты
controller.setZoom(17);

// Устанавливаем режимы отображения карты
map.setSatellite(true);
map.setStreetView(true);

LocationManager locationManager =
    (LocationManager) getSystemService(Context.LOCATION_SERVICE);
locationManager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER, 2000, 10, locListener);
printLocation(locationManager.getLastKnownLocation(
    LocationManager.GPS_PROVIDER));
}

private void printLocation(Location location) {
    if (location != null)
    {
        double lat = location.getLatitude();
        double lon = location.getLongitude();

        // Создаем экземпляр GeoPoint
        GeoPoint point = new GeoPoint(
            (int)(lat * 1E6), (int)(lon * 1E6));

        // Перемещаем карту в заданное местоположение
        controller.animateTo(point);

        // Отображаем координаты местоположения на экране
        text.setText(
            String.format("Lat: %4.2f, Long: %4.2f", lat, lon));

        try {
            // Используем Geocoder для получения информации
            // о местоположении и выводим ее на экран
            Geocoder geocoder = new Geocoder(this, Locale.getDefault());
            List<Address> addresses =
                geocoder.getFromLocation(lat, lon, 1);

            if (addresses.size() > 0) {
                Address address = addresses.get(0);

                for(int i=0; i<address.getMaxAddressLineIndex(); i++) {
                    text.append(", " + address.getAddressLine(i));
                }
                text.append(", " + address.getCountryName());
            }
        }
    }
}

```

```

        catch (IOException e) {
            Toast.makeText(getApplicationContext(), e.toString(),
                Toast.LENGTH_LONG).show();
        }
        catch (Exception e) {
            Toast.makeText(getApplicationContext(), e.toString(),
                Toast.LENGTH_LONG).show();
        }
    }
    else {
        text.setText("Unable get location");
    }
}

@Override
protected boolean isRouteDisplayed() {
    return false;
}
}

```

Скомпилируйте проект и запустите его на эмуляторе Android. Откройте в IDE Eclipse представление **Emulator Control** и задайте координаты в текстовых полях, например **Longitude: 30.31** и **Latitude: 59.94**. Наше приложение должно отобразить карту Санкт-Петербурга в районе Эрмитажа, а в верхней части окна будут указаны координаты и адрес данного местоположения.

Внешний вид приложения представлен на рис. 38.4.

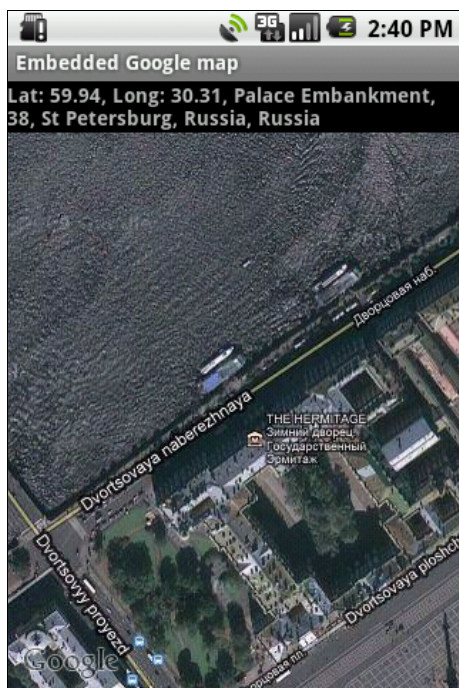


Рис. 38.4. Использование MapView в приложении

Управление масштабированием карты

Теперь усовершенствуем наше приложение, добавив возможность управления масштабом. Как уже говорилось ранее в этой главе, виджет `MapView` содержит встроенный элемент управления масштабированием. Для того чтобы его использовать, необходимо вызвать метод `setBuiltInZoomControls()`, передав ему параметр `true`. Для управления видимостью данного элемента существует еще один метод `displayZoomControls()`.

Давайте используем элемент управления масштабированием в нашем приложении. Для этого в код класса `MapViewActivity` из листинга 38.3 добавим вызовы этих методов. Фрагмент метода `onCreate()` с внесенными добавлениями для управления масштабом карты представлен в листинге 38.4.

Листинг 38.4. Дополнения в классе `MapViewActivity`

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
    map.setSatellite(true);
    map.setStreetView(true);

    // Добавляем элемент управления масштабированием карты
    map.setBuiltInZoomControls(true);
    map.displayZoomControls(true);
    ...
}
```

Теперь наше приложение имеет встроенный элемент управления для изменения масштаба на карте, как представлено на рис. 38.5.

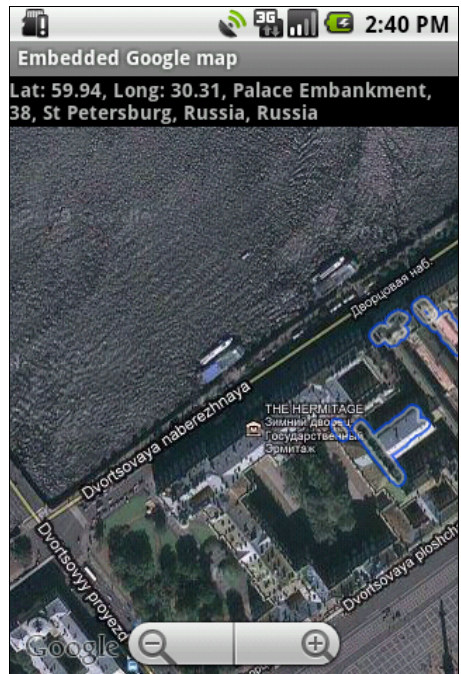


Рис. 38.5. Встроенный элемент управления масштабированием на карте

Добавление маркера

Когда вы работаете с сервисом Google Maps в интернет-браузере, вы видите, что сервис пометает заданную точку специальным маркером. Такой маркер можно встраивать и в приложения, использующие Google Maps. Однако, кроме использования стандартных маркеров, можно создать свое собственное изображение и применять его на карте вместо стандартного маркера.

Сейчас мы создадим приложение, в котором на карте будет отображаться собственный маркер. Вы можете использовать уже созданное ранее приложение, добавив в него новый код, или, если хотите, создайте новый проект.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch38_MapView_Overlay.

Для маркера понадобится иконка в формате PNG, которую надо будет разместить в каталоге `res/drawable` проекта. Если вы не хотите делать свою иконку, можете взять готовую в каталоге проекта `res/drawable/star.png`.

Для отображения маркера на карте сначала необходимо загрузить внешний ресурс, а затем внедрить его в объект `MapView`. Изменения, которые необходимо внести в метод `printLocation()` из листинга 38.3, чтобы устанавливать маркер на заданной точке, представлены в листинге 38.5.

Листинг 38.5. Изменения в методе `printLocation()` класса `MapViewActivity`

```
private void printLocation(Location location) {
    if (location != null)
    {
        double lat = location.getLatitude();
        double lon = location.getLongitude();

        GeoPoint point = new GeoPoint(
            (int)(lat * 1E6), (int)(lon * 1E6));
        controller.animateTo(point);

        // Добавляем маркер, используя внешний ресурс
        ImageView marker = new ImageView(getApplicationContext());
        marker.setImageResource(R.drawable.star);

        MapView.LayoutParams markerParams = new MapView.LayoutParams(
            LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT,
            point, MapView.LayoutParams.TOP_LEFT );
        map.addView(marker, markerParams);

        text.setText(String.format("Lat: %4.2f, Long: %4.2f", lat, lon));
        ...
    }
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Теперь на точке с заданными координатами местоположения будет отображаться маркер. Внешний вид приложения с маркером в виде звездочки представлен на рис. 38.6.

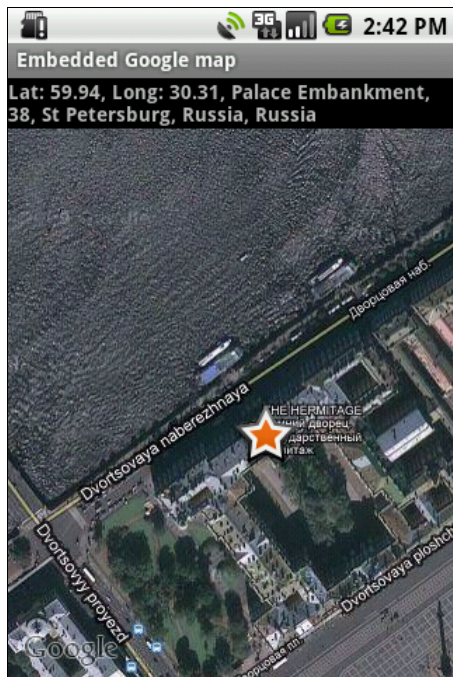


Рис. 38.6. Маркер на карте, подгружаемый из внешнего ресурса

Изменение масштаба карты с помощью виджета *SeekBar*

Для управления масштабом карты, если требуется более плавное его изменение, вместо встроенного элемента управления *ZoomControls* удобнее использовать ползунок, похожий на тот, который отображается в левом верхнем углу на карте Google Map в интернет-браузере. В приложении Android для этого можно использовать стандартный виджет *SeekBar*.

Давайте создадим такое приложение. Откройте в IDE Eclipse новый проект Android и заполните поля в диалоговом окне **New Android Project** следующими значениями:

- ☐ **Project name** — `MapView_Zoom`;
- ☐ **Application name** — `Embedded Google map`;
- ☐ **Package name** — `com.samples.location.mapzoom`;
- ☐ **Create Activity** — `MapViewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch38_MapView_Zoom`.

В файл компоновки главного окна приложения `main.xml` помимо виджетов, которые мы использовали в предыдущих приложениях, добавьте виджет *SeekBar* и элементы *TextView* для отображения надписи и значения текущего масштаба карты.

Код файла компоновки окна приложения представлен в листинге 38.6.

Листинг 38.6. Файл компоновки окна приложения main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frame"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"/>

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <TextView
            android:text="Zoom "
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textStyle="bold"
            android:layout_gravity="center"/>

        <TextView
            android:id="@+id/zoom"
            android:text="0.0"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textStyle="bold"
            android:layout_gravity="center"/>

        <SeekBar
            android:id="@+id/seekbar"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:progress="0"/>

    </LinearLayout>

    <com.google.android.maps.MapView
        android:id="@+id/map"
        android:apiKey="YOUR_API_KEY"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"/>

</LinearLayout>
```

В коде класса главного окна приложения используем в качестве основы класс `MapViewActivity` из предыдущего примера, внося в него дополнительные изменения. Для ползунка нам потребуется создать обработчик события перемещения ползунка `OnSeekBarChangeListener()`, в котором будет необходимо переопределить метод `onProgressChanged()`:

```
SeekBar.OnSeekBarChangeListener seekBarListener =
    new SeekBar.OnSeekBarChangeListener() {
        @Override
        public void onProgressChanged(SearchBar arg0, int arg1, boolean arg2) {
            ...
        }
    }
}
```

В теле метода `onProgressChanged()` можно будет использовать уже знакомый нам метод установки масштабирования `setZoom()` класса `MapController`.

Полный код класса `MapViewActivity` представлен в листинге 38.7.

Листинг 38.7. Файл класса окна приложения `MapViewActivity`

```
package com.samples.location.mapzoom;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.MapView.LayoutParams;

import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

public class MapViewActivity extends MapActivity {
    private static final int ZOOM_MAX = 21;
    private static final int ZOOM_INIT = 17;

    private MapController controller;
    private TextView text;
```

```
private TextView zoom;
private MapView map;
private SeekBar seek;

private final LocationListener locListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        printLocation(location);
    }

    @Override
    public void onProviderDisabled(String arg0) {
        printLocation(null);
    }

    @Override
    public void onProviderEnabled(String arg0) { }

    @Override
    public void onStatusChanged(String arg0, int arg1, Bundle arg2) { }
};

private SeekBar.OnSeekBarChangeListener seekBarListener =
    new SeekBar.OnSeekBarChangeListener() {

        @Override
        public void onProgressChanged(
            SeekBar arg0, int arg1, boolean arg2) {

            // Получаем текущее положение ползунка
            int myZoomLevel = seek.getProgress() + 1;

            // Устанавливаем новое значение масштаба
            controller.setZoom(myZoomLevel);

            // Отображаем новое значение масштаба в текстовом поле
            zoom.setText(String.valueOf(seek.getProgress() + 1));
        }

        @Override
        public void onStartTrackingTouch(SeekBar arg0) { }

        @Override
        public void onStopTrackingTouch(SeekBar arg0) { }
    };

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

```
map = (MapView) findViewById(R.id.map);
text = (TextView) findViewById(R.id.text);
zoom = (TextView) findViewById(R.id.zoom);
seek = (SeekBar) findViewById(R.id.seekbar);

seek.setMax(ZOOM_MAX);
seek.setProgress(ZOOM_INIT);
zoom.setText(String.valueOf(ZOOM_INIT));

controller = map.getController();

// Устанавливаем масштаб карты
controller.setZoom(ZOOM_INIT);

map.setSatellite(true);

// Добавляем элемент управления масштабированием карты
map.displayZoomControls(false);
map.setBuiltInZoomControls(true);

LocationManager locationManager =
    (LocationManager) getSystemService(Context.LOCATION_SERVICE);

locationManager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER, 2000, 10, locListener);
printLocation(locationManager.getLastKnownLocation(
    LocationManager.GPS_PROVIDER));

// Подключаем обработчик события перемещения для ползунка
seek.setOnSeekBarChangeListener(seekBarListener);
}

@Override
protected boolean isRouteDisplayed() {
    return false;
}

private void printLocation(Location location) {
    if (location != null)
    {
        double lat = location.getLatitude();
        double lon = location.getLongitude();

        // Создаем экземпляр GeoPoint
        GeoPoint point = new GeoPoint(
            (int) (lat * 1E6), (int) (lon * 1E6));

        // Перемещаем карту в заданное местоположение
        controller.animateTo(point);
    }
}
```

```
ImageView marker = new ImageView(getApplicationContext());
marker.setImageResource(R.drawable.star);
MapView.LayoutParams markerParams = new MapView.LayoutParams(
    LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT,
    point, MapView.LayoutParams.TOP_LEFT );

map.addView(marker, markerParams);

// Отображаем координаты местоположения на экране
text.setText(
    String.format("Lat: %4.2f, Long: %4.2f", lat, lon));

try {
    // Используем Geocoder для получения информации
    // о местоположении и выводим ее на экран
    Geocoder geocoder = new Geocoder(this, Locale.getDefault());
    List<Address> addresses =
        geocoder.getFromLocation(lat, lon, 1);

    if (addresses.size() > 0) {
        Address address = addresses.get(0);

        for(int i=0; i<address.getMaxAddressLineIndex(); i++) {
            text.append(", " + address.getAddressLine(i));
        }

        text.append(", " + address.getCountryName());
    }
}
catch (IOException e) {
    Toast.makeText(getApplicationContext(), e.toString(),
        Toast.LENGTH_LONG).show();
}
catch (Exception e) {
    Toast.makeText(getApplicationContext(), e.toString(),
        Toast.LENGTH_LONG).show();
}
}
else {
    text.setText("Unable get location");
}
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Внешний вид нашего приложения с добавленным ползунком для плавного регулирования масштаба представлен на рис. 38.7.

Используя вместо встроенного элемента управления масштабированием виджет SeekBar, мы можем плавно изменять масштаб карты, однако ползунок занимает доволь-

но большую площадь на экране мобильного устройства, поэтому, как правило, для приложений с использованием карт применяют встроенный элемент масштабирования.

Кстати, ползунок и надпись можно наложить на карту — поместить их на поверхности изображения карты так же, как и маркер. Сейчас мы это делать не будем, но в *главе 39*, когда будем изучать работу со встроенной видеокамерой мобильного устройства, мы рассмотрим создание собственных оверлеев в окне приложения.

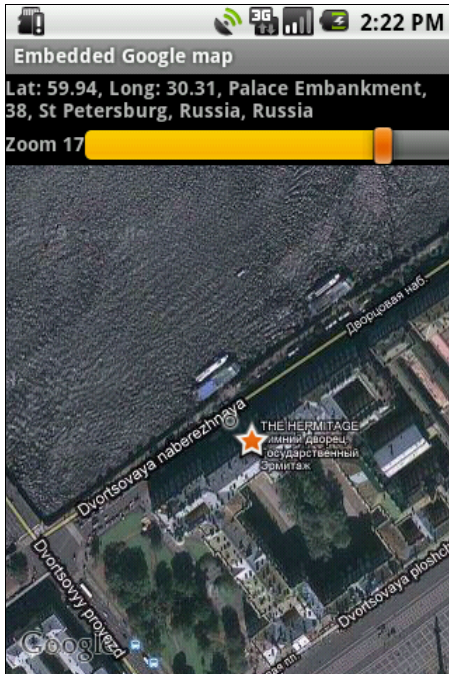
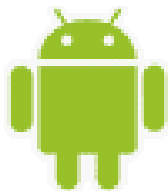


Рис. 38.7. Использование виджета SeekBar для плавного регулирования масштаба карты

Резюме

Применение встроенных карт Google Maps в приложениях открывает широкие возможности при создании различных приложений для мобильных устройств, используемых в области навигации. Как вы могли убедиться, такие приложения создавать очень легко, т. к. библиотеки Android SDK и Google API предлагают очень удобные компоненты, которые предоставляют хорошую базовую функциональность для использования в ваших приложениях.

Этой главой мы заканчиваем рассмотрение сетевых сервисов и в следующей части книги переходим к еще одной интересной теме — работе с "железом" устройства Android.



ЧАСТЬ VI

Работа с оборудованием

- Глава 39.** Использование видеокамеры
- Глава 40.** Встроенные датчики
- Глава 41.** Управление дисплеем
- Глава 42.** Доступ к аккумуляторной батарее
- Глава 43.** Управление энергопотреблением телефона
- Глава 44.** Получение информации о системе



ГЛАВА 39

Использование видеокамеры

В настоящее время практически любой телефон оснащен встроенной видеокамерой. Камера — это наиболее часто используемый датчик в мобильном телефоне на платформе Android. Поэтому создание приложений, использующих возможности встроенной камеры, является очень популярным направлением для разработчиков мобильных приложений под платформу Android.

В этой главе будет рассказано о работе со встроенной камерой мобильного телефона Android, о доступе и управлении камерой из приложений.

Работа с камерой в приложении

Для открытия камеры в коде приложения и получения доступа к настройкам, параметрам и режимам работы камеры используется класс `Camera`. Этот класс находится в библиотеке `android.hardware`, которая содержит набор классов для доступа к оборудованию мобильного устройства, в том числе к видеокамере и к встроенным датчикам (их мы будем рассматривать в следующей главе).

Класс `Camera` является клиентом для системной службы Android, которая непосредственно осуществляет управление встроенной видеокамерой как оборудованием мобильного устройства.

Для управления встроенной камерой из кода приложения в классе `Camera` определена следующая группа методов:

- ❑ `open()` — создает новый объект `Camera` для программного доступа к видеокамере;
- ❑ `open(int cameraId)` — создает новый объект `Camera` для программного доступа к видеокамере с заданным во входном параметре идентификатором этой камеры (возможен вариант, когда мобильное устройство имеет более одной видеокамеры);
- ❑ `release()` — освобождает системный ресурс `Camera`.

Для работы с камерой в приложении необходимо сначала создать объект `Camera`, используя вызов статического метода `open()` этого класса. Применение объекта `Camera` в приложении не является потокобезопасным, поэтому его можно использовать только в одном потоке. После окончания использования камеры обязательно требуется освободить ресурс, вызвав метод `release()`, чтобы камерой могли воспользоваться другие приложения, установленные на этом мобильном устройстве.

Например, при работе с камерой в коде можно использовать следующее:

```
// Открываем камеру
Camera camera = Camera.open();
// Читаем параметры камеры
Camera.Parameters params = camera.getParameters();
...
// Закрываем камеру и освобождаем ресурсы
camera.release();
```

Для доступа к камере из кода приложения также обязательно требуется разрешение `android.permission.CAMERA`, которое необходимо поместить в файле манифеста приложения.

Параметры камеры

Класс `Camera` для доступа к параметрам камеры имеет пару методов: `getParameters()`, который возвращает текущие параметры видеокamеры, и `setParameters()`, с помощью которого можно изменять настройки этих параметров. Параметры камеры представлены классом `Camera.Parameters`.

Класс `Camera.Parameters` имеет большое количество методов для чтения и установки требуемых характеристик видеокamеры, например, для фокусировки, масштабирования изображения, установки баланса белого, изменения ориентации камеры и др. Кроме того, в этом классе также определены константы, представляющие значения для различных параметров камеры. Класс `Camera.Parameters` и его функциональность мы будем рассматривать на протяжении всей этой главы, используя его в наших приложениях.

Получение параметров камеры в приложении

Для начала создадим простое приложение, открывающее встроенную камеру мобильного устройства и читающее базовые характеристики этой камеры. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `CameraInfo`;
- ☐ **Application name** — `Camera information`;
- ☐ **Package name** — `com.samples.camera.info`;
- ☐ **Create Activity** — `CameraInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch39_CameraInfo`.

Для доступа к камере мобильного устройства из приложения нам понадобится разрешение `android.permission.CAMERA`, и оно должно использоваться во всех примерах этой главы. В файл манифеста приложения `AndroidManifest.xml` обязательно нужно добавить это разрешение, как показано в листинге 39.1.

Листинг 39.1. Файл манифеста приложения AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.camera.info"
    android:versionCode="1"
    android:versionName="1.0">

    <application
        android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".CameraInfoActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.CAMERA" />
</manifest>

```

В файле компоновки окна приложения `main.xml` будет только текстовое поле `TextView` с идентификатором `text` для вывода параметров камеры.

В классе `CameraInfoActivity` в обработчике события `onCreate()` будет создаваться объект `Camera`, и с помощью его метода `getParameters()` мы можем получить доступ к параметрам встроенной видеокамеры. Код класса `CameraInfoActivity` представлен в листинге 39.2.

Листинг 39.2. Файл класса окна приложения CameraInfoActivity.java

```

package com.samples.camera.info;

import android.app.Activity;
import android.hardware.Camera;
import android.os.Bundle;
import android.widget.TextView;

public class CameraInfoActivity extends Activity {

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

        TextView text = (TextView)findViewById(R.id.text);
        Camera camera = Camera.open();
        Camera.Parameters params = camera.getParameters();
    }
}

```

```

text.append("Antibanding:\t" + params.getAntibanding());
text.append("\nColor effect:\t" + params.getColorEffect());
text.append("\nFlash mode:\t" + params.getFlashMode());
text.append("\nFocus mode:\t" + params.getFocusMode());
text.append("\nPicture format:\t" + params.getPictureFormat());
text.append("\nPreview format:\t" + params.getPreviewFormat());
text.append("\nPreview frame rate:\t" +
    params.getPreviewFrameRate());
text.append("\nScene mode:\t" + params.getSceneMode());
text.append("\nWhite balance:\t" + params.getWhiteBalance());

camera.release();
}
}

```

Скомпилируйте проект и запустите его на мобильном устройстве. Приложение откроет камеру и выведет характеристики встроенной видеокамеры вашего мобильного устройства. Внешний вид приложения представлен на рис. 39.1.

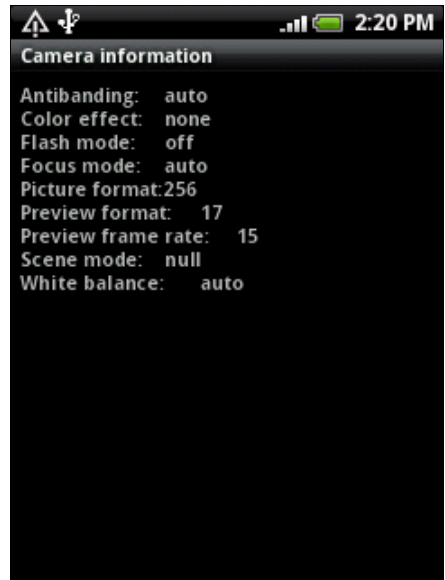


Рис. 39.1. Вывод информации о камере

Поддержка различных режимов камерой

Когда вы создаете приложение для работы с камерой мобильного устройства Android, необходимо учитывать огромное разнообразие существующих мобильных телефонов с системой Android, выпускаемых разными производителями и, как следствие, большое разнообразие встроенных в эти телефоны видеокамер, имеющих большой разброс характеристик. В случае предъявления каких-либо специфических требований к встроенной камере приложение должно проверять камеру на соответствие этим требованиям перед ее использованием.

Для этих целей в классе `Camera.Parameters` предусмотрен большой набор методов для получения полной картины о поддерживаемых режимах работы камеры:

- ❑ `getSupportedColorEffects()` — возвращает список поддерживаемых цветовых эффектов;

- ❑ `getSupportedFlashModes()` — возвращает список поддерживаемых режимов встроенной вспышки;
- ❑ `getSupportedFocusModes()` — возвращает список поддерживаемых режимов фокусировки;
- ❑ `getSupportedPictureFormats()` — возвращает список поддерживаемых форматов изображения;
- ❑ `getSupportedPreviewFormats()` — возвращает список поддерживаемых форматов просмотра изображений;
- ❑ `getSupportedPreviewFpsRange()` — возвращает список поддерживаемых диапазонов скорости съемки, измеряемой в количестве кадров в секунду. Этот метод возвращает список целочисленных массивов, каждый из которых содержит два элемента — минимальную и максимальную скорость съемки;
- ❑ `getSupportedSceneModes()` — возвращает список поддерживаемых режимов сцены (например, для различной освещенности, скорости перемещения объектов съемки и др.);
- ❑ `getSupportedWhiteBalance()` — возвращает список поддерживаемых режимов баланса белого.

Важными характеристиками камеры являются также размер и формат изображений, с которыми может работать встроенная камера. Для этих целей в классе `Camera.Parameters` предусмотрен набор методов для оценки размеров изображений, поддерживаемых камерой:

- ❑ `getSupportedPictureSizes()` — возвращает список поддерживаемых форматов изображений;
- ❑ `getSupportedJpegThumbnailSizes()` — возвращает список поддерживаемых форматов для изображений JPEG;
- ❑ `getSupportedPreviewSizes()` — возвращает список поддерживаемых форматов изображений для предварительного просмотра;
- ❑ `getSupportedVideoSizes()` — возвращает список поддерживаемых размеров экрана.

А теперь рассмотрим получение приложением информации о режимах работы реальной видеокамеры на конкретном мобильном устройстве. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `CameraSupportedModesInfo`;
- ❑ **Application name** — `Camera Supported Modes`;
- ❑ **Package name** — `com.samples.camera.supportedmodes`;
- ❑ **Create Activity** — `CameraInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch39_CameraSupportedModesInfo`.

В коде класса `CameraInfoActivity`, представляющего главное окно приложения, в методе `onCreate()`, так же как и в предыдущем примере, мы используем метод `getParameters()`

для получения объекта класса `Camera.Parameters` и чтения режимов, поддерживаемых данной камерой.

Код класса `CameraInfoActivity` представлен в листинге 39.3.

Листинг 39.3. Файл класса окна приложения `CameraInfoActivity.java`

```
package com.samples.camera.supportedmodes;

import java.util.List;

import android.app.Activity;
import android.hardware.Camera;
import android.hardware.Camera.Size;
import android.os.Bundle;
import android.widget.TextView;

public class CameraInfoActivity extends Activity {

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

        TextView text = (TextView)findViewById(R.id.text);
        Camera camera = Camera.open();
        Camera.Parameters params = camera.getParameters();

        List<Camera.Size> sizes = params.getSupportedPictureSizes();
        if (sizes != null) {
            text.append("Supported Picture Sizes:\n");
            for (Camera.Size size : sizes) {
                text.append(String.format("%dx%d; ",
                    size.height, size.width));
            }
        }
        else {
            text.append("\nNo Supported Picture Sizes");
        }

        List<String> flashModes = params.getSupportedFlashModes();
        if (flashModes != null) {
            text.append("\n\nSupported Flash modes :\n");
            for (String mode : flashModes) {
                text.append(String.format("%s; ", mode));
            }
        }
        else {
            text.append("\nNo Supported Flash Modes");
        }

        List<String> antibandings = params.getSupportedAntibanding();
        if (antibandings != null) {
            text.append("\n\nSupported Antibanding :\n");
        }
    }
}
```

```
        for (String mode : antibandings) {
            text.append(String.format("%s; ", mode));
        }
    }
    else {
        text.append("\nNo Supported Antibanding");
    }

    List<String> colorEffects = params.getSupportedColorEffects();
    if (colorEffects != null) {
        text.append("\n\nSupported color effects \n");
        for (String mode : colorEffects) {
            text.append(String.format("%s; ", mode));
        }
    }
    else {
        text.append("\nNo Supported Color Effects");
    }

    List<String> focusModes = params.getSupportedFocusModes();
    if (focusModes != null) {
        text.append("\n\nSupported Focus Modes \n");
        for (String mode : focusModes) {
            text.append(String.format("%s; ", mode));
        }
    }
    else {
        text.append("\nNo Supported Focus Modes");
    }

    List<Size> previewSizes = params.getSupportedPreviewSizes();
    if (previewSizes != null) {
        text.append("\n\nSupported Preview Sizes \n");
        for (Size mode : previewSizes) {
            text.append(String.format("%dx%d; ",
                                     mode.height, mode.width));
        }
    }
    else {
        text.append("\nNo Supported Preview Sizes");
    }

    List<String> sceneModes = params.getSupportedSceneModes();
    if (sceneModes != null) {
        text.append("\n\nSupported SceneModes \n");
        for (String mode : sceneModes) {
            text.append(String.format("%s; ", mode));
        }
    }
    else {
        text.append("\nNo Supported SceneModes");
    }
}
```

```

List<String> whiteBalances = params.getSupportedWhiteBalance();
if (whiteBalances != null) {
    text.append("\nSupported White Balance \n");
    for (String mode : whiteBalances) {
        text.append(String.format("%s; ", mode));
    }
}
else {
    text.append("\nNo Supported White Balance");
}

camera.release();
}
}

```

Скомпилируйте проект и запустите его на мобильном устройстве. Внешний вид приложения и выводимая им информация о режимах, поддерживаемых камерой, представлены на рис. 39.2.

Конечно приложение, запущенное на вашем мобильном устройстве, будет выводить другие значения режимов работы. Поэтому при использовании встроенной камеры в серийных приложениях, если к камере применяются особенные требования, например, разрешение, размер изображения, диапазон изменения масштаба и другие характеристики, требуемые для корректной работы данного приложения, необходимо вначале проверить возможности видеокамеры в мобильном устройстве, на которое установлено данное приложение.

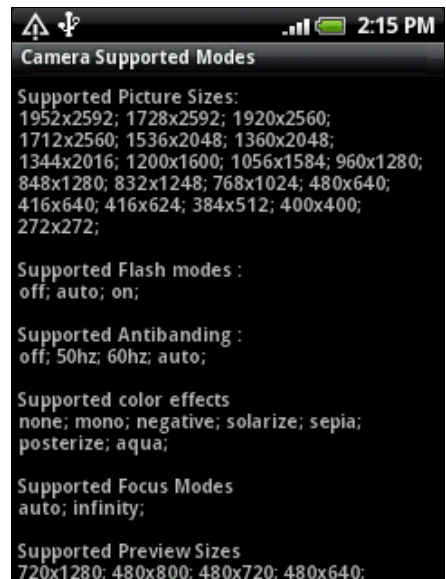


Рис. 39.2. Получение информации о режимах, поддерживаемых камерой

Использование объектов *Intent* для открытия камеры

Для работы с камерой существует несколько констант, определяющих объекты *Intent* для открытия стандартных *Activity*, использующих встроенную камеру. Эти константы определены в классе *MediaStore* пакета *android.provider*.

По своему назначению их можно разделить на две группы: для открытия и запуска видеокамеры и для захвата изображений. Для создания объектов `Intent` для открытия камеры в видеорежиме или в режиме фотографирования используются следующие значения:

- ❑ `INTENT_ACTION_STILL_IMAGE_CAMERA` — имя объекта `Intent` для открытия камеры в режиме фотоаппарата;
- ❑ `INTENT_ACTION_VIDEO_CAMERA` — имя объекта `Intent` для открытия камеры в видеорежиме.

Для захвата камерой видео или фотографий используются такие константы:

- ❑ `ACTION_IMAGE_CAPTURE` — стандартный `Intent` для действия по захвату камерой изображения и получения его в приложении;
- ❑ `ACTION_VIDEO_CAPTURE` — стандартный `Intent` для действия по захвату камерой видео и получения его в приложении.

Таким образом, можно запустить стандартный `Activity`, объявляя неявный объект `Intent` с параметром:

```
Intent intent = new Intent("android.media.action.ACTION_IMAGE_CAPTURE ");
startActivity(intent);
```

Сейчас мы попробуем такой способ вызова камеры в приложении. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `CameraCallActions`;
- ❑ **Application name** — `Call Camera Activities`;
- ❑ **Package name** — `com.samples.camera.callactivities`;
- ❑ **Create Activity** — `CallCameraActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch39_CameraCallActions`.

В файле компоновки главного окна приложения `main.xml` будут четыре кнопки для вызова всех четырех вариантов стандартного `Activity` с камерой. Файл компоновки окна представлен в листинге 39.4.

Листинг 39.4. Файл компоновки окна приложения `main.xml`

```
<?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">

    <Button
        android:id="@+id/bImageCapture"
        android:text="Image Capture"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"/>
```

```

<Button
    android:id="@+id/bVideoCapture"
    android:text="Video Capture"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"/>
<Button
    android:id="@+id/bStillImage"
    android:text="Still image mode"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"/>
<Button
    android:id="@+id/bVideoCamera"
    android:text="Video Camera"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"/>

</LinearLayout>

```

В классе `CallCameraActivity`, представляющем главное окно приложения, в обработчике события `onClick()` командных кнопок реализуем все четыре режима запуска встроенной видекамеры. Код класса `CallCameraActivity` представлен в листинге 39.5.

Листинг 39.5. Файл класса окна приложения `CallCameraActivity.java`

```

package com.samples.camera.callactivities;

import android.app.Activity;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.provider.MediaStore;
import android.view.View;
import android.widget.Button;

public class CallCameraActivity extends Activity
    implements View.OnClickListener {

    private Button bImageCapture;
    private Button bVideoCapture;
    private Button bStillImage;
    private Button bVideoCamera;

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

        bImageCapture = (Button) findViewById(R.id.bImageCapture);
        bVideoCapture = (Button) findViewById(R.id.bVideoCapture);
    }
}

```

```

bStillImage = (Button)findViewById(R.id.bStillImage);
bVideoCamera = (Button)findViewById(R.id.bVideoCamera);

bImageCapture.setOnClickListener(this);
bVideoCapture.setOnClickListener(this);
bStillImage.setOnClickListener(this);
bVideoCamera.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bImageCapture:
            startActivity(new Intent(MediaStore.ACTION_IMAGE_CAPTURE));
            break;
        case R.id.bVideoCapture:
            startActivity(new Intent(MediaStore.ACTION_VIDEO_CAPTURE));
            break;
        case R.id.bStillImage:
            startActivity(new Intent(
                MediaStore.INTENT_ACTION_STILL_IMAGE_CAMERA));
            break;
        case R.id.bVideoCamera:
            startActivity(new Intent(
                MediaStore.INTENT_ACTION_VIDEO_CAMERA));
            break;
    }
}
}
}

```

Скомпилируйте проект и запустите его на своем мобильном устройстве. Внешний вид приложения представлен на рис. 39.3.

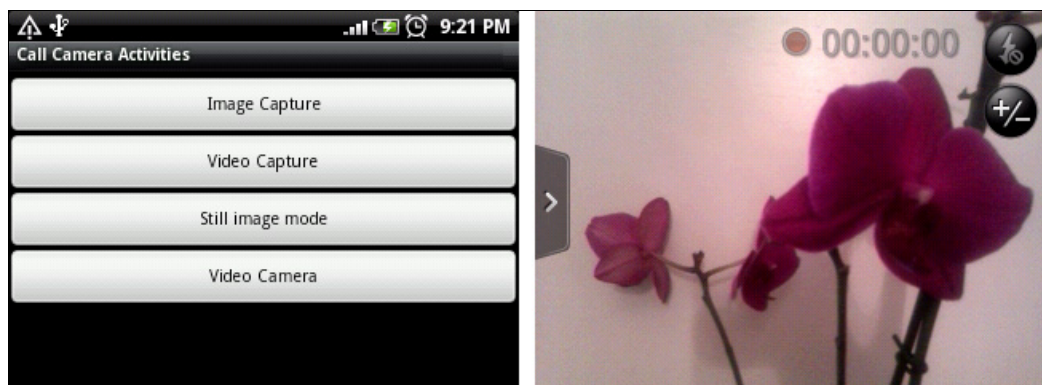


Рис. 39.3. Запуск стандартного Activity с камерой в режиме ACTION_VIDEO_CAPTURE

Встраивание камеры в приложения

В предыдущем разделе мы использовали стандартный Activity для работы с камерой. Камеру также можно использовать непосредственно в приложении, в собственном Activity. Однако, в отличие от работы с картами Google Maps, когда для их отображения мы использовали виджет MapView, в библиотеке Android SDK отсутствует специализированный виджет для видеокamеры.

Выход из положения все же есть — для получения изображений с камеры можно использовать виджет SurfaceView. Этот виджет также может быть использован для размещения дополнительных элементов управления и изображений, например кнопок или видеискателя на поверхности виджета SurfaceView (что также мы рассмотрим несколько позже в этой главе).

Доступ к поверхности осуществляется через интерфейс SurfaceHolder, который можно получить с помощью метода `getHolder()`. Поверхность с изображением будет показываться лишь до тех пор, пока окно SurfaceView является видимым. Поэтому в коде программы нужно поймать события, когда поверхность создается, изменяется или разрушается. Для этого надо реализовать три метода интерфейса SurfaceHolder:

- ❑ `surfaceCreated()` — вызывается при создании поверхности;
- ❑ `surfaceChanged()` — вызывается при изменении изображения (или при перекрытии поверхности другим Activity);
- ❑ `surfaceDestroyed()` — вызывается после закрытия при освобождении ресурсов.

В теле этих методов надо написать необходимые действия при наступлении этих событий:

```
Camera camera;
...
@Override
public void surfaceCreated(SurfaceHolder holder) {
    // Открываем камеру
    camera = Camera.open();

    // Отображаем камеру на поверхности окна
    camera.setPreviewDisplay(holder);

    // Запускаем камеру
    camera.startPreview();
}

@Override
public void surfaceChanged(
    SurfaceHolder holder, int format, int width, int height) {
    // Действия при изменении изображения
}

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
```

```
// Закрытие камеры и освобождение ресурсов
camera.stopPreview();
camera.release();
}
```

Реализуем теперь вывод изображений с камеры в реальном приложении. Для этого создайте в IDE Eclipse новый проект Android и заполните следующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — CameraPreview;
- ☐ **Application name** — Camera preview;
- ☐ **Package name** — com.samples.camera.preview;
- ☐ **Create Activity** — CameraPreviewActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch39_CameraPreview.

В файле компоновки главного окна приложения `main.xml` будет находиться виджет `SurfaceView` с идентификатором `surfaceview`, как показано в листинге 39.6.

Листинг 39.6. Файл компоновки окна приложения `main.xml`

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

    <SurfaceView
        android:id="@+id/surfaceview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

</LinearLayout>
```

В классе `CameraPreviewActivity` главного окна нам необходимо реализовать методы интерфейса `SurfaceHolder.Callback`, как было показано ранее в этом разделе.

Код класса `CameraPreviewActivity` представлен в листинге 39.7.

Листинг 39.7. Файл класса окна приложения `CameraPreviewActivity.java`

```
package com.samplpes.camera.preview;

import android.app.Activity;
import android.hardware.Camera;
import android.os.Bundle;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.widget.Toast;
```

```
public class CameraPreviewActivity extends Activity
    implements SurfaceHolder.Callback {

    private SurfaceView surView;
    private SurfaceHolder surHolder;

    private Camera camera;

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

        surView = (SurfaceView)findViewById(R.id.surfaceview);

        surHolder = surView.getHolder();
        surHolder.addCallback(this);
        surHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        try {
            // Открываем камеру
            camera = Camera.open();
            camera.setPreviewDisplay(surHolder);

            // Запускаем просмотр
            camera.startPreview();
        }
        catch (Exception e) {
            Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void surfaceChanged(
        SurfaceHolder holder, int format, int width, int height) { }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        // Закрываем камеру и освобождаем ресурсы
        camera.stopPreview();
        camera.release();
    }
}
```

Скомпилируйте проект и разверните его на мобильном устройстве. При запуске приложения камера будет сразу запущена для просмотра изображений, а при закрытии камеры остановится и освободит ресурсы для других приложений. Внешний вид приложения представлен на рис. 39.4.

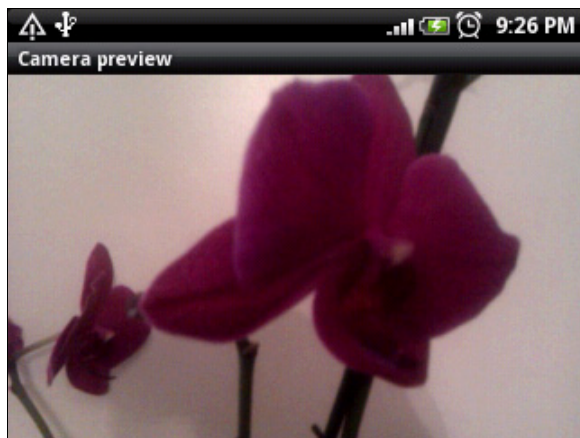


Рис. 39.4. Использование камеры в приложении

Управление работой камеры

В предыдущем примере мы только открывали камеру. Теперь рассмотрим возможности управления режимами работы камеры в приложении с помощью дополнительных элементов управления. Сделать это очень легко — надо просто вызвать методы для открытия и закрытия камеры в обработчиках событий соответствующих кнопок.

Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — CameraManagePreview;
- ☐ **Application name** — Camera Preview;
- ☐ **Package name** — com.samples.camera.managepreview;
- ☐ **Create Activity** — CameraPreviewActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch39_CameraManagePreview.

В файл компоновки главного окна приложения `main.xml`, помимо элемента `SurfaceView`, добавим две кнопки для управления камерой **Start** и **Stop** с идентификаторами `bStart` и `bStop`.

Код файла компоновки `main.xml` представлен в листинге 39.8.

Листинг 39.8. Файл компоновки окна приложения `main.xml`

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

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_margin="3px">

    <Button
        android:layout_height="wrap_content"
        android:id="@+id/bStart"
        android:text="Start"
        android:layout_width="wrap_content"
        android:layout_weight="1"/>

    <Button
        android:layout_height="wrap_content"
        android:id="@+id/bStop"
        android:text="Stop"
        android:layout_width="wrap_content"
        android:layout_weight="1"/>

</LinearLayout>

<SurfaceView
    android:id="@+id/surfaceview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>

</LinearLayout>
```

В коде класса `CameraPreviewActivity` нам понадобятся обработчики событий `onClick()` для кнопок, с помощью которых мы будем производить запуск и останов камеры. В классе нам также понадобится дополнительная переменная `isCameraPreview` типа `boolean`, которая будет определять состояние камеры. Эта переменная нам потребуется для отслеживания состояния камеры при нажатии пользователем кнопок запуска и остановки камеры и при выходе из приложения, т. е. в методе `onClick()` для кнопок, а также в методе `surfaceDestroyed()`. Дело в том, что если камера уже открыта, повторная попытка открытия камеры методом `open()` вызовет исключение. То же произойдет и при попытке повторного закрытия камеры. Поэтому переменная `isCameraPreview` нужна при блокировании кнопок для защиты от повторного нажатия и в теле метода `surfaceDestroyed()` для проверки состояния камеры при выходе из приложения.

Код класса главного окна приложения представлен в листинге 39.9.

Листинг 39.9. Файл класса окна приложения `CameraPreviewActivity.java`

```
package com.samples.camera.managepreview;

import java.io.IOException;

import android.app.Activity;
import android.hardware.Camera;
import android.os.Bundle;
```



```
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class CameraPreviewActivity extends Activity
    implements SurfaceHolder.Callback, View.OnClickListener {

    private Button bStart;
    private Button bStop;
    private SurfaceView surView;
    private SurfaceHolder surHolder;

    private Camera camera;
    boolean isCameraPreview = false;

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

        bStart = (Button)findViewById(R.id.bStart);
        bStop = (Button)findViewById(R.id.bStop);

        surView = (SurfaceView)findViewById(R.id.surfaceview);
        surHolder = surView.getHolder();
        surHolder.addCallback(this);
        surHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

        bStart.setOnClickListener(this);
        bStop.setOnClickListener(this);

        bStop.setEnabled(false);
    }

    @Override
    public void surfaceChanged(
        SurfaceHolder holder, int format, int width, int height) { }

    @Override
    public void surfaceCreated(SurfaceHolder holder) { }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        if (isCameraPreview) {
            camera.stopPreview();
            camera.release();
            isCameraPreview = false;
        }
    }
}
```

```

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bStart:
            try {
                camera = Camera.open();
                camera.setPreviewDisplay(surHolder);
                camera.startPreview();
                isCameraPreview = true;

                bStart.setEnabled(!isCameraPreview);
                bStop.setEnabled(isCameraPreview);
            }
            catch (IOException e) {
                Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
            }
            break;

        case R.id.bStop:
            camera.stopPreview();
            camera.release();
            isCameraPreview = false;

            bStart.setEnabled(!isCameraPreview);
            bStop.setEnabled(isCameraPreview);
            break;
    }
}
}
}

```

Скомпилируйте проект и запустите его на мобильном устройстве. Теперь видеочамера открывается только при нажатии кнопки **Start**. После запуска видеочамеры кнопка **Start** блокируется во избежание повторного нажатия и одновременно становится доступной кнопка **Stop** для возможности останова видеочамеры. Внешний вид приложения представлен на рис. 39.5.

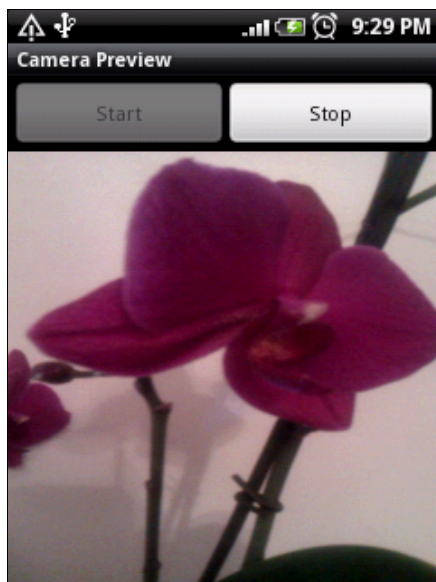


Рис. 39.5. Приложение с управлением камерой

Добавление оверлеев

Помимо показа изображения с камеры, можно накладывать на поверхность изображения различные объекты (оверлеи), например кнопки для управления камерой, рамку видеискателя, а также выводить индикацию времени съемки и другую полезную информацию.

Использование оверлеев позволяет экономить площадь экрана мобильного устройства, т. к. дополнительные элементы управления, как в примере из предыдущего раздела (см. рис. 39.5), занимают место, уменьшая и без того малую площадь экрана мобильного устройства.

Оверлеи хранятся в отдельном XML-файле компоновки. Для создания оверлея в программе используется объект класса `LayoutInflater`, чтобы обработать XML-файл компоновки оверлея и преобразовать его в объект `View`. Это можно сделать, например, следующим образом:

```
LayoutInflater inflater = LayoutInflater.from(getBaseContext());
// Получаем объект View из файла компоновки оверлея (overlay.xml)
View overlay = inflater.inflate(R.layout.overlay, null);
LayoutParams params = new LayoutParams(
    LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
// Добавляем оверлей на поверхность Activity
addContentView(overlay, params);
// Теперь можно получить доступ к кнопкам-оверлеям
// с помощью стандартного вызова метода findViewById()
ImageButton button = (ImageButton)overlay.findViewById(R.id.button1);
```

Теперь усовершенствуем наше приложение для работы с камерой, добавив встроенные элементы управления. Можете использовать в качестве основы предыдущий проект или создать новый. Для нового проекта Android заполните поля в диалоговом окне **New Android Project** следующими значениями:

- ☐ **Project name** — `CameraOverlay`;
- ☐ **Application name** — `Camera Overlay`;
- ☐ **Package name** — `com.samples.camera.overlay`;
- ☐ **Create Activity** — `CameraPreviewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch39_CameraOverlay`.

Код файла `main.xml` остается таким же, как в листинге 39.6. Для оверлеев необходимо будет создать дополнительный файл компоновки, который назовем `overlay.xml`.

В этот файл добавим две кнопки `ImageButton` для запуска и останова работы видеокамеры и присвоим им идентификаторы `bStart` и `bStop`.

ПРИМЕЧАНИЕ

Для изображений на этих кнопках потребуются дополнительные значки, которые вы можете найти на диске в каталоге `Ch39_CameraOverlay/res/drawable`.

Код файла `overlay.xml` представлен в листинге 39.10.

Листинг 39.10. Файл компоновки `overlay.xml` для оверлеев

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="bottom|right">

    <ImageButton
        android:id="@+id/bStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/start"
        android:layout_margin="1px"/>

    <ImageButton
        android:id="@+id/bStop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/stop"
        android:layout_margin="1px"/>

</LinearLayout>
```

В классе `CameraPreviewActivity` приложения реализуем приведенный ранее в этом разделе способ наложения дополнительных изображений на поверхность. Код класса `CameraPreviewActivity` представлен в листинге 39.11.

Листинг 39.11. Файл класса окна приложения `CameraPreviewActivity.java`

```
package com.samples.camera.overlay;

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

import java.io.IOException;

import android.hardware.Camera;
import android.view.LayoutInflater;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.ImageButton;
import android.widget.Toast;

public class CameraPreviewActivity extends Activity
    implements SurfaceHolder.Callback, View.OnClickListener {

    private ImageButton bStart;
    private ImageButton bStop;
```

```
private SurfaceView surView;
private SurfaceHolder surHolder;

private Camera camera;
private boolean isCameraPreview = false;

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

    surView = (SurfaceView)findViewById(R.id.surfaceview);
    surHolder = surView.getHolder();
    surHolder.addCallback(this);

    // Создаем объект LayoutInflater
    LayoutInflater inflater = LayoutInflater.from(getBaseContext());
    View overlay = inflater.inflate(R.layout.overlay, null);
    LayoutParams params = new LayoutParams(
        LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
    // Добавляем оверлей на поверхность Activity
    addContentView(overlay, params);

    bStart = (ImageButton)overlay.findViewById(R.id.bStart);
    bStop = (ImageButton)overlay.findViewById(R.id.bStop);

    bStart.setOnClickListener(this);
    bStop.setOnClickListener(this);

    bStop.setEnabled(false);
}

@Override
public void surfaceChanged(
    SurfaceHolder holder, int format, int width, int height) {
}

@Override
public void surfaceCreated(SurfaceHolder holder) { }

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    if (isCameraPreview) {
        camera.stopPreview();
        camera.release();
        isCameraPreview = false;
    }
}
```

```

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bStart:
            try {
                camera = Camera.open();
                camera.setPreviewDisplay(surHolder);
                camera.startPreview();
                isCameraPreview = true;

                bStart.setEnabled(!isCameraPreview);
                bStop.setEnabled(isCameraPreview);
            }
            catch (IOException e) {
                Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
            }
            break;

        case R.id.bStop:
            camera.stopPreview();
            camera.release();
            isCameraPreview = false;

            bStart.setEnabled(!isCameraPreview);
            bStop.setEnabled(isCameraPreview);
            break;
    }
}
}

```

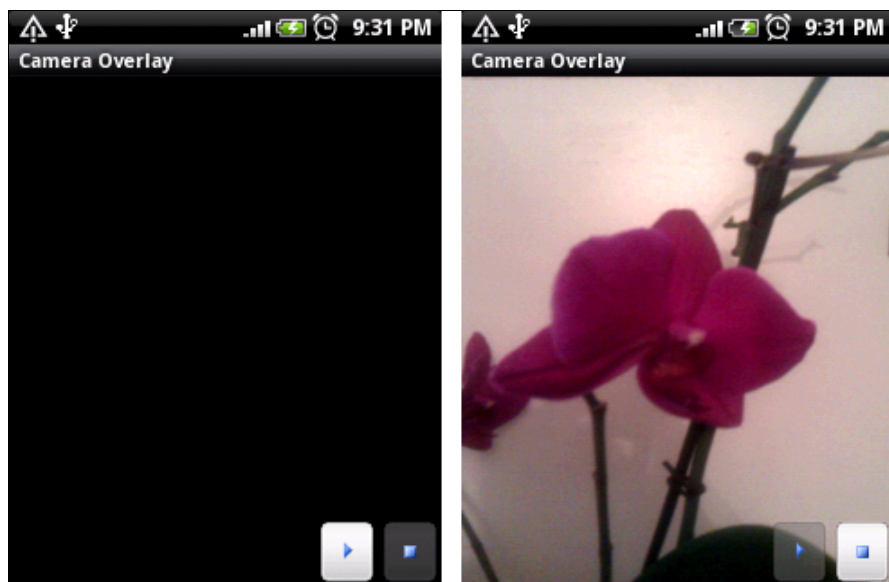


Рис. 39.6. Приложение с оверлеями для управления камерой

Скомпилируйте проект и запустите его на мобильном устройстве. Принцип работы кнопок управления камерой и их блокировка остались такими же, как и в предыдущем проекте, но теперь эти кнопки расположены на поверхности изображения, как показано на рис. 39.6.

Однако при добавлении элементов управления и дополнительных изображений на поверхность виджета `SurfaceView` необходимо учитывать, что это негативно влияет на производительность приложения, поскольку увеличивается время на перерисовку изображения.

Захват изображения

Чтобы сохранить изображение с видеокамеры, необходимо произвести его захват. Для этого в классе `Camera` предусмотрено два метода `takePicture()`, отличающихся набором входных параметров:

- ❑ `takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback postview, Camera.PictureCallback jpeg);`
- ❑ `takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg).`

Рассмотрим входные параметры методов для захвата изображения подробнее. Поскольку метод `takePicture()` является асинхронным, для захвата изображения нам надо передать несколько параметров, являющихся именами методов обратного вызова. Эти методы мы должны реализовать в коде программы, работающей с видеокамерой:

- ❑ `Camera.ShutterCallback shutter` — имя метода обратного вызова, который вызывает-ся в момент захвата изображения камерой;
- ❑ `Camera.PictureCallback raw` — имя метода обратного вызова для передачи в программу данных изображения в несжатом виде;
- ❑ `Camera.PictureCallback jpeg` — имя метода обратного вызова для передачи данных изображения, сжатых в формате JPEG;
- ❑ `Camera.PictureCallback postview` — имя метода обратного вызова для передачи в программу данных изображения в полностью обработанном виде (если поддерживается данным типом камеры).

Таким образом, для захвата изображения видеокамерой нам необходимо реализовать два интерфейса:

- ❑ `Camera.ShutterCallback` — для перехвата момента фактического захвата изображения;
- ❑ `Camera.PictureCallback` — для получения образа захваченного с камеры изображения.

Однако, поскольку все параметры в методах `takePicture()` допускают значение `null`, необязательно в коде программы реализовывать оба интерфейса.

Образ захваченного с камеры изображения в виде массива байтов можно преобразовать, например, в объект `Bitmap` и сохранить его в файловой системе мобильного устройства:

```
PictureCallback jpg = new PictureCallback() {
    @Override
    public void onPictureTaken(byte[] arg0, Camera arg1) {
        Bitmap bitmapPicture
            = BitmapFactory.decodeByteArray(arg0, 0, arg0.length);

        // Далее можно, например, сохранить полученный объект Bitmap
        // на карте памяти мобильного устройства

        // Если камера больше не нужна, освобождаем ресурс
        camera.stopPreview();
        camera.release();
    }
};
```

Теперь рассмотрим использование захвата изображения камерой в реальном приложении. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `CameraTakePicture`;
- ☐ **Application name** — `Take picture`;
- ☐ **Package name** — `com.samples.camera.takepicture`;
- ☐ **Create Activity** — `CameraPreviewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch39_CameraTakePicture`.

В файл компоновки главного окна приложения `overlay.xml` к уже имеющимся двум кнопкам для запуска и останова камеры добавим дополнительную кнопку `ImageButton` с идентификатором `bTake`, при нажатии которой будет происходить захват изображения. Файл компоновки окна представлен в листинге 39.12.

Листинг 39.12. Файл `overlay.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="bottom|right">

    <ImageButton
        android:id="@+id/bStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```



```
        android:src="@drawable/start"
        android:layout_margin="1px"/>
<ImageButton
    android:id="@+id/bTake"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/fix"
    android:layout_margin="1px"/>
<ImageButton
    android:id="@+id/bStop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/stop"
    android:layout_margin="1px"/>

</LinearLayout>
```

В коде класса `CameraPreviewActivity` реализуем дополнительную функциональность для захвата изображения по принципу, описанному ранее в этом разделе. Измененный код класса `CameraPreviewActivity` представлен в листинге 39.13.

Листинг 39.13. Файл класса окна приложения `CameraPreviewActivity.java`

```
package com.samples.camera.takepicture;

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

import java.io.IOException;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.view.LayoutInflater;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.ImageButton;
import android.widget.Toast;

public class CameraPreviewActivity extends Activity
    implements SurfaceHolder.Callback, View.OnClickListener {

    private ImageButton bStart;
    private ImageButton bStop;
    private ImageButton bTake;
```

```

private SurfaceView surView;
private SurfaceHolder surHolder;

private Camera camera;
private boolean isCameraPreview = false;

private ShutterCallback shutter = new ShutterCallback(){
    @Override
    public void onShutter() { }
};

private PictureCallback raw = new PictureCallback(){
    @Override
    public void onPictureTaken(byte[] arg0, Camera arg1) { }
};

private PictureCallback jpg = new PictureCallback(){
    @Override
    public void onPictureTaken(byte[] arg0, Camera arg1) {
        Bitmap bitmapPicture
            = BitmapFactory.decodeByteArray(arg0, 0, arg0.length);
        camera.stopPreview();
        camera.release();
        isCameraPreview = false;

        bStart.setEnabled(!isCameraPreview);
        bStop.setEnabled(isCameraPreview);
    }
};

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

    surView = (SurfaceView)findViewById(R.id.surfaceview);
    surHolder = surView.getHolder();
    surHolder.addCallback(this);

    LayoutInflater inflater = LayoutInflater.from(getBaseContext());
    View overlay = inflater.inflate(R.layout.overlay, null);
    LayoutParams params = new LayoutParams(
        LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
    addContentView(overlay, params);

    bStart = (ImageButton)overlay.findViewById(R.id.bStart);
    bStop = (ImageButton)overlay.findViewById(R.id.bStop);
    bTake = (ImageButton)overlay.findViewById(R.id.bTake);

```

```
bStart.setOnClickListener(this);
bStop.setOnClickListener(this);
bTake.setOnClickListener(this);

bTake.setEnabled(!isCameraPreview);
bStop.setEnabled(!isCameraPreview);
}

@Override
public void surfaceChanged(
    SurfaceHolder holder, int format, int width, int height) { }

@Override
public void surfaceCreated(SurfaceHolder holder) { }

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    // Если камера открыта, закрываем камеру
    // и освобождаем ресурсы
    if (isCameraPreview) {
        camera.stopPreview();
        camera.release();
        isCameraPreview = false;
    }
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bStart:
            try {
                camera = Camera.open();
                camera.setPreviewDisplay(surHolder);
                camera.startPreview();
                isCameraPreview = true;

                bStart.setEnabled(!isCameraPreview);
                bStop.setEnabled(isCameraPreview);
            }
            catch (IOException e) {
                Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
            }
            break;

        case R.id.bTake:
            camera.takePicture(shutter, raw, jpg);
            break;

        case R.id.bStop:
            camera.stopPreview();
    }
}
```

```

        camera.release();
        isCameraPreview = false;

        bStart.setEnabled(!isCameraPreview);
        bStop.setEnabled(isCameraPreview);
        break;
    }
}
}

```

Скомпилируйте проект и запустите его на мобильном устройстве. При нажатии средней кнопки будет происходить захват изображения видекамеры. Внешний вид приложения представлен на рис. 39.7.

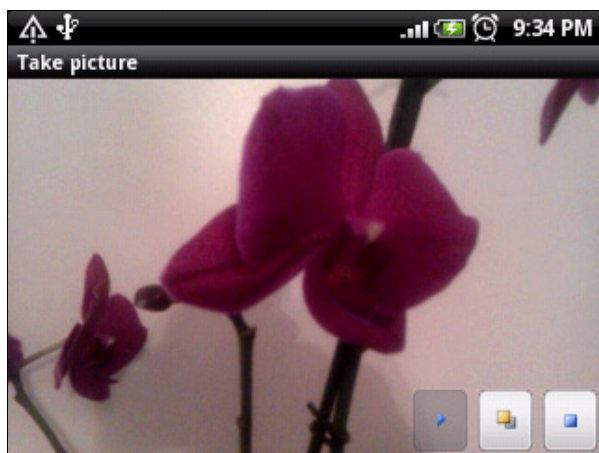


Рис. 39.7. Приложение с захватом изображения камерой

Использование автофокуса

Если встроенная камера поддерживает режим автофокуса, вы можете использовать этот режим в своем приложении, например, производить захват изображения только в том случае, если фотографируемый объект находится в фокусе. Дело в том, что для настройки фокуса камере необходимо некоторое время, особенно при резких изменениях расстояния до объекта.

Для отслеживания настройки фокуса нужно в коде приложения создать объект `AutoFocusCallback` и переопределить его метод `onAutoFocus()`:

```

AutoFocusCallback autoFocusCallback = new AutoFocusCallback() {
    @Override
    public void onAutoFocus(boolean arg0, Camera arg1) {
        // Действия при установке фокуса
    }
};

```

Затем созданный объект `AutoFocusCallback` надо добавить к объекту `Camera` с помощью метода `autoFocus()` следующим образом:

```
Camera camera = Camera.open();
camera.autoFocus(autoFocusCallback);
```

Рассмотрим теперь использование автофокуса в нашем приложении. Создадим новое приложение, работающее с камерой с поддержкой автофокуса, которое будет позволять делать снимок изображения камеры только при настроенном фокусе. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `CameraAutoFocus`;
- ☐ **Application name** — `Set Autofocus`;
- ☐ **Package name** — `com.samples.camera.autofocus`;
- ☐ **Create Activity** — `CameraPreviewActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch39_CameraAutoFocus`.

В коде класса `CameraPreviewActivity` добавим функциональность для отслеживания автофокуса. Код класса `CameraPreviewActivity` приложения представлен в листинге 39.14.

Листинг 39.14. Файл класса окна приложения `CameraPreviewActivity.java`

```
package com.samples.camera.autofocus;

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

import java.io.IOException;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.hardware.Camera;
import android.hardware.Camera.AutoFocusCallback;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.view.LayoutInflater;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.ViewGroup.LayoutParams;
import android.widget.ImageButton;
import android.widget.Toast;

public class CameraPreviewActivity extends Activity
    implements SurfaceHolder.Callback, View.OnClickListener {
    private ImageButton bStart;
```

```

private ImageButton bStop;
private ImageButton bTake;

private SurfaceView surView;
private SurfaceHolder surHolder;

private Camera camera;
private boolean isCameraPreview = false;

private ShutterCallback shutter = new ShutterCallback(){
    @Override
    public void onShutter() { }
};

private PictureCallback raw = new PictureCallback(){
    @Override
    public void onPictureTaken(byte[] arg0, Camera arg1) { }
};

private PictureCallback jpg = new PictureCallback(){
    @Override
    public void onPictureTaken(byte[] arg0, Camera arg1) {
        Bitmap bitmapPicture
            = BitmapFactory.decodeByteArray(arg0, 0, arg0.length);
        camera.startPreview();
    }
};

private AutoFocusCallback autoFocusCallback = new AutoFocusCallback(){

    @Override
    public void onAutoFocus(boolean arg0, Camera arg1) {
        // Когда фокус настроен, разблокируем кнопку
        bTake.setEnabled(true);
    }
};

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

    surView = (SurfaceView)findViewById(R.id.surfaceview);
    surHolder = surView.getHolder();
    surHolder.addCallback(this);
    surHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    LayoutInflater inflater = LayoutInflater.from(getBaseContext());
    View overlay = inflater.inflate(R.layout.overlay, null);
    LayoutParams params = new LayoutParams(
        LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
    addContentView(overlay, params);
}

```

```
bStart = (ImageButton)overlay.findViewById(R.id.bStart);
bStop = (ImageButton)overlay.findViewById(R.id.bStop);
bTake = (ImageButton)overlay.findViewById(R.id.bTake);

bStart.setOnClickListener(this);
bStop.setOnClickListener(this);
bTake.setOnClickListener(this);

// Блокируем кнопки
bStop.setEnabled(!isCameraPreview);
bTake.setEnabled(false);
}

@Override
public void surfaceChanged(
    SurfaceHolder holder, int format, int width, int height) { }

@Override
public void surfaceCreated(SurfaceHolder holder) { }

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    // Если камера открыта, закрываем камеру
    // и освобождаем ресурсы
    if (isCameraPreview) {
        camera.stopPreview();
        camera.release();
        isCameraPreview = false;
    }
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bStart:
            try {
                camera = Camera.open();
                camera.setPreviewDisplay(surHolder);
                camera.startPreview();
                isCameraPreview = true;
                camera.autoFocus(autoFocusCallback);
                bStart.setEnabled(!isCameraPreview);
                bStop.setEnabled(isCameraPreview);
            }
            catch (IOException e) {
                Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
            }
            break;
    }
}
```

```
case R.id.bTake:
    camera.takePicture(shutter, raw, jpg);
    break;

case R.id.bStop:
    camera.stopPreview();
    camera.release();
    isCameraPreview = false;

    bStart.setEnabled(!isCameraPreview);
    bStop.setEnabled(isCameraPreview);
    break;
}
}
```

Скомпилируйте проект и запустите его на мобильном устройстве. Теперь приложение сможет производить захват изображения с камеры только в том случае, если изображение находится в фокусе. Если изображение расфокусировано, кнопка захвата изображения будет недоступна. При установке фокуса кнопка захвата изображения будет разблокирована, и можно будет произвести захват изображения. Внешний вид приложения представлен на рис. 39.8.

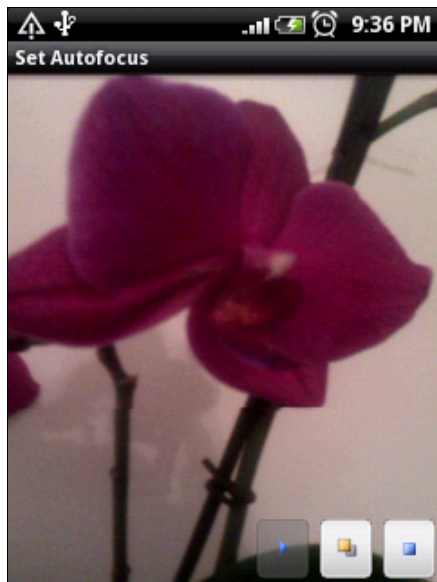


Рис. 39.8. Приложение с отслеживанием автофокуса

Резюме

В этой главе мы научились работать со встроенной камерой. Как вы убедились, система Android предоставляет большой набор функциональностей, которые можно использовать для создания собственных приложений с широкими возможностями управления видеокамерой.

В следующей главе будет рассмотрена работа с другим оборудованием, предоставляемым мобильными устройствами на платформе Android, — это различные встроенные датчики для взаимодействия мобильного устройства с окружающей средой.



ГЛАВА 40

Встроенные датчики

Современные мобильные устройства давно уже не являются обычными телефонами, предназначенными только для звонков и передачи SMS. В комплект оборудования мобильных устройств Android входят разнообразные датчики (сенсоры), такие как камера, акселерометр, датчик магнитного поля, датчик температуры и датчик расстояния. Набор датчиков зависит от конкретной модели и производителя мобильного устройства и может значительно отличаться у двух разных моделей даже одного производителя.

В этой главе будет рассказано о возможностях использования встроенных датчиков и об управлении ими из приложений. Android SDK обеспечивает API для получения доступа к оборудованию мобильного устройства.

С выходом каждой новой версии платформы Android постоянно добавляется поддержка все новых типов датчиков. Например, в версии Android 2.3.3 была добавлена поддержка таких датчиков, как гироскоп, датчик давления и датчик ротации. В версии 4.0 — датчик относительной влажности.

Для работы с этой главой понадобится реальное мобильное устройство, т. к. эмулятор не может полностью имитировать оборудование мобильного устройства.

Библиотека для работы с датчиками

Android SDK обеспечивает простой доступ к датчикам на устройстве и предоставляет библиотеку `android.hardware` для работы с ними в приложениях. Эта библиотека включает в себя набор классов и интерфейсов для доступа к встроенным датчикам мобильного устройства, чтения измеряемых датчиками значений и мониторинга изменений этих значений.

Управление датчиками

Доступ к датчикам устройства в программном коде выполняется через объект класса `SensorManager`. Экземпляр `SensorManager` можно получить с помощью метода `getSystemService()`, передав ему в качестве параметра значение `SENSOR_SERVICE`:

```
SensorManager sensors =  
    (SensorManager) getSystemService(Context.SENSOR_SERVICE);
```

Класс `SensorManager` содержит набор констант, которые определяют характеристики датчиков Android-устройства, и методов для управления этими датчиками. Их мы рассмотрим далее в этой главе.

Характеристики конкретного датчика представляет класс `Sensor`. Этот класс используется для описания свойств каждого датчика, включая его тип, название, изготовителя, точность и диапазон измерения.

Класс `Sensor` включает ряд констант, используемых для определения типа датчика. Набор этих констант постоянно пополняется. На момент выхода версии Android SDK 4.03 в классе `Sensor` содержится следующий набор констант, определяющих типы датчиков:

- ❑ `TYPE_ACCELEROMETER` — акселерометр, который возвращает текущее ускорение для трех осей измерения;
- ❑ `TYPE_AMBIENT_TEMPERATURE` — датчик температуры окружающей среды;
- ❑ `TYPE_GRAVITY` — датчик гравитации;
- ❑ `TYPE_GYROSCOPE` — гироскопический датчик, измеряющий скорость вращения корпуса мобильного устройства в 3D-системе координат;
- ❑ `TYPE_LIGHT` — датчик освещенности;
- ❑ `TYPE_LINEAR_ACCELERATION` — линейный акселерометр;
- ❑ `TYPE_MAGNETIC_FIELD` — датчик магнитного поля;
- ❑ `TYPE_ORIENTATION` — датчик ориентации устройства в 3D-системе координат;
- ❑ `TYPE_PRESSURE` — датчик давления;
- ❑ `TYPE_PROXIMITY` — датчик для определения расстояния до объекта;
- ❑ `TYPE_RELATIVE_HUMIDITY` — датчик относительной влажности;
- ❑ `TYPE_ROTATION_VECTOR` — датчик ротации. Этот датчик представляет ориентацию мобильного устройства в виде комбинации угла поворота и оси;
- ❑ `TYPE_ALL` — константа, определяющая все типы датчиков.

Кроме того, в классе `Sensor` есть ряд методов для получения модели и данных о производителе датчика:

- ❑ `getName()` — возвращает имя датчика в виде строки;
- ❑ `getVendor()` — возвращает имя вендора этого датчика;
- ❑ `getVersion()` — возвращает номер версии датчика.

Также есть набор методов для чтения технических характеристик датчика:

- ❑ `getMaximumRange()` — возвращает верхнюю величину измеряемого диапазона;
- ❑ `getMinDelay()` — возвращает минимальную задержку измерения в миллисекундах;
- ❑ `getPower()` — возвращает значение силы тока в миллиамперах, потребляемого данным сенсором. Это довольно важная характеристика, т. к. некоторые датчики мобильного устройства потребляют достаточно большой ток во время работы;
- ❑ `getResolution()` — возвращает разрешающую способность датчика.

Набор методов, определенных в этом классе, так же как и константы, изменяется с выходом новых версий Android SDK. Например, метод `getMinDelay()` появился только в

версии API Level 9. Поэтому при проектировании приложений внимательно применяйте данный класс, если эти приложения в будущем будут использоваться на старых платформах Android. Дело в том, что не каждое устройство на платформе Android поддерживает все датчики из класса `SensorManager`, поэтому в приложении необходимо предусмотреть корректную ситуацию, когда требуемый датчик отсутствует.

Поиск доступных датчиков на мобильном устройстве

Мобильные телефоны, в зависимости от модели и производителя, имеют различный набор датчиков. Поэтому перед запуском приложения, использующего датчики, необходимо выяснить, какие датчики доступны на данной модели телефона. Для поиска датчиков в классе `SensorManager` определены два метода:

- ❑ `getSensorList(int type);`
- ❑ `getDefaultSensor(int type).`

Чтобы получить полный список датчиков, доступных на данном устройстве, используют метод `getSensorList()` с параметром `Sensor.TYPE_ALL`:

```
SensorManager manager = (SensorManager) getSystemService(  
    Context.SENSOR_SERVICE);  
...  
List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ALL);
```

Можно использовать метод `getSensorList()`, чтобы получить список всех доступных датчиков определенного типа, например, как показано в следующем коде, который возвращает все доступные объекты датчика давления:

```
SensorManager manager = (SensorManager) getSystemService(  
    Context.SENSOR_SERVICE);  
...  
List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ACCELEROMETER);
```

Если на данном устройстве присутствует несколько датчиков одного типа, то один из них всегда является основным, т. е. датчиком по умолчанию. Для получения такого датчика используется метод `getDefaultSensor()`:

```
Sensor defaultGyroscope =  
    sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Давайте теперь разработаем приложение для поиска и получения характеристик встроенных датчиков на мобильном устройстве. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `GetDeviceSensors`;
- ❑ **Application name** — `Get available sensors`;
- ❑ **Package name** — `com.samples.hardware.getsensors`;
- ❑ **Create Activity** — `GetSensorsActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch40_GetDeviceSensors`.

Для доступа к датчикам разрешения не нужны, поэтому в файл `AndroidManifest` никаких дополнений вносить не будем. В файле компоновки создайте один виджет `TextView` для вывода данных о доступных датчиках.

Код класса `GetSensorsActivity` будет простым. В этом классе в методе `onCreate()` мы получаем экземпляр `SensorManager` и с помощью метода `getSensorList()` получаем список всех доступных на мобильном устройстве датчиков в виде объектов `Sensor`, из которых мы можем получить информацию о каждом датчике.

Код класса `GetSensorsActivity` представлен в листинге 40.1.

Листинг 40.1. Файл класса окна приложения `GetSensorsActivity.java`

```
package com.samples.hardware.getsensors;

import java.util.List;
import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;

public class GetSensorsActivity extends Activity {

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

        TextView text = (TextView)findViewById(R.id.text);

        SensorManager manager = (SensorManager) getSystemService(
            Context.SENSOR_SERVICE);
        List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ALL);
        text.append("Available sensors:");
        for (Sensor s : sensors) {
            text.append("\n" + s.getName());
        }
    }
}
```

Скомпилируйте приложение и запустите его на мобильном устройстве. Например, для модели HTC Wildfire приложение выведет набор датчиков, имеющихся на этом устройстве, и их типы, как представлено на рис. 40.1.

Конечно, для другого мобильного устройства набор датчиков будет отличаться от представленного на рис. 40.1.

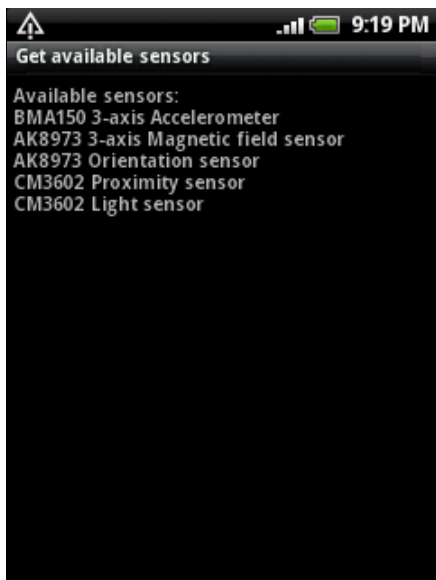


Рис. 40.1. Список доступных сенсоров на мобильном устройстве

Отслеживание изменений, измеряемых датчиками значений

Для мониторинга изменений значений датчиков в пакете `android.hardware` есть интерфейс `SensorEventListener`. Этот интерфейс включает в себя два метода: `onAccuracyChanged()` и `onSensorChanged()`, которые вы должны реализовать в коде своей программы.

Метод `onAccuracyChanged()` вызывается при изменении точности показаний датчика. Первым входным параметром метода является объект класса `Sensor`, представляющий датчик. Второй параметр представляет целое число, которое соответствует новому значению точности этого датчика. Данное число может принимать значение одной из констант, определенных в классе `Sensor`:

- ☐ `SENSOR_STATUS_ACCURACY_HIGH` — большая точность измерения;
- ☐ `SENSOR_STATUS_ACCURACY_MEDIUM` — средняя точность измерений;
- ☐ `SENSOR_STATUS_ACCURACY_LOW` — низкая точность измерения;
- ☐ `SENSOR_STATUS_UNRELIABLE` — измерениям, полученным этим датчиком, нельзя доверять.

Метод `onSensorChanged()` вызывается при изменении значения измеряемой величины, считываемой с датчика. В качестве параметра передается объект класса `SensorEvent`. Этот класс представляет событие, происходящее на датчике, и состоит всего из четырех полей:

- ☐ `accuracy` — точность измерений;
- ☐ `sensor` — объект класса `Sensor`, представляющий этот датчик;
- ☐ `timestamp` — время в наносекундах, в течение которого это событие произошло;

- ❑ `values` — целое число, представляющее собой массив чисел с плавающей запятой, отражающих показания датчика. Массив может иметь размерность 1 или 3. Дело в том, что некоторые датчики измеряют только одно значение, тогда как существуют датчики, предоставляющие три значения. Например, датчики освещения и расстояния имеют всего одно измеряемое значение, а датчики акселерометра и магнитного поля имеют по три значения (для каждой из осей координат X, Y, Z).

Таким образом, в коде приложения объект `SensorEventListener` объявляется следующим образом:

```
SensorEventListener listener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        // Действия при изменении измеряемого датчиком значения
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // Действия при изменении точности показаний датчика
    }
};
```

Далее, для взаимодействия с датчиком в коде приложения необходимо зарегистрировать слушатель `SensorEventListener` для приема событий, связанных с одним или несколькими датчиками. Регистрация слушателя событий осуществляется с помощью метода `registerListener()` класса `SensorManager`. В этот метод необходимо передать три параметра:

- ❑ объект `SensorEventListener`, рассмотренный ранее;
- ❑ объект `Sensor`, представляющий датчик, который мы будем отслеживать;
- ❑ целочисленную константу, устанавливающую задержку получения события.

Последний параметр нужен для определения задержки чтения данных с датчика и может принимать следующие значения:

- ❑ `SENSOR_DELAY_FASTEST` — чтение данных с датчика без задержки;
- ❑ `SENSOR_DELAY_NORMAL` — стандартная задержка для чтения данных с датчика, используется по умолчанию;
- ❑ `SENSOR_DELAY_GAME` — задержка используется в играх, в которых учитывается поворот корпуса мобильного устройства, например автогонки или подобные им игры;
- ❑ `SENSOR_DELAY_UI` — стандартная задержка при повороте корпуса мобильного устройства для переключения режима дисплея с `Portrait` на `Landscape` и обратно.

Например, при работе с датчиком ориентации, для отслеживания поворота корпуса мобильного устройства подключить слушатель событий можно следующим образом:

```
SensorManager manager = (SensorManager) getSystemService(
    Context.SENSOR_SERVICE);
List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ORIENTATION);
```

```
// Проверка доступности датчика на устройстве
if(sensors.size() != 0) {
    manager.registerListener(listener,
        sensors.get(0), SensorManager.SENSOR_DELAY_GAME);
}
```

Работа с датчиками в приложении

Сейчас мы рассмотрим особенности использования датчиков разных типов и отслеживания изменений величин, измеряемых этими датчиками в приложении. В целом, работа с датчиками разных типов примерно одинаковая и отличается в основном тем, что необходимо обрабатывать один параметр для простых датчиков или три параметра для более сложных датчиков.

Кроме того, при необходимости для датчиков можно устанавливать режимы задержки измерений, которые были описаны ранее в этой главе.

Датчик освещенности

Световой датчик обычно используется, чтобы динамически управлять яркостью экрана. Это простой датчик, измеряющий только одно значение — уровень освещенности в люксах (lux).

Если вам необходимо сравнить полученную величину освещенности с каким-то стандартным значением, то для этого в классе `SensorManager` есть константы, представляющие стандартные величины освещенности, которые соответствуют уровням солнечной освещенности при различных условиях. Например, есть значения освещенности в солнечную погоду, при облачности, лунном свете и т. д. (`LIGHT_CLOUDY`, `LIGHT_FULLMOON`, `LIGHT_NO_MOON`, `LIGHT_OVERCAST`, `LIGHT_SHADE`, `LIGHT_SUNLIGHT`, `LIGHT_SUNLIGHT_MAX`, `LIGHT_SUNRISE` и пр.).

Теперь разработаем приложение, работающее с датчиком освещенности. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `LightSensor`;
- ☐ **Application name** — `Light sensor`;
- ☐ **Package name** — `com.samples.hardware.light`;
- ☐ **Create Activity** — `SensorActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch40_LightSensor`.

Данное приложение мы будем использовать для работы с другими датчиками далее в этой главе, постепенно изменяя и расширяя код в зависимости от типа используемого датчика.

Файл компоновки окна приложения `main.xml` будет содержать два виджета `TextView`: один со статической надписью **Light level**, другой — для вывода значения освещенности. Код файла компоновки `main.xml` представлен в листинге 40.2.

Листинг 40.2. Файл компоновки окна приложения main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <TextView
        android:text="@string/name0"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginRight="5px"
        android:textSize="24sp"/>

    <TextView
        android:id="@+id/text0"
        android:text="0.0"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_width="wrap_content"
        android:textSize="24sp"/>

</LinearLayout>
```

Для удобства все надписи, используемые в приложении, мы будем хранить в файле строковых ресурсов. Файл строковых ресурсов strings.xml нашего приложения представлен в листинге 40.3.

Листинг 40.3. Файл строковых ресурсов strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Light sensor</string>
    <string name="error_msg">This sensor is not available on device</string>
    <string name="name0">Light level:</string>
</resources>
```

В классе `SensorActivity` мы должны прочитать доступные датчики освещенности, зарегистрировать слушатель `SensorEventListener` для отслеживания изменений уровня освещенности и вывести текущее значение уровня освещенности на экран мобильного устройства.

Код класса окна приложения `SensorActivity` представлен в листинге 40.4.

Листинг 40.4. Файл класса окна приложения SensorActivity.java

```
package com.samples.hardware.light;

import android.app.Activity;
import android.app.AlertDialog;
```



```
import android.content.Context;
import android.content.DialogInterface;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;

import java.util.List;

public class SensorActivity extends Activity {
    private TextView text0;

    private SensorManager manager;
    private List<Sensor> sensors;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
        text0 = (TextView)findViewById(R.id.text0);
    }

    @Override
    public void onStart() {
        super.onStart();

        manager =
            (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        sensors =
            manager.getSensorList(Sensor.TYPE_LIGHT);

        if(sensors.size() != 0) {
            manager.registerListener(listener,
                sensors.get(0), SensorManager.SENSOR_DELAY_NORMAL);
        }
        else {
            // Если датчика нет, отображаем диалоговое окно
            // с предупреждением и закрываем приложение
            AlertDialog.Builder builder = new AlertDialog.Builder(this);

            builder.setTitle(R.string.app_name);
            builder.setMessage(R.string.error_msg);
            builder.setPositiveButton(
                "OK", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int item) {
                        SensorActivity.this.finish();
                    }
                });
        }
    }
};
```

```
        builder.show();
    }
}

private SensorEventListener listener = new SensorEventListener() {
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) { }

    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        // Выводим текущее значение на экран
        text0.setText(String.valueOf(sensorEvent.values[0]));
    }
};
}
```

Скомпилируйте приложение и запустите его на мобильном устройстве. Испытайте работу приложения при изменении уровня освещенности, приближая и удаляя мобильный телефон к источнику света, например к настольной лампе. Приложение будет отображать изменение уровня освещенности. Внешний вид приложения представлен на рис. 40.2.

Как уже говорилось ранее в этой главе, для приложений, работающих с оборудованием мобильного телефона, обязательно нужно предусматривать корректную обработку

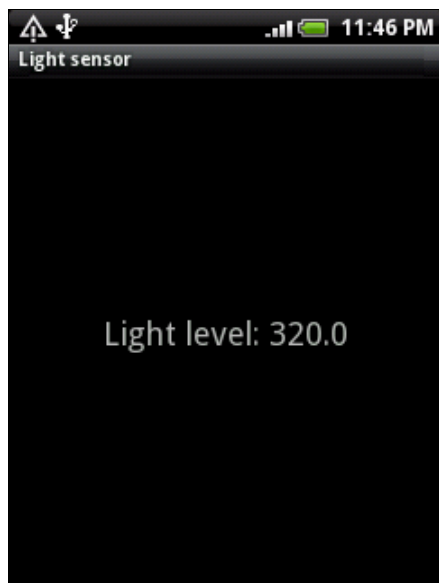


Рис. 40.2. Измерение уровня освещенности

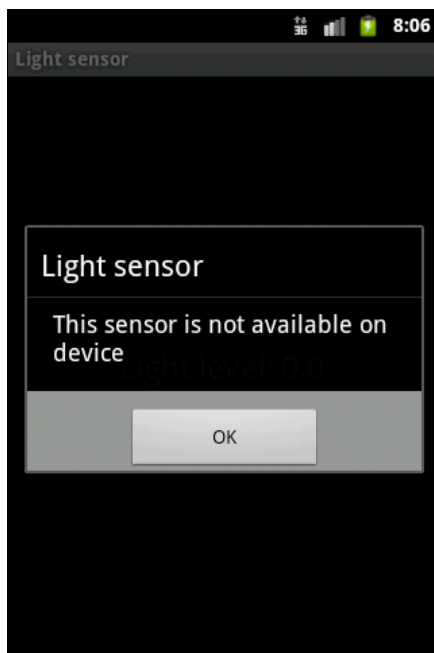


Рис. 40.3. Диалоговое окно при отсутствии датчика данного типа на устройстве

ситуации, когда датчик данного типа отсутствует на мобильном устройстве. В коде класса окна приложения из листинга 40.4 в случае отсутствия датчика предусмотрено отображение диалогового окна с соответствующим сообщением, и после нажатия пользователем кнопки **ОК** приложение закрывается.

Теперь попробуйте запустить приложение на эмуляторе Android. Так как эмулятор не позволяет имитировать датчики, приложение выдаст диалоговое окно с сообщением, как показано на рис. 40.3.

Датчик расстояния

Работа с датчиком расстояния аналогична работе с датчиком освещенности, рассмотренным в предыдущем разделе. Этот датчик также измеряет только один параметр, который показывает удаленность до объекта в метрах.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch40_ProximitySensor.

Для работы приложения с датчиком расстояния измените одну строку кода в методе `onCreate()` класса `SensorActivity` (см. листинг 40.4), как показано в листинге 40.5.

Листинг 40.5. Изменения в классе `SensorActivity.java`

```
...
public void onStart() {
    super.onStart();

    manager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    sensors = manager.getSensorList(Sensor.TYPE_PROXIMITY);
    ...
}
```

Поменяйте также строковые значения в файле `strings.xml`, как показано в листинге 40.6.

Листинг 40.6. Файл строковых ресурсов `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Proximity sensor</string>
    <string name="error_msg">This sensor is not available on device</string>
    <string name="name0">Proximity:</string>
</resources>
```

Скомпилируйте приложение и запустите его на мобильном устройстве. Поэкспериментируйте с приложением. Внешний вид приложения для работы с датчиком измерения расстояния представлен на рис. 40.4.

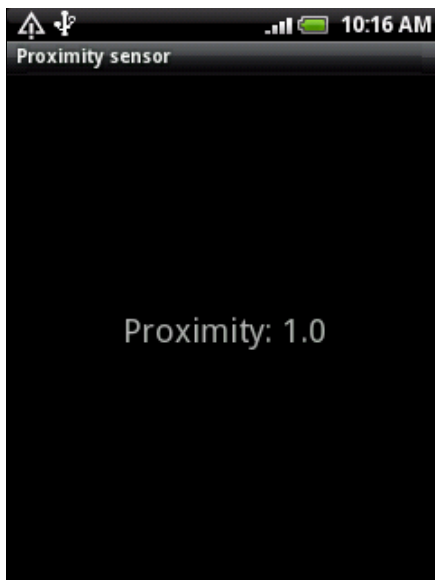


Рис. 40.4. Измерение расстояния до объекта

Датчик ориентации

Датчик ориентации показывает расположение корпуса мобильного устройства в пространстве относительно трех осей, в качестве единиц измерения используются градусы.

Датчик ориентации, в отличие от предыдущих рассмотренных датчиков, более сложный и измеряет три параметра:

- ☐ Pitch — вращение корпуса относительно горизонтальной оси, направленной поперек корпуса телефона (ось X);
- ☐ Roll — вращение корпуса относительно вертикальной оси, направленной вдоль корпуса телефона (ось Y);
- ☐ Azimuth — вращение корпуса относительно вертикальной оси, когда телефон лежит на ровной горизонтальной поверхности (ось Z).

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch40_OrientationSensor_Azimuth.

Кстати, совсем необязательно для работы с приложением использовать все три параметра. Например, этот датчик можно применять только для измерения азимута, если остальные параметры не нужны.

Параметр `Azimuth` является первым элементом в массиве параметров, передаваемом в качестве аргумента в метод `onSensorChanged()`, поэтому тело метода `onSensorChanged()` из листинга 40.4 нам изменять не надо, достаточно лишь изменить доступ к датчику ориентации в классе `SensorActivity`, как показано в листинге 40.7.

Листинг 40.7. Изменения в классе `SensorActivity`

```
...  
public void onStart() {  
    super.onStart();  
  
    manager = SensorManager.getSystemService(Context.SENSOR_SERVICE);  
    sensors = manager.getSensorList(Sensor.TYPE_ORIENTATION);  
    ...  
}
```

Поменяйте также строковые значения в файле `strings.xml`, как показано в листинге 40.8.

Листинг 40.8. Файл строковых ресурсов `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string name="app_name">Azimuth Orientation sensor</string>  
    <string name="error_msg">This sensor is not available on device</string>  
    <string name="name0">Azimuth:</string>  
</resources>
```

Скомпилируйте приложение и запустите его на мобильном устройстве. Поэкспериментируйте с приложением, вращая корпус мобильного устройства. Внешний вид приложения для работы с датчиком ориентации представлен на рис. 40.5.

Используя датчик ориентации, из мобильного устройства можно сделать измеритель уровня наклона поверхности, работающий так же, как и строительный уровень. Для этого нам потребуется небольшая модификация нашего приложения.

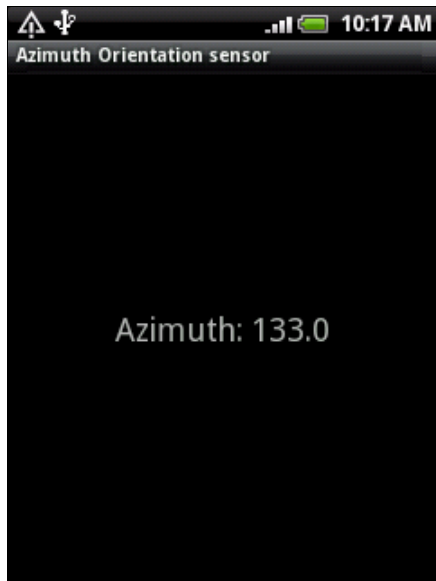


Рис. 40.5. Изменение ориентации в пространстве

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch40_OrientationSensor_Horizont`.

Сначала в файл компоновки главного окна приложения `main.xml` добавим дополнительную пару виджетов `TextView` для отображения двух других измеряемых параметров датчика ориентации. Код измененного файла компоновки окна приложения `main.xml` представлен в листинге 40.9.

Листинг 40.9. Файл компоновки окна приложения main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal"
        android:layout_margin="10px"
        android:gravity="center">

        <TextView
            android:text="@string/name1"
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_marginRight="5px"
            android:textSize="24sp"/>

        <TextView
            android:id="@+id/text1"
            android:text="0.0"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:layout_width="wrap_content"
            android:textSize="24sp"/>

    </LinearLayout>
    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal"
        android:layout_margin="10px"
        android:gravity="center">

        <TextView
            android:text="@string/name2"
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_marginRight="5px"
            android:textSize="24sp"/>

        <TextView
            android:id="@+id/text2"
            android:text="0.0"
            android:layout_height="wrap_content"
            android:gravity="center"
```

```
        android:layout_width="wrap_content"
        android:textSize="24sp"/>
    </LinearLayout>
</LinearLayout>
```

В файл строковых ресурсов `strings.xml` также внесем изменения и добавим новые наименования параметров, как показано в листинге 40.10.

Листинг 40.10. Файл строковых ресурсов `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Horizontal orientation sensor</string>
    <string name="error_msg">This sensor is not available on device</string>
    <string name="name1">Pitch:</string>
    <string name="name2">Roll:</string>
</resources>
```

В классе `SensorActivity` в теле метода `onSensorChanged()` объекта `SensorEventListener` мы будем выводить в текстовые поля показания датчика ориентации, используя 2-й и 3-й параметр (`Roll` и `Pitch`). Код класса `SensorActivity` представлен в листинге 40.11.

Листинг 40.11. Файл класса окна приложения `SensorActivity.java`

```
package com.samples.hardware.horizont;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;
import java.util.List;

public class SensorActivity extends Activity {

    private TextView text1;
    private TextView text2;

    private SensorManager manager;
    private List<Sensor> sensors;

    private SensorEventListener listener = new SensorEventListener() {
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) { }
```

```

@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    text1.setText(String.valueOf(sensorEvent.values[1]));
    text2.setText(String.valueOf(sensorEvent.values[2]));
}
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    text1 = (TextView) findViewById(R.id.text1);
    text2 = (TextView) findViewById(R.id.text2);
}

@Override
public void onStart() {
    super.onStart();

    manager =
        (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    sensors =
        manager.getSensorList(Sensor.TYPE_ORIENTATION);

    if(sensors.size() != 0) {
        manager.registerListener(listener,
            sensors.get(0), SensorManager.SENSOR_DELAY_NORMAL);
    }
    else {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);

        builder.setTitle(R.string.app_name);
        builder.setMessage(R.string.error_msg);
        builder.setPositiveButton(
            "OK", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int item) {
                    SensorActivity.this.finish();
                }
            });

        builder.show();
    }
}
}
}

```

Скомпилируйте приложение и запустите его на мобильном устройстве. Телефон с работающим приложением напоминает строительный уровень для измерения угла наклона.

на поверхности. Внешний вид приложения для работы с датчиком ориентации для измерения уровня наклона поверхности представлен на рис. 40.6.

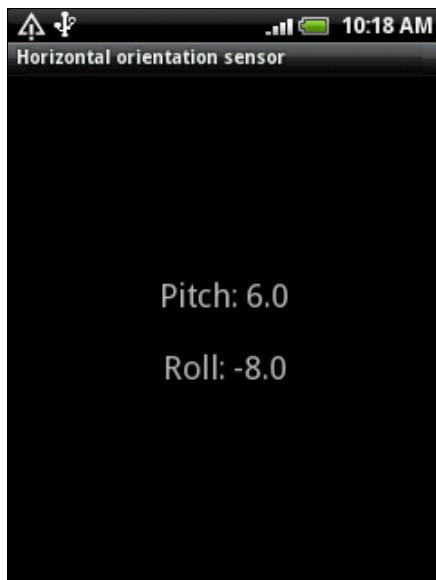


Рис. 40.6. Датчик ориентации как измеритель уровня наклона поверхности

Акселерометр

Акселерометр, как понятно из его названия, используется для измерения ускорения. Этот датчик способен измерять ускорение в трехмерной системе координат и работает с тремя параметрами:

- ☐ Longitudinal — ускорение, направленное вдоль корпуса мобильного устройства;
- ☐ Lateral — ускорение, направленное поперек корпуса мобильного устройства;
- ☐ Vertical — вертикальное ускорение.

Акселерометр измеряет ускорение в м/с^2 . Особенность датчика акселерометра в том, что он неспособен отличить ускорение, вызванное перемещением корпуса мобильного устройства, от ускорения из-за земной силы тяжести. Однако при использовании акселерометра можно определять изменение характера перемещения корпуса телефона, что применяется при создании игр.

Для оценки относительной величины ускорения в классе `SensorManager` имеется константа `STANDARD_GRAVITY`, определяющая значение ускорения свободного падения на поверхности Земли. Кроме того, в этом классе определены константы, представляющие стандартные величины ускорения свободного падения всех планет Солнечной системы (`GRAVITY_MERCURY`, `GRAVITY_VENUS` и др.), Луны (`GRAVITY_MOON`), Солнца (`GRAVITY_SUN`) и даже черной дыры — `GRAVITY_DEATH_STAR_I`!

Теперь модифицируем наше приложение для работы с акселерометром.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch40_Accelerometer3DSensor`.

В файле компоновки у нас будет три текстовых поля для обозначения осей и три поля для вывода значений ускорения относительно этих осей. Код файла компоновки `main.xml` представлен в листинге 40.12.

Листинг 40.12. Файл компоновки окна приложения main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center">

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal"
        android:layout_margin="10px"
        android:gravity="center">

        <TextView
            android:text="@string/name0"
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_marginRight="5px"
            android:textSize="24sp"/>

        <TextView
            android:id="@+id/text0"
            android:text="0.0"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:layout_width="wrap_content"
            android:textSize="24sp"/>

    </LinearLayout>

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal"
        android:layout_margin="10px"
        android:gravity="center">

        <TextView
            android:text="@string/name1"
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:layout_marginRight="5px"
            android:textSize="24sp"/>

        <TextView
            android:id="@+id/text1"
```

```

        android:text="0.0"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_width="wrap_content"
        android:textSize="24sp"/>
</LinearLayout>

<LinearLayout
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:orientation="horizontal"
    android:layout_margin="10px"
    android:gravity="center">

    <TextView
        android:text="@string/name2"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginRight="5px"
        android:textSize="24sp"/>

    <TextView
        android:id="@+id/text2"
        android:text="0.0"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_width="wrap_content"
        android:textSize="24sp"/>

</LinearLayout>
</LinearLayout>

```

В файл строковых ресурсов добавим строки для отображения трех осей, как показано в листинге 40.13.

Листинг 40.13. Файл строковых ресурсов strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">3D Accelerometer sensor</string>
    <string name="error_msg">This sensor is not available on device</string>
    <string name="name0">X axis:</string>
    <string name="name1">Y axis:</string>
    <string name="name2">Z axis:</string>
</resources>

```

Изменения и дополнения в коде класса `SensorActivity` при использовании акселерометра представлены в листинге 40.14.

Листинг 40.14. Файл класса окна приложения SensorActivity.java

```

public class SensorActivity extends Activity {

    private TextView text0;
    private TextView text1;
    private TextView text2;

    private SensorManager manager;
    private List<Sensor> sensors;

    private SensorEventListener listener = new SensorEventListener() {
        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) { }

        @Override
        public void onSensorChanged(SensorEvent sensorEvent) {
            text0.setText(String.valueOf(sensorEvent.values[0]));
            text1.setText(String.valueOf(sensorEvent.values[1]));
            text2.setText(String.valueOf(sensorEvent.values[2]));
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        text0 = (TextView) findViewById(R.id.text0);
        text1 = (TextView) findViewById(R.id.text1);
        text2 = (TextView) findViewById(R.id.text2);
    }

    @Override
    public void onStart() {
        ...
        sensors = manager.getSensorList(Sensor.TYPE_ACCELEROMETER);
        ...
    }
}

```

Скомпилируйте приложение и запустите его на мобильном устройстве. Внешний вид приложения для работы с акселерометром представлен на рис. 40.7.

Как видно из рисунка, датчик акселерометра показывает значение ускорения свободного падения по оси Z с учетом погрешности измерения, примерно равной земному ускорению свободного падения, а значения по осям X и Y практически равны нулю. Однако датчик довольно чувствителен к перемещениям мобильного устройства и реагирует даже на небольшие перемещения.

Датчик акселерометра довольно часто используют в различных игровых приложениях, когда требуется реакция приложения на перемещение корпуса мобильного устройства пользователем.

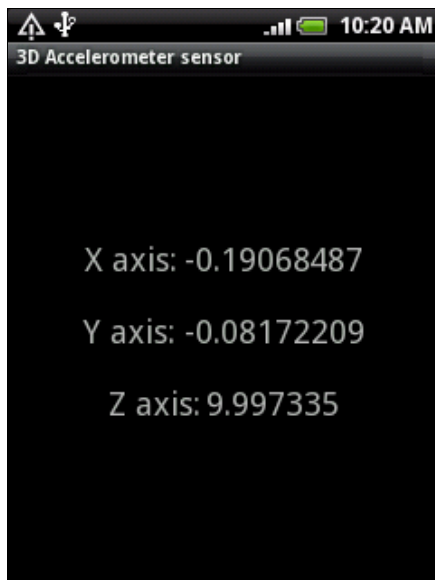


Рис. 40.7. Измерение ускорения по трем осям

Датчик уровня магнитного поля

Датчик уровня магнитного поля выдает данные уровня магнитного поля по трем осям, как и для акселерометра: Lateral, Longitudinal, Vertical. Единицей измерения служит μT (микротесла).

Для оценки уровня магнитного поля в классе `Sensor` определены две константы:

□ `MAGNETIC_FIELD_EARTH_MAX`;

□ `MAGNETIC_FIELD_EARTH_MIN`.

Данные константы определяют соответственно максимальное и минимальное значения уровня магнитного поля на поверхности Земли. Вы можете воспользоваться этими константами, если в приложении вам необходимо определять относительный уровень магнитного поля.

Приложение для чтения параметров с датчика магнитного поля практически идентично приведенному ранее приложению для работы с датчиком акселерометра.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch40_MagneticField3DSensor`.

Файл строковых ресурсов приложения для текстовых полей и текста диалогового окна представлен в листинге 40.15.

Листинг 40.15. Файл строковых ресурсов `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">3D Magnetic field sensor</string>
    <string name="error_msg">This sensor is not available on device</string>
```

```
<string name="name0">X axis:</string>
<string name="name1">Y axis:</string>
<string name="name2">Z axis:</string>
</resources>
```

В классе окна приложения измените параметр в методе `getSensorList()`, как показано в листинге 40.16.

Листинг 40.16. Изменения в классе `SensorActivity`

```
...
@Override
public void onStart() {
    ...
    sensors = manager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
    ...
}
```

Скомпилируйте и запустите приложение на мобильном устройстве. Датчик магнитного поля, как правило, очень чувствительный. Если мобильный телефон поднести к корпусу настольного компьютера или ноутбука, которые являются источниками магнитных полей, приложение отобразит значение магнитного поля, которое будет изменяться в зависимости от расстояния от мобильного устройства до корпуса компьютера и от их взаимного расположения. Работа приложения, использующего датчик магнитного поля, показана на рис. 40.8.

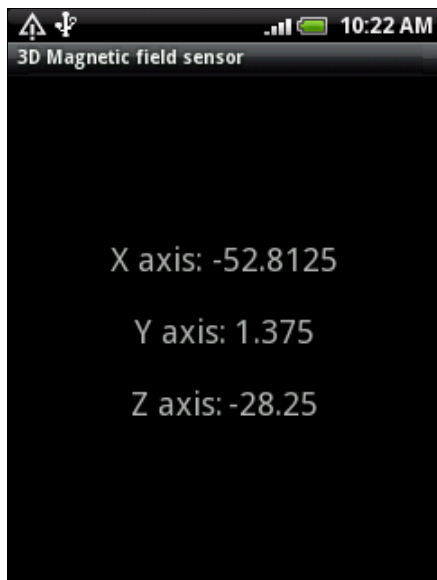


Рис. 40.8. Измерение уровня магнитного поля

Другие датчики, доступные на мобильных устройствах Android

Различные производители мобильных телефонов предоставляют и другие типы датчиков, которые значительно расширяют возможности мобильного устройства для взаимодействия с окружающей средой: датчик гравитации, гироскопический датчик, линейный акселерометр, датчик давления, датчик ротации, датчик температуры.

Работа с этими датчиками аналогична примерам, уже рассмотренным нами ранее в данной главе, и при создании приложений для работы с ними вы можете использовать такие же принципы, как и в приведенных здесь программах.

Имитация работы сенсоров для эмулятора Android

В заключение хотелось бы остановиться на тестировании приложений для работы со встроенными датчиками. Как вы смогли убедиться в этой главе, эмулятор не обеспечивает симуляцию встроенных датчиков. Для тестирования приложений, работающих со встроенными датчиками, необходимо использовать реальное мобильное устройство.

Однако есть сайт **openintents.org**, где можно найти различные приложения для Android. В частности, там предлагается довольно интересный имитатор встроенных датчиков устройств Android — инструмент Sensor Simulator. Этот инструмент также доступен по адресу:

<http://code.google.com/p/openintents/wiki/SensorSimulator>

Этот имитатор моделирует работу акселерометра, компаса и датчика температуры и передает данные на эмулятор Android. Симулятор схематично отображает корпус мобильного телефона (в левом верхнем углу) и позволяет изменять его положение в окне программы с помощью мыши, как показано на рис. 40.9.

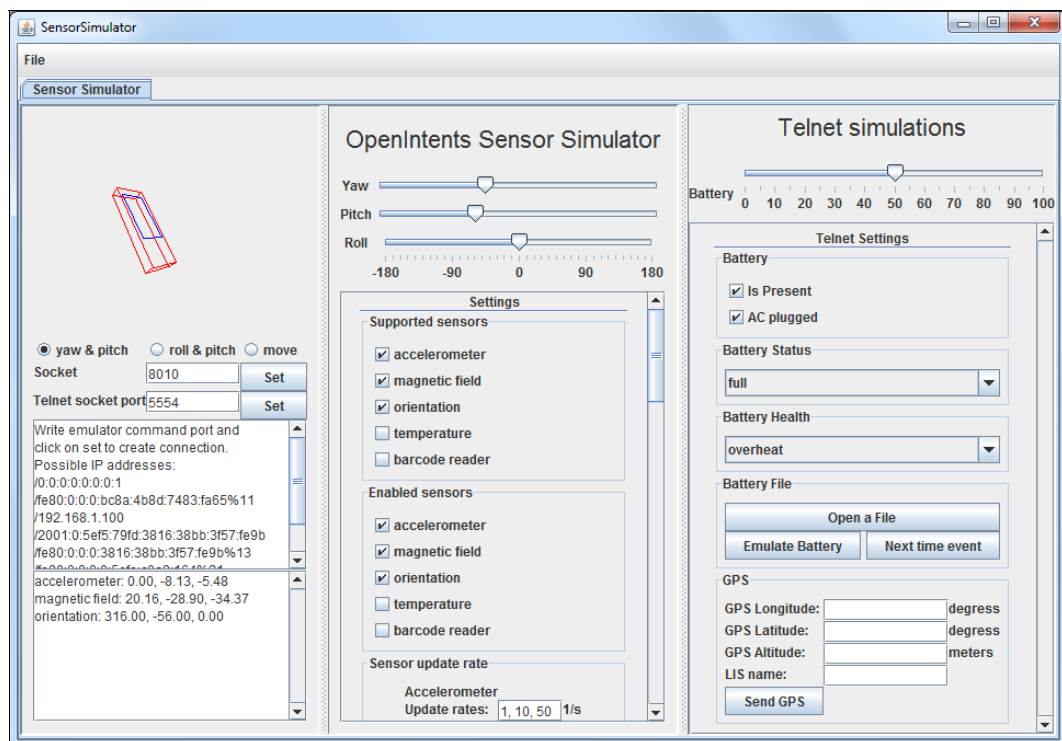


Рис. 40.9. Инструмент Sensor Simulator от OpenIntents

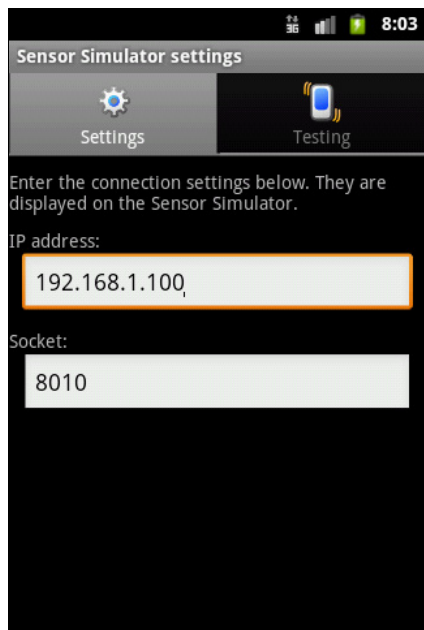


Рис. 40.10. Настройка IP-адреса и номера порта имитатора

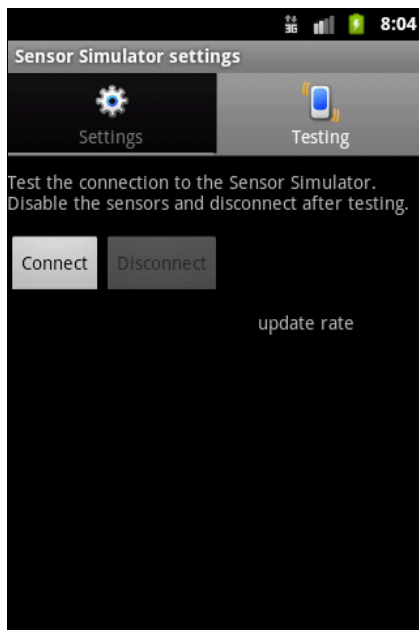


Рис. 40.11. Тестирование соединения с имитатором

Клиентское приложение **Sensor Simulator**, устанавливаемое на эмуляторе Android, представляет собой окно с двумя вкладками. На вкладке **Settings** требуется задать IP-адрес и номер порта, как показано на рис. 40.10. Значения IP-адреса и номера порта надо взять из окна **Sensor Simulator** на левой панели (см. рис. 40.9).

Затем перейдите на вкладку **Testing** приложения **Sensor Simulator** на эмуляторе Android и протестируйте соединение, нажав кнопку **Connect** (рис. 40.11). Теперь инструмент **Sensor Simulator** соединен с эмулятором Android, и его можно использовать для тестирования приложений для работы с датчиками.

Конечно, этот имитатор работы встроенных датчиков не позволяет полностью симулировать работу всех доступных сенсоров, однако те из читателей, которых заинтересовал данный инструмент, при желании могут познакомиться с ним поближе и использовать его при тестировании своих приложений.

Резюме

В этой главе вы получили представление об использовании в приложениях встроенных датчиков Android и возможностях, предоставляемых этими датчиками. Использование датчиков позволяет расширить возможности приложений, которые вы будете создавать для платформы Android, особенно если вы занимаетесь разработкой игр.

Кроме того, как вы убедились при создании приложений для этой главы, мобильное устройство Android можно использовать в качестве инструмента для измерения различных физических величин, что открывает новые возможности для создания интересных приложений с широким применением в различных предметных областях, даже напрямую не связанных со стандартными функциями обычного мобильного устройства.



ГЛАВА 41

Управление дисплеем

В этой главе рассматривается получение и использование информации о дисплее мобильного устройства — получение метрик дисплея. Поскольку существует множество моделей мобильных телефонов с разными типами дисплеев, знание приложением метрик дисплея важно для адаптации приложения к конкретной модели мобильного устройства.

Также мы рассмотрим возможность управления яркостью дисплея из программного кода. Функциональность для управления яркостью дисплея можно применить, например, для экономного использования батареи при ее разрядке.

Программный доступ к дисплею

Платформа Android имеет системную службу Window Service. С помощью этой службы можно управлять яркостью экрана, а также получать информацию о характеристиках экрана, используемого на мобильном устройстве. Для управления этой службой в программном коде используется менеджер окон WindowManager из пакета `android.view`.

Менеджер окон

Для использования менеджера окон WindowManager в коде приложения необходимо сделать обычный вызов метода `getSystemService()`, передав ему в качестве входного параметра значение `Context.WINDOW_SERVICE`:

```
WindowManager manager =  
    (WindowManager) getSystemService(Context.WINDOW_SERVICE);
```

Кстати, в отличие от большинства системных менеджеров, объект WindowManager является не классом, а интерфейсом. Интерфейс WindowManager содержит метод `getDefaultDisplay()`, возвращающий объект класса Display, который инкапсулирует свойства установленного на данном мобильном устройстве дисплея.

Параметры дисплея мобильного устройства

Класс Display содержит набор методов, позволяющих определить основные технические параметры дисплея:

- ❑ `getDisplayId()` — возвращает идентификатор дисплея;
- ❑ `getRefreshRate()` — возвращает значение частоты обновления изображения экрана, измеряемое количеством кадров в секунду;
- ❑ `getHeight()` — возвращает высоту дисплея в пикселах;
- ❑ `getWidth()` — возвращает ширину дисплея в пикселах.

Объект `Display` также позволяет определять текущую ориентацию экрана мобильного устройства. Для этого используется метод `getRotation()`, который возвращает значение угла поворота для экрана мобильного устройства. Значение угла поворота определяется константами, объявленными в классе `Surface`:

- ❑ `ROTATION_0`;
- ❑ `ROTATION_90`;
- ❑ `ROTATION_180`;
- ❑ `ROTATION_270`.

Если мобильное устройство в силу своей конструкции имеет экран с вертикальной ориентацией, а пользователь переворачивает его в горизонтальное положение, возвращаемое значение может быть `ROTATION_90` или `ROTATION_270`, в зависимости от направления вращения телефона.

Для получения метрик дисплея используется вызов метода `getMetrics()` класса `Display`. В этот метод передается Out-параметр типа `DisplayMetrics`. Объект `DisplayMetrics` определяет метрики дисплея — общую информацию о размерах дисплея, плотности и количестве пикселей.

Класс `DisplayMetrics` содержит набор открытых полей с метриками дисплея, например:

- ❑ `heightPixels` — абсолютная высота дисплея в пикселах;
- ❑ `widthPixels` — абсолютная ширина дисплея в пикселах;
- ❑ `scaledDensity` — фактор масштабирования для шрифтов, отображаемых на дисплее;
- ❑ `xdpi` — точное физическое количество пикселей на дюйм экрана по оси X (по ширине экрана);
- ❑ `ydpi` — точное физическое количество пикселей на дюйм экрана по оси Y (по высоте экрана);
- ❑ `densityDpi` — плотность пикселей на дисплее в dpi (dots-per-inch, количество пикселей на один дюйм).

Поле `densityDpi`, в отличие от полей `xdpi` и `ydpi`, возвращает не точную физическую плотность пикселей, а обобщенную величину плотности пикселей на экране, которая используется для качественной оценки свойств дисплея. Эта величина характеризуется четырьмя константами, представляющими типы дисплеев, которые устанавливаются на мобильных устройствах:

- ❑ `DENSITY_LOW` — низкая плотность пикселей;
- ❑ `DENSITY_MEDIUM` — средняя плотность пикселей;
- ❑ `DENSITY_HIGH` — высокая плотность пикселей;
- ❑ `DENSITY_XHIGH` — сверхвысокая плотность пикселей.

Для некоторых типов приложений, особенно тех, которые используют графику, размер и возможности дисплея очень актуальны для корректной и качественной работы приложения. Для таких приложений при запуске на мобильном устройстве желательно обеспечить проверку возможностей дисплея мобильного устройства на соответствие требованиям, предъявляемым приложением.

Получить в коде приложения метрики дисплея мобильного устройства можно, например, следующим образом:

```
WindowManager manager =  
    (WindowManager) getSystemService(Context.WINDOW_SERVICE);  
  
Display display = manager.getDefaultDisplay();  
DisplayMetrics metrics = new DisplayMetrics();  
display.getMetrics(metrics);
```

Сейчас мы рассмотрим практический пример, организовав простое приложение, читающее метрики дисплея. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `Window_DisplayInfo`;
- ☐ **Application name** — `Window manager info`;
- ☐ **Package name** — `com.samples.windowmanagerinfo`;
- ☐ **Create Activity** — `WindowInfoActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch41_Window_DisplayInfo`.

Файл компоновки `main.xml` состоит из одного текстового поля с идентификатором `text`, в который будет выводиться полученная информация.

В коде класса `WindowInfoActivity` в методе `onCreate()` необходимо будет получить объект `WindowManager`, с помощью которого мы можем прочесть метрики дисплея мобильного устройства. Код класса главного окна приложения представлен в листинге 41.1.

Листинг 41.1. Файл класса главного окна приложения `WindowInfoActivity.java`

```
package com.samples.windowmanagerinfo;  
  
import android.app.Activity;  
import android.content.Context;  
import android.os.Bundle;  
import android.util.DisplayMetrics;  
import android.view.Display;  
import android.view.WindowManager;  
import android.widget.TextView;  
  
public class WindowInfoActivity extends Activity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.main);

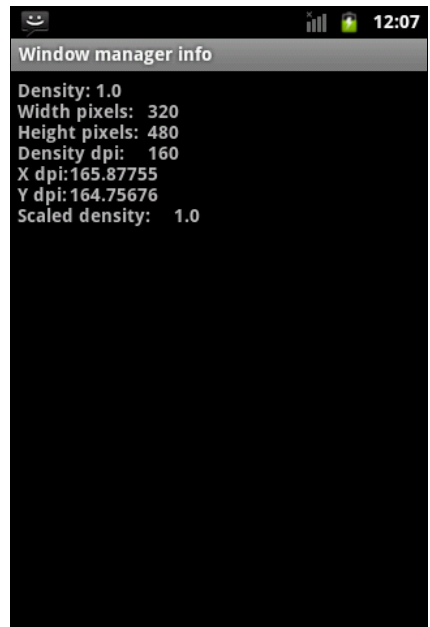
TextView text = (TextView)findViewById(R.id.text);
WindowManager manager =
    (WindowManager)getSystemService(Context.WINDOW_SERVICE);

Display display = manager.getDefaultDisplay();
DisplayMetrics metrics = new DisplayMetrics();
display.getMetrics(metrics);

text.append("Density:\t" + metrics.density);
text.append("\nWidth pixels:\t" + metrics.widthPixels);
text.append("\nHeight pixels:\t" + metrics.heightPixels);
text.append("\nDensity dpi:\t" + metrics.densityDpi);
text.append("\nX dpi:\t" + metrics.xdpi);
text.append("\nY dpi:\t" + metrics.ydpi);
text.append("\nScaled density:\t" + metrics.scaledDensity);
}
```

Скомпилируйте проект и запустите его на эмуляторе Android или на своем мобильном устройстве. Приложение должно выдать набор параметров, характеризующих свойства дисплея данного мобильного устройства. Внешний вид приложения представлен на рис. 41.1.

Рис. 41.1. Вывод информации о дисплее



Управление яркостью экрана

Особенность управления яркостью экрана в системе Android заключается в том, что можно устанавливать яркость не для экрана в целом, а для отдельных элементов окна — для кнопок и элементов экранной клавиатуры, а также устанавливать разные

уровни яркости для окон, находящихся на переднем плане (или в фокусе) или на заднем плане.

При работе с мобильным устройством вы, наверняка, обратили внимание на то, что окна, расположенные на заднем плане, имеют минимальную яркость и выглядят почти черными. Это сделано специально для экономии энергии источника питания мобильного устройства.

Класс `WindowManager` содержит вложенный класс `LayoutParams`. Этот класс предназначен для управления режимом отображения на экране различных элементов пользовательского интерфейса системы Android, которые в своей совокупности образуют изображение, его пользователь и видит на экране мобильного устройства.

Класс `LayoutParams` имеет набор открытых полей, методов и констант — флагов для установки различных режимов отображения экранных элементов. Здесь мы не будем подробно изучать этот класс, а рассмотрим только функциональность для управления яркостью экрана. Для этих целей в классе предусмотрено поле `screenBrightness`. Это поле используется в пользовательских настройках мобильного телефона для установки общей яркости экрана и имеет тип `float`, изменяющийся в диапазоне от 0 (полностью темный экран) до 1 (максимальная яркость).

Значение яркости экрана, установленное в пользовательских настройках мобильного телефона, можно получить с помощью константы `Settings.System.SCREEN_BRIGHTNESS` и метода `Settings.System.getInt()` следующим образом:

```
int brightness = android.provider.Settings.System.getInt(  
    getContentResolver(),  
    android.provider.Settings.System.SCREEN_BRIGHTNESS);
```

Сохранение настроек осуществляется методом `Settings.System.putInt()`, которому мы передаем в качестве параметра новое значение яркости:

```
android.provider.Settings.System.putInt(getContentResolver(),  
    android.provider.Settings.System.SCREEN_BRIGHTNESS, brightness);
```

Обратите внимание, что значение яркости, получаемое из системных пользовательских настроек, является целочисленным типом и изменяется в диапазоне от 0 (полностью темный экран) до 255 (максимальная яркость), а значение для поля `screenBrightness` имеет тип `float` и изменяется в диапазоне от 0 до 1, поэтому не забывайте в программном коде делать соответствующие преобразования при чтении и записи пользовательских настроек и при установке яркости экрана.

Чтобы получить доступ к полю `screenBrightness` класса `LayoutParams` для установки текущего уровня яркости экрана, сначала надо получить объект `WindowManager.LayoutParams`. Для этого, используя метод `getWindow()` класса `Activity`, получаем объект `Window`, представляющий текущее окно, а затем через метод `getAttributes()` этого класса мы можем получить экземпляр `WindowManager.LayoutParams`. Далее можно присвоить полю `screenBrightness` новое значение яркости и изменить яркость окна, используя метод `setAttributes()`:

```
WindowManager.LayoutParams params = getWindow().getAttributes();  
params.screenBrightness = 0.7;  
getWindow().setAttributes(params);
```

Сейчас мы попробуем реализовать приложение, управляющее яркостью экрана и сохраняющее значение яркости в системных пользовательских настройках. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — Window_Brightness;
- ☐ **Application name** — Set Screen Brightness;
- ☐ **Package name** — com.samples.window.setscreenbright;
- ☐ **Create Activity** — ScreenBrightnessActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch41_Window_Brightness.

В файл манифеста приложения AndroidManifest.xml добавьте разрешение android.permission.WRITE_SETTINGS. Это разрешение необходимо для сохранения настроек в системе, в данном случае — уровня яркости дисплея. Код файла манифеста приложения AndroidManifest.xml представлен в листинге 41.2.

Листинг 41.2. Файл манифеста приложения AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.window.setscreenbright"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".ScreenBrightnessActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

    <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
</manifest>
```

В файле компоновки главного окна приложения main.xml разместим три элемента управления:

- ☐ ползунок SeekBar с идентификатором seek для регулировки яркости экрана;
- ☐ текстовое поле TextView с идентификатором text;
- ☐ кнопку Button с идентификатором bSetBright.

Файл компоновки окна представлен в листинге 41.3.

Листинг 41.3. Файл компоновки окна приложения main.xml

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

    <SeekBar
        android:id="@+id/seek"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10px"
        android:max="100"
        android:progress="50"/>

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/text"
        android:text="0.50"/>

    <Button
        android:id="@+id/bSetBright"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Set Brightness Preference"
        android:onClick="onClick"/>

</LinearLayout>
```

В коде класса `ScreenBrightnessActivity` главного окна приложения реализуем код, приведенный ранее в этом разделе, для получения системных настроек яркости экрана, изменения яркости и сохранения нового значения в системных настройках мобильного телефона.

Код класса `ScreenBrightnessActivity` представлен в листинге 41.4.

Листинг 41.4. Файл класса окна приложения ScreenBrightnessActivity.java

```
package com.samples.window.setscreenbright;

import android.app.Activity;
import android.os.Bundle;
import android.provider.Settings.SettingNotFoundException;
import android.view.View;
import android.view.WindowManager;
import android.widget.SeekBar;
import android.widget.TextView;

public class ScreenBrightnessActivity extends Activity
implements OnClickListener, OnSeekBarChangeListener {
```

```

private SeekBar seek;
private Button bSetBright;
private TextView text;

private int brightness = 128;

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

    seek = (SeekBar)findViewById(R.id.seek);
    text = (TextView)findViewById(R.id.text);

    try {
        // Получаем текущее значение яркости экрана
        // из системных настроек (диапазон значений 0..255)
        brightness = android.provider.Settings.System.getInt(
            getContentResolver(),
            android.provider.Settings.System.SCREEN_BRIGHTNESS);
    }
    catch (SettingNotFoundException e) {
        e.printStackTrace();
    }

    // Устанавливаем ползунок в текущее значение яркости экрана
    int percBrigh = brightness * 100 / 255;
    seek.setProgress(percBrigh);

    // Выводим в текстовое поле значение яркости экрана в %
    text.setText(percBrigh + "%");

    seek.setOnSeekBarChangeListener(this);
}

@Override
public void onProgressChanged(SeekBar arg0, int arg1, boolean arg2) {
    text.setText(arg1 + "%");

    // Получаем параметр, определяющий яркость экрана
    WindowManager.LayoutParams params = getWindow().getAttributes();

    // Устанавливаем значение поля screenBrightness,
    // преобразуя значение яркости в % (0...100) в диапазон
    // значений поля screenBrightness (0...1)
    params.screenBrightness = (float)arg1 / 100;

    // Устанавливаем яркость экрана
    getWindow().setAttributes(params);
    brightness = arg1 * 255 / 100;
}

```



```
@Override
public void onStartTrackingTouch(SeekBar arg0) { }

@Override
public void onStopTrackingTouch(SeekBar arg0) { }

@Override
public void onClick(View arg0) {
    // Сохраняем значение яркости экрана в системных настройках
    android.provider.Settings.System.putInt(
        getContentResolver(),
        android.provider.Settings.System.SCREEN_BRIGHTNESS,
        brightness);
}
```

Скомпилируйте проект и запустите его на эмуляторе Android или на своем мобильном устройстве. Приложение позволяет регулировать яркость окна и, если требуется сохранить новое значение яркости экрана в системных пользовательских настройках, можно использовать кнопку **Set Brightness Preference**. Внешний вид приложения представлен на рис. 41.2.

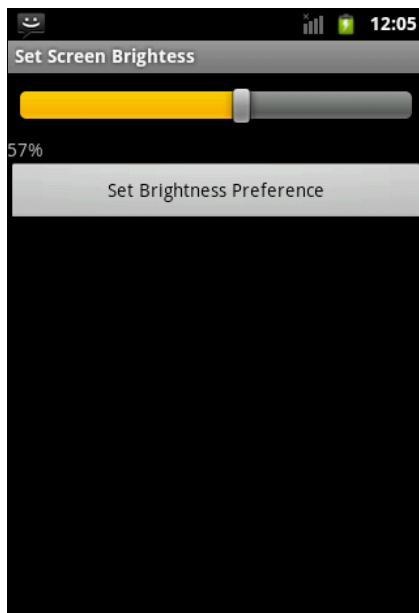


Рис. 41.2. Приложение, управляющее яркостью дисплея

Резюме

В этой главе мы изучили возможности использования службы Window Service в своих приложениях. Эту службу можно применять для получения сведений о дисплее мобильного телефона, на который установлено приложение. Получение характеристик дисплея мобильного устройства актуально для корректной работы приложений, использующих графику.

Также мы рассмотрели функциональность, предоставляемую службой Window Service для управления яркостью экрана. Эту возможность можно использовать в приложениях для обеспечения эффективного энергопотребления мобильного устройства.

Следующая глава будет посвящена вопросу управления энергопотреблением телефона и использованию соответствующих системных служб Android.



ГЛАВА 42

Доступ к аккумуляторной батарее

В этой главе рассматривается использование менеджера источника питания. Этот менеджер можно использовать для контроля состояния батареи в приложениях — можно отслеживать уровень заряда батареи, ее техническое состояние, температуру и другие параметры.

Менеджер источника питания

Все данные об аккумуляторной батарее мобильного устройства инкапсулированы в классе `BatteryManager`. В отличие от других менеджеров, рассмотренных в книге, экземпляр этого класса нельзя получить через вызов метода `getSystemService()`, поскольку службы, управляющей батареей мобильного устройства, не существует. Для оповещения об изменениях состояния аккумуляторной батареи система Android рассылает объекты `Broadcast Intent` всем компонентам, находящимся на мобильном устройстве.

Следовательно, для получения информации о состоянии батареи надо в приложении использовать объект `BroadcastReceiver`, создав для него фильтр со значением `Intent.ACTION_BATTERY_CHANGED`. Кстати, этот фильтр нельзя задекларировать в манифесте приложения, его можно только зарегистрировать с помощью метода `registerReceiver()` класса `Context`.

Объект `Intent`, который система передает в качестве входного параметра в метод `onReceive()` объекта `BroadcastReceiver`, зарегистрированного в приложении, имеет набор Extra-параметров, которые являются полями класса `BatteryManager`. Эти Extra-параметры предоставляют информацию о технических характеристиках и состоянии аккумуляторной батареи:

- ❑ `EXTRA_SCALE` — максимальный уровень заряда батареи;
- ❑ `EXTRA_LEVEL` — текущий уровень заряда батареи, его величина может изменяться от 0 до `EXTRA_SCALE`. Обычно значения `EXTRA_LEVEL` и `EXTRA_SCALE` используют, чтобы отобразить текущий уровень заряда батареи в процентах относительно полного заряда;
- ❑ `EXTRA_TEMPERATURE` — температура батареи;
- ❑ `EXTRA_VOLTAGE` — уровень напряжения на батарее;
- ❑ `EXTRA_PRESENT` — значение типа `boolean`, которое показывает наличие батареи на мобильном устройстве;

- ❑ `EXTRA_TECHNOLOGY` — тип технологии этой батареи (например, "Li-Ion", если это литиевая аккумуляторная батарея);
- ❑ `EXTRA_ICON_SMALL` — число, которое содержит идентификатор ресурса значка батареи. Этот значок обычно отображается в строке состояния мобильного телефона и показывает текущий уровень заряда батареи. Значок уровня заряда батареи также можно при необходимости вывести в окне своего приложения, используя этот параметр для доступа к ресурсу.

Также в объекте `Intent` есть три Extra-параметра, характеризующие состояние батареи: `EXTRA_HEALTH`, `EXTRA_STATUS` и `EXTRA_PLUGGED`.

Параметр `EXTRA_HEALTH` характеризует техническое состояние батареи. Этот параметр может принимать одно из следующих значений:

- ❑ `BATTERY_HEALTH_GOOD` — батарея в хорошем состоянии;
- ❑ `BATTERY_HEALTH_OVER_VOLTAGE` — у батареи повышенное напряжение (например, произошла перезарядка батареи);
- ❑ `BATTERY_HEALTH_OVERHEAT` — батарея перегрета, имеет повышенную температуру (такое тоже часто бывает при перезарядке батареи);
- ❑ `BATTERY_HEALTH_UNSPECIFIED_FAILURE` — батарея неисправна из-за различных причин;
- ❑ `BATTERY_HEALTH_DEAD` — батарея полностью неработоспособна;
- ❑ `BATTERY_HEALTH_UNKNOWN` — состояние батареи неизвестно.

Параметр `EXTRA_STATUS` характеризует текущее состояние заряда аккумуляторной батареи мобильного телефона. Это состояние определяется константами, показывающими уровень заряда батареи:

- ❑ `BATTERY_STATUS_CHARGING` — батарея в данный момент заряжается;
- ❑ `BATTERY_STATUS_FULL` — батарея полностью заряжена;
- ❑ `BATTERY_STATUS_NOT_CHARGING` — батарея не заряжается;
- ❑ `BATTERY_STATUS_DISCHARGING` — батарея разряжена;
- ❑ `BATTERY_STATUS_UNKNOWN` — статус батареи неизвестен.

Параметр `EXTRA_PLUGGED` определяет подключение зарядного устройства к мобильному телефону:

- ❑ `BATTERY_PLUGGED_AC` — через сетевое зарядное устройство;
- ❑ `BATTERY_PLUGGED_USB` — зарядка через USB.

Для отслеживания состояния аккумуляторной батареи необходимо, прежде всего, создать объект `BroadcastReceiver`:

```
BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        // Читаем Extra-параметры из объекта Intent
        ...
    }
}
```

Созданный объект `BroadcastReceiver` надо зарегистрировать с помощью метода `registerReceiver()`, определив для него фильтр, чтобы при отслеживании событий, происходящих в системе, приложение "ловило" только объекты `Intent`, генерируемые системой при изменении состояния батареи:

```
registerReceiver(receiver,  
    new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
```

Для остановки отслеживания состояния аккумуляторной батареи используют вызов метода `unregisterReceiver()`:

```
unregisterReceiver(receiver);
```

При использовании `BatteryManager` в приложении требуется обязательное объявление разрешения `android.permission.BATTERY_STATS`.

Если в приложении не нужна полная информация о состоянии и характеристиках батареи, например, требуется только получать извещение о зарядке или разрядке батареи, можно использовать более узкоспециализированные действия:

- ❑ `ACTION_BATTERY_LOW` — генерируется при разрядке батареи. При появлении этого Broadcast Intent на экране мобильного устройства отображается системное диалоговое окно **Low battery warning**;
- ❑ `ACTION_BATTERY_OKAY` — генерируется при полной зарядке батареи. Этот Broadcast Intent, в частности, используется для изменения цвета светодиодного индикатора зарядки с красного на зеленый;
- ❑ `ACTION_POWER_CONNECTED` — генерируется при подключении внешнего зарядного устройства к мобильному телефону;
- ❑ `ACTION_POWER_DISCONNECTED` — генерируется при отключении внешнего зарядного устройства от мобильного телефона.

Сейчас мы напишем практическое приложение, которое может отслеживать состояние источника питания мобильного устройства. Для этого создайте новый проект Android в IDE Eclipse и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `BatteryManagerInfo`;
- ❑ **Application name** — `Battery Manager`;
- ❑ **Package name** — `com.samples.hardware.batterymanager`;
- ❑ **Create Activity** — `BatteryManagerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch42_BatteryManagerInfo`.

В файл манифеста приложения `AndroidManifest.xml` не забудьте добавить разрешение `android.permission.BATTERY_STATS`. Код файла `AndroidManifest.xml` представлен в листинге 42.1.

Листинг 42.1. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
```

```

package="com.samples.hardware.batterymanager"
android:versionCode="1"
android:versionName="1.0">
<application android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity android:name=".BatteryManagerActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

<uses-permission
    android:name="android.permission.BATTERY_STATS" />
<uses-sdk android:minSdkVersion="10" />
</manifest>

```

В файле компоновки главного окна приложения `main.xml` будут располагаться две кнопки `Button` с идентификаторами `bStart`, `bStop` и надписями **Start** и **Stop** для запуска и останова отслеживания состояния батареи, а также текстовое поле `TextView` с идентификатором `text` для вывода результатов.

Файл компоновки окна представлен в листинге 42.2.

Листинг 42.2. Файл компоновки окна приложения `main.xml`

```

<?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">

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="fill_parent">

        <Button
            android:id="@+id/bStart"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:text="Start"
            android:onClick="onClick"
            android:layout_weight="1"
            android:layout_margin="5px"/>

        <Button
            android:id="@+id/bStop"

```

```

        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="Stop"
        android:onClick="onClick"
        android:layout_weight="1"
        android:layout_margin="5px"/>
</LinearLayout>

<TextView
    android:id="@+id/text"
    android:text=""
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textStyle="bold"
    android:layout_margin="10px"/>
</LinearLayout>

```

В коде класса главного окна приложения для кнопок `bStart` и `bStop` будет определен обработчик события `onClick()`, где в зависимости от нажатой кнопки будет происходить подключение или отключение `BroadcastReceiver` для отслеживания изменений состояния батареи.

Код класса главного окна приложения представлен в листинге 42.3.

Листинг 42.3. Файл класса окна приложения `BatteryManagerActivity.java`

```

package com.samples.hardware.batterymanager;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.BatteryManager;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class BatteryManagerActivity extends Activity
    implements View.OnClickListener {

    private Button bStart;
    private Button bStop;
    private BroadcastReceiver receiver;
    private TextView text;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

```

```

setContentView(R.layout.main);
text = (TextView)findViewById(R.id.text);
InitReceiver();
}

private void InitReceiver()
{
    receiver = new BroadcastReceiver() {

        @Override
        public void onReceive(Context context, Intent intent) {
            int level = intent.getIntExtra(
                BatteryManager.EXTRA_LEVEL, -1);
            int scale = intent.getIntExtra(
                BatteryManager.EXTRA_SCALE, -1);
            int status = intent.getIntExtra(
                BatteryManager.EXTRA_STATUS, -1);
            int healt = intent.getIntExtra(
                BatteryManager.EXTRA_HEALTH, -1);
            int plugged = intent.getIntExtra(
                BatteryManager.EXTRA_PLUGGED, -1);
            String technology = intent.getStringExtra(
                BatteryManager.EXTRA_TECHNOLOGY);
            int icon = intent.getIntExtra(
                BatteryManager.EXTRA_ICON_SMALL, -1);
            float voltage = (float)intent.getIntExtra(
                BatteryManager.EXTRA_VOLTAGE, -1) / 1000;
            boolean present = intent.getBooleanExtra(
                BatteryManager.EXTRA_PRESENT, false);
            float temperature = (float)intent.getIntExtra(
                BatteryManager.EXTRA_TEMPERATURE, -1) / 10;

            // Общее состояние батареи
            String shealth = "Not reported";
            switch (healt) {
                case BatteryManager.BATTERY_HEALTH_DEAD:
                    shealth = "Dead";
                    break;
                case BatteryManager.BATTERY_HEALTH_GOOD:
                    shealth = "Good";
                    break;
                case BatteryManager.BATTERY_HEALTH_OVER_VOLTAGE:
                    shealth = "Over voltage";
                    break;
                case BatteryManager.BATTERY_HEALTH_OVERHEAT:
                    shealth = "Over heating";
                    break;
                case BatteryManager.BATTERY_HEALTH_UNKNOWN:
                    shealth = "Unknown";
                    break;
            }
        }
    };
}

```



```
case BatteryManager.BATTERY_HEALTH_UNSPECIFIED_FAILURE:
    shealth = "Unspecified failure";
    break;
}

// Состояние зарядки батареи
String sStatus = "Not reported";
switch (status) {
case BatteryManager.BATTERY_STATUS_CHARGING:
    sStatus = "Charging";
    break;
case BatteryManager.BATTERY_STATUS_DISCHARGING:
    sStatus = "Discharging";
    break;
case BatteryManager.BATTERY_STATUS_FULL:
    sStatus = "Full";
    break;
case BatteryManager.BATTERY_STATUS_NOT_CHARGING:
    sStatus = "Not Charging";
    break;
case BatteryManager.BATTERY_STATUS_UNKNOWN:
    sStatus = "Unknown";
    break;
}

// Тип зарядки
String splugged = "Not Reported";
switch (plugged) {
case BatteryManager.BATTERY_PLUGGED_AC:
    splugged = "On AC";
    break;
case BatteryManager.BATTERY_PLUGGED_USB:
    splugged = "On USB";
    break;
}

int chargedPct = (level * 100) / scale;

String batteryInfo = "Battery Info:" +
    "\nHealth: " + shealth +
    "\nStatus: " + sStatus +
    "\nCharged: " + chargedPct + "%" +
    "\nPlugged: " + splugged +
    "\nVoltage: " + voltage +
    "\nTechnology: " + technology +
    "\nTemperature: " + temperature + "C" +
    "\nBattery present: " + present + "\n";

text.setText(batteryInfo);
}
```

```
};
```

```
}
```

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.bStart:
            // Запускаем мониторинг состояния батареи
            registerReceiver(receiver,
                new IntentFilter(Intent.ACTION_BATTERY_CHANGED));

            text.setText("Start phone info listener...");
            break;

        case R.id.bStop:
            // Останавливаем мониторинг состояния батареи
            unregisterReceiver(receiver);
            text.setText("Listener is stopped");
            break;
    }
}

@Override
protected void onPause() {
    if (receiver != null) {
        unregisterReceiver(receiver);
        receiver = null;
    }

    super.onPause();
}
}
```

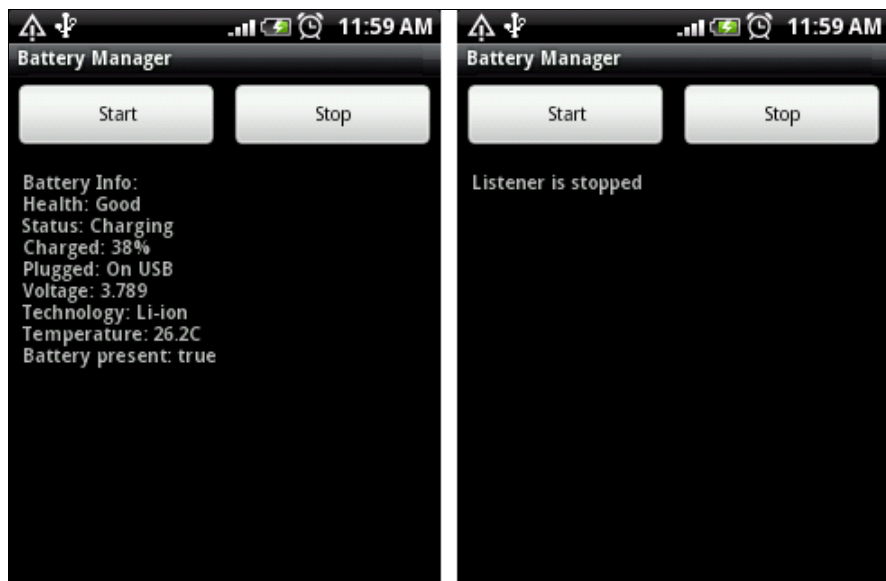


Рис. 42.1. Получение информации о состоянии батареи

Скомпилируйте проект и запустите его на вашем мобильном устройстве. При нажатии кнопки **Start** запустится мониторинг состояния батареи и приложение выдаст информацию о батарее. При изменении какого-либо параметра информация в окне будет обновляться (например, заряд батареи, если в данный момент устройство стоит на зарядке). Внешний вид приложения представлен на рис. 42.1.

Отображение статистики использования батареи

Если вам требуется в приложении вывести суммарную статистику использования батареи на устройстве, можно вызвать стандартный Activity, который отображает данную информацию.

Для этого нужно создать объект Intent с параметром ACTION_POWER_USAGE_SUMMARY, например, следующим образом:

```
Intent intent = new Intent(Intent.ACTION_POWER_USAGE_SUMMARY);
startActivity(intent);
```

Чтобы посмотреть Activity для отображения статистики использования батареи в действии, сделаем простое приложение. Для этого создайте в IDE Eclipse новый проект Android и в диалоговом окне **New Android Project** заполните поля следующими значениями:

- ☐ **Project name** — BatteryUsageSummary;
- ☐ **Application name** — Battery Usage;
- ☐ **Package name** — com.samples.hardware.batteryusage;
- ☐ **Create Activity** — BatteryUsageActivity.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге Ch42_BatteryUsageSummary.

В файле компоновки главного окна приложения main.xml будет расположена единственная кнопка с надписью **Battery Usage** и идентификатором bBatteryUsage для открытия стандартного Activity, отображающего состояние батареи. Файл компоновки окна представлен в листинге 42.4.

Листинг 42.4. Файл компоновки окна приложения main.xml

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

    <Button
        android:text="Battery Usage"
        android:id="@+id/bBatteryUsage"
        android:layout_height="wrap_content"
```

```

    android:layout_width="fill_parent"
    android:onClick="onClick"
    android:layout_margin="5px"/>

```

```
</LinearLayout>
```

В коде класса главного окна приложения создайте обработчик события `onClick()` для кнопки, в котором будет создаваться объект `Intent` для вызова внешнего `Activity`. Код класса главного окна приложения представлен в листинге 42.5.

Листинг 42.5. Файл класса окна приложения `BatteryUsageActivity.java`

```

package com.samples.hardware.batteryusage;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class BatteryUsageActivity extends Activity
    implements OnClickListener {

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

    @Override
    public void onClick(View arg0) {
        // Открываем стандартный Activity с информацией о батарее
        Intent intent = new Intent(Intent.ACTION_POWER_USAGE_SUMMARY);
        startActivity(intent);
    }
}

```

Скомпилируйте проект и запустите его на своем мобильном телефоне. При нажатии на кнопку **Battery Usage** должно открыться окно, отображающее суммарную статистику использования батареи мобильного устройства (рис. 42.2). Конечно, это окно зависит от модели телефона и может отличаться от представленного на рисунке.

Резюме

Используя менеджер источника питания `Battery Manager`, можно создавать приложения, которые будут отслеживать состояние аккумуляторной батареи. Эту функциональность можно использовать для экономии потребления питания: например, при падении

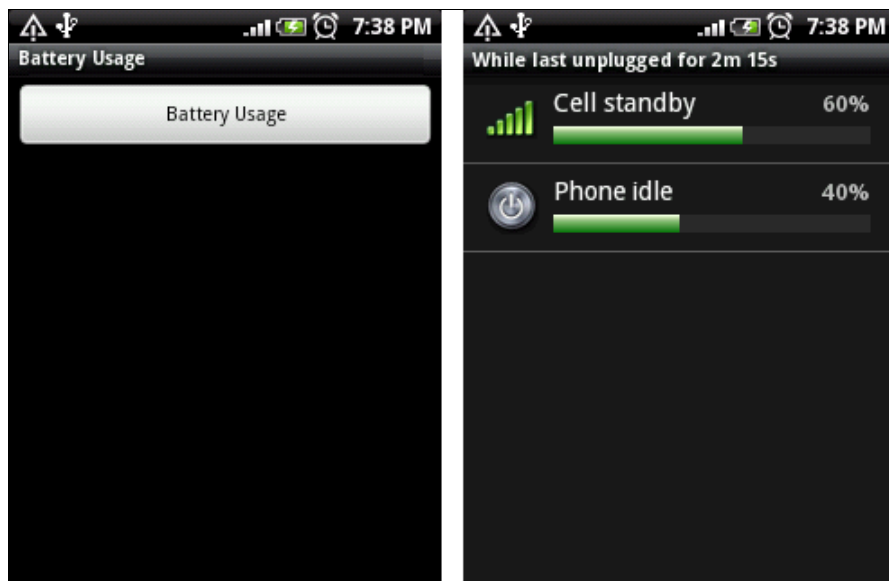


Рис. 42.2. Вызов стандартного Activity, отображающего использование батареи

уровня напряжения меньше определенного предела и невозможности сразу зарядить мобильное устройство, переключать его в экономичный режим для продления работы телефона.

Для управления режимом энергопотребления телефона в системе Android есть специальный менеджер энергопотребления, работу с которым мы рассмотрим в следующей главе.



ГЛАВА 43

Управление энергопотреблением телефона

Эффективное использование энергии очень актуально для всех мобильных телефонов. В этой главе рассматривается использование менеджера энергопотребления для управления службой Power Service.

Менеджер энергопотребления позволяет приложениям использовать специальные блокировки для управления процессором, клавиатурой и дисплеем мобильного устройства. С помощью этих блокировок, управляемых через менеджер, можно, например, ввести дополнительную временную задержку на выключение экрана или подсветки клавиатуры телефона. В случае если у мобильного телефона разряжена батарея и в данный момент невозможно его зарядить, можно перевести его в экономичный режим — сократить время работы дисплея или подсветки клавиатуры для продления работы аккумулятора.

Менеджер энергопотребления

Менеджер энергопотребления является важным компонентом системы Android. С его помощью можно эффективно управлять потреблением энергии от батареи мобильного устройства.

Получить доступ к менеджеру энергопотребления в коде программы можно через метод `getSystemService()` с параметром `Context.POWER_SERVICE`:

```
PowerManager manager =  
    (PowerManager) getSystemService(Context.POWER_SERVICE);
```

Вот основные методы, которые объявлены в классе `PowerManager`:

- ☐ `goToSleep(long time)` — переключает устройство в спящий режим. Входным параметром является время в миллисекундах;
- ☐ `isScreenOn()` — возвращает `true`, если дисплей находится во включенном состоянии;
- ☐ `reboot(String reason)` — производит перезагрузку мобильного устройства. Входным параметром для этого метода является строка, описывающая причину перезагрузки.

Управление энергопотреблением и блокировки

Помимо переключения устройства в спящий режим и перезагрузки, класс `PowerManager` предназначен для создания блокировок (Wake Lock). Блокировки могут управлять ре-

жимом работы процессора, включением и выключением экрана и подсветкой клавиатуры. Длительность работы батареи на мобильном устройстве существенно зависит от использования блокировок менеджера энергопотребления.

Для создания блокировки в классе `PowerManager` определен метод `newWakeLock(int flags, String tag)`, который имеет два входных параметра:

- ❑ флаг, определяющий тип блокировки;
- ❑ строку, идентифицирующую эту блокировку (уникальное в пределах приложения имя, которое присваивается ей при создании).

Флаг определяет режим использования процессора, экрана и клавиатуры мобильного устройства. Значения флага являются взаимоисключающими, т. е. при создании блокировки можно использовать только один флаг:

- ❑ `PARTIAL_WAKE_LOCK` — частичная блокировка, которая гарантирует, что процессор всегда будет в рабочем состоянии, но экран и подсветка клавиатуры будут выключены;
- ❑ `SCREEN_BRIGHT_WAKE_LOCK` — блокировка, которая гарантирует, что экран телефона будет включен, а подсветка клавиатуры выключена;
- ❑ `SCREEN_DIM_WAKE_LOCK` — блокировка, которая гарантирует, что экран может быть затемненным (но не выключенным), а подсветка клавиатуры будет выключена;
- ❑ `FULL_WAKE_LOCK` — блокировка, при которой экран и подсветка клавиатуры включены.

В режиме `PARTIAL_WAKE_LOCK` процессор мобильного устройства остается в рабочем состоянии даже после нажатия кнопки выключения питания. При остальных значениях флагов блокировки пользователь может выключить свой телефон с помощью кнопки питания.

Кроме того, существуют еще два дополнительных флага, которые влияют на поведение экрана:

- ❑ `ACQUIRE_CAUSES_WAKEUP` — блокировка запускается сразу, вне зависимости от текущей активности пользователя на мобильном устройстве;
- ❑ `ON_AFTER_RELEASE` — блокировка запускается только после освобождения устройства. Например, если пользователь работал с мобильным устройством, сначала отработывает таймер, задающий стандартное время до отключения экрана, затем запускается заданная блокировка. Если при этом был установлен `SCREEN_BRIGHT_WAKE_LOCK`, экран останется включенным после отработки таймером стандартного времени блокировки.

Метод `newWakeLock()` возвращает объект `WakeLock`, который используется для управления созданной блокировкой. В этом классе определен набор методов, позволяющих включать или отключать выбранные режимы:

- ❑ `acquire()` — включает блокировку на мобильном устройстве;
- ❑ `acquire(long timeout)` — включает блокировку на мобильном устройстве на время, заданное входным параметром `timeout`, определяющим время работы блокировки в миллисекундах;
- ❑ `release()` — отключает блокировку на мобильном устройстве;

- ❑ `setReferenceCounted(boolean value)` — запускает или останавливает внутренний счетчик ссылок для данной блокировки. При входном параметре, равным `true`, счетчик ссылок будет включен. В этом случае будут подсчитываться любые вызовы методов `acquire()` и `release()`. Каждый вызов метода `acquire()` будет увеличивать значение счетчика на 1, а вызов метода `release()`, соответственно, будет уменьшать значение в счетчике.

Использование блокировок в коде приложения обязательно требует наличия разрешения `android.permission.WAKE_LOCK` в файле манифеста приложения.

Для реализации в программном коде управления энергопотреблением сначала надо получить объект `PowerManager`, используя стандартный вызов метода `getSystemService()` с параметром `Context.POWER_SERVICE`. Затем необходимо вызвать метод `newWakeLock()` класса `PowerManager`. При этом мы получим объект `WakeLock` и можем использовать его методы `acquire()` и `release()` для управления блокировками и, соответственно, энергопотреблением устройства. Код может выглядеть следующим образом:

```
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
WakeLock wl = pm.newWakeLock(
    PowerManager.SCREEN_BRIGHT_WAKE_LOCK, "Screen Bright Wake Lock");
// Включаем Wake Lock
wl.acquire();
...
// Отключаем Wake Lock
wl.release();
```

Теперь давайте рассмотрим работу с менеджером энергопотребления в приложении, чтобы увидеть, как работают различные варианты блокировок на мобильном устройстве. Для этого создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ❑ **Project name** — `PowerManager`;
- ❑ **Application name** — `Power Manager`;
- ❑ **Package name** — `com.samples.hardware.powermanager`;
- ❑ **Create Activity** — `PowerManagerActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch43_PowerManager`.

В файл манифеста приложения `AndroidManifest.xml` добавьте разрешение `android.permission.WAKE_LOCK`, как показано в листинге 43.1.

Листинг 43.1. Файл манифеста приложения `AndroidManifest.xml`

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

```

<application
    android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".PowerManagerActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

<uses-permission android:name="android.permission.WAKE_LOCK"/>
</manifest>

```

В файле компоновки `main.xml` главного окна приложения будет набор из четырех кнопок для управления работой мобильного устройства и создания блокировок разного типа, для надписей на кнопках используются строковые ресурсы, определенные в файле `strings.xml` из листинга 43.2.

Листинг 43.2. Файл ресурсов `strings.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, PowerManagerActivity!</string>
    <string name="app_name">Power Manager</string>
    <string name="partial_wake_lock">Partial Wake Lock</string>
    <string name="screen_bright_wake_lock">Screen Bright Wake Lock</string>
    <string name="screen_dim_wake_lock">Screen Dim Wake Lock</string>
    <string name="full_wake_lock">Full Wake Lock</string>
</resources>

```

Код файла компоновки `main.xml` представлен в листинге 43.3.

Листинг 43.3. Файл компоновки окна приложения `main.xml`

```

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

    <Button
        android:text="@string/partial_wake_lock"
        android:id="@+id/bPartialWL"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:onClick="onClick"
        android:layout_margin="5px"/>

```

```

<Button
    android:text="@string/screen_bright_wake_lock"
    android:id="@+id/bScreenBrightWL"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:onClick="onClick"
    android:layout_margin="5px"/>
<Button
    android:text="@string/screen_dim_wake_lock"
    android:id="@+id/bScreenDimWL"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:onClick="onClick"
    android:layout_margin="5px"/>
<Button
    android:text="@string/full_wake_lock"
    android:id="@+id/bFullWL"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:onClick="onClick"
    android:layout_margin="5px"/>
</LinearLayout>

```

В коде класса `PowerManagerActivity` главного окна приложения в методе `onCreate()` будет получен экземпляр `PowerManager`. В методе `onClick()`, в зависимости от нажатой кнопки, будет происходить создание блокировки заданного типа.

Код класса `PowerManagerActivity` представлен в листинге 43.4.

Листинг 43.4. Файл класса окна приложения `PowerManagerActivity.java`

```

package com.samples.hardware.powermanager;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.view.View;
import android.widget.Toast;

public class PowerManagerActivity extends Activity
    implements OnClickListener {

    PowerManager manager;
    WakeLock wakeLock;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

```

```

        setContentView(R.layout.main);
        manager = (PowerManager) getSystemService(
            Context.POWER_SERVICE);
    }

    @Override
    public void onClick(View v) {
        // Освобождаем предыдущий Wake Lock,
        // если такой существует
        if (wakeLock != null) {
            wakeLock.release();
        }

        String wlName = "";

        switch (v.getId()) {
            case R.id.bPartialWL:
                wlName = getResources().getString(
                    R.string.partial_wake_lock);
                wakeLock = manager.newWakeLock(
                    PowerManager.PARTIAL_WAKE_LOCK, wlName);
                break;

            case R.id.bScreenBrightWL:
                wlName = getResources().getString(
                    R.string.screen_bright_wake_lock);
                wakeLock = manager.newWakeLock(
                    PowerManager.SCREEN_BRIGHT_WAKE_LOCK, wlName);
                break;

            case R.id.bScreenDimWL:
                wlName = getResources().getString(
                    R.string.screen_dim_wake_lock);
                wakeLock = manager.newWakeLock(
                    PowerManager.SCREEN_DIM_WAKE_LOCK, wlName);
                break;

            case R.id.bFullWL:
                wlName = getResources().getString(
                    R.string.full_wake_lock);
                wakeLock = manager.newWakeLock(
                    PowerManager.FULL_WAKE_LOCK, wlName);
                break;
        }

        Toast.makeText(this, wlName + " ON", Toast.LENGTH_LONG).show();
        wakeLock.acquire();
    }
}

```

Скомпилируйте и запустите приложение на своем мобильном устройстве. Внешний вид нашего приложения показан на рис. 43.1.

Протестируйте приложение, запустив поочередно разные варианты блокировок. Посмотрите различия в работе экрана и, если на мобильном устройстве есть клавиатура, подсветки клавиатуры при создании разных блокировок.

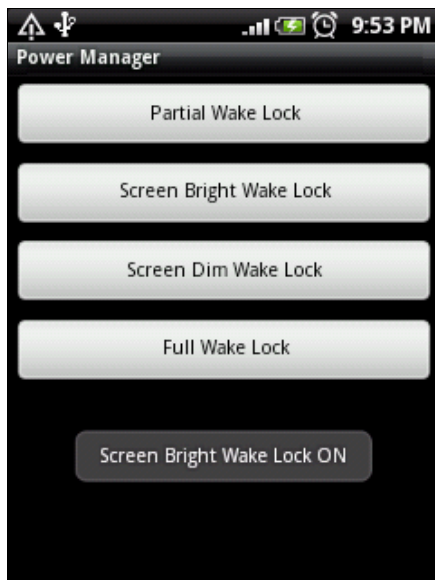


Рис. 43.1. Приложение для управления блокировками

Резюме

В этой главе мы рассмотрели использование менеджера энергопотребления для управления процессором, экраном и клавиатурой мобильного устройства. С помощью менеджера энергопотребления можно управлять мобильным телефоном с целью уменьшения потребления им энергии или, если для приложения требуются специфические варианты управления экраном и клавиатурой, создавать соответствующие блокировки.

В следующей, заключительной, главе мы изучим возможности получения в приложении информации о параметрах системы, доступной памяти, выполняющихся процессах, службах и заданиях, а также рассмотрим запуск команд Linux и создание процессов в коде приложения.



ГЛАВА 44

Получение информации о системе

Часто приложениям для работы на мобильном устройстве необходимо получать информацию о системе, наличии необходимого объема доступной памяти, выполняющихся процессах, службах, заданиях и открытых Activity. Также в приложениях может потребоваться информация об ошибках, возникающих в системе при выполнении процессов и заданий.

В этой главе мы рассмотрим, каким образом можно получать и обрабатывать информацию о состоянии системы Android и об аппаратной конфигурации мобильного устройства, на котором будет работать ваше приложение.

Класс *ActivityManager*

Для получения приложением информации о выполняющихся на мобильном устройстве службах и работающих в данный момент Activity используется класс `ActivityManager`, расположенный в пакете `android.app`. Получить объект `ActivityManager` можно обычным способом, как и другие менеджеры служб Android, через вызов метода `getSystemService()`, передав ему во входном параметре константу `ACTIVITY_SERVICE`, определенную в классе `Context`:

```
ActivityManager manager =  
    (ActivityManager) getSystemService(ACTIVITY_SERVICE);
```

Здесь необходимо уточнить, что название класса `ActivityManager` не в полной мере отражает его функциональность, поскольку класс `ActivityManager` организует взаимодействие приложения не только с Activity, но и с выполняющимися на мобильном устройстве службами.

Этот класс позволяет получить большое количество ценной информации о текущем состоянии системы. Использование объекта данного класса в приложении позволяет осуществлять мониторинг состояния системы, распределения памяти, отслеживать выполнение процессов и заданий.

Класс `ActivityManager` также позволяет получать доступ к аппаратной конфигурации мобильного устройства. Это важно при создании серийных приложений, которые могут выполняться на мобильных устройствах разных типов, моделей и версий операционной системы Android, чтобы гарантировать наличие требуемой конфигурации оборудования на данном устройстве, необходимой для корректной работы приложения.

Для изучения этих возможностей разработаем приложение, использующее функциональность `ActivityManager`. Данное приложение мы будем дополнять на протяжении всей этой главы, постепенно изучая возможности для доступа к информации о системе. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `ActivityManager`;
- ☐ **Application name** — `Activity Manager Info`;
- ☐ **Package name** — `com.samples.os.activitymanager`;
- ☐ **Create Activity** — `SystemInfoListActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch44_ActivityManager`.

В нашем приложении будут два `Activity` — для главного окна `SystemInfoListActivity`, где будет расположено меню в виде списка, и вспомогательное `SystemInfoItemActivity` для отображения информации о выполняющихся процессах, заданиях и др.

Код файла манифеста приложения `AndroidManifest.xml` представлен в листинге 44.1.

Листинг 44.1. Файл манифеста приложения `AndroidManifest.xml`

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

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".SystemInfoListActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SystemInfoItemActivity"
            android:label="@string/app_name">
        </activity>
    </application>

    <uses-permission
        android:name="android.permission.GET_TASKS">
    </uses-permission>

    <uses-sdk android:minSdkVersion="10" />
</manifest>
```


Файл компоновки окна приложения нам не понадобится, т. к. мы будем использовать для окна приложения стандартный список, определенный в системных ресурсах `android.R.layout.simple_list_item_1`.

Меню приложения будет включать в себя 7 пунктов:

- ❑ **Device Configuration Info** — информация о конфигурации устройства;
- ❑ **Memory Info** — информация об использовании памяти;
- ❑ **Running Processes** — работающие в данный момент процессы;
- ❑ **Running Services** — выполняющиеся службы;
- ❑ **Running Tasks** — выполняющиеся задания;
- ❑ **Recent Tasks** — последние выполненные задания;
- ❑ **Processes in Error State** — процессы, находящиеся в состоянии ошибки.

Для этого списка определим строковые элементы, которые поместим в файл ресурсов `strings.xml` (каталог `res/values`), код которого представлен в листинге 44.2.

Листинг 44.2. Файл ресурсов `strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">
        Activity Manager Info</string>
    <string name="itemDeviceConfigInfo">
        Device Configuration Info</string>
    <string name="itemMemoryInfo">
        Memory Info</string>
    <string name="itemRecentTasks">
        Recent Tasks</string>
    <string name="itemRunningAppProcesses">
        Running Processes</string>
    <string name="itemProcessesInErrorState">
        Processes in Error State</string>
    <string name="itemRunningServices">
        Running Services</string>
    <string name="itemRunningTasks">
        Running Tasks</string>
</resources>
```

В файле класса главного окна приложения `SystemInfoListActivity`, которое будет расширением для базового класса `ListActivity`, определим обработчик события выбора элемента списка `onListItemClick()`. В теле обработчика будет создаваться объект `Intent`, которому передадим выбранный элемент списка в виде строкового параметра, и будет происходить открытие дополнительного окна приложения `SystemInfoItemActivity` для вывода требуемой информации.

Файл класса `SystemInfoListActivity` представлен в листинге 44.3.

Листинг 44.3. Файл класса главного окна приложения SystemInfoListActivity.java

```
package com.samples.os.activitymanager;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class SystemInfoListActivity extends ListActivity {

    private String[] items;

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

        items = new String[] {
            getResources().getString(R.string.itemDeviceConfigInfo),
            getResources().getString(R.string.itemMemoryInfo),
            getResources().getString(R.string.itemRunningAppProcesses),
            getResources().getString(R.string.itemRunningServices),
            getResources().getString(R.string.itemRunningTasks),
            getResources().getString(R.string.itemRecentTasks),
            getResources().getString(R.string.itemProcessesInErrorState)
        };

        setListAdapter(new ArrayAdapter<String>(getApplicationContext(),
            android.R.layout.simple_list_item_1, items));
    }

    @Override
    protected void onItemClick(
        ListView parent, View v, int position, long id) {

        String item = items[position];
        Intent intent = new Intent(getApplicationContext(),
            SystemInfoItemActivity.class);
        intent.setAction(item);
        startActivity(intent);
    }
}
```

Для дополнительного окна, выводящего информацию о системе, создадим новый файл компоновки, который назовем `item.xml`. В этом окне будет текстовое поле `TextView`, в которое приложение будет выводить нужную нам информацию. Файл компоновки окна `item.xml` представлен в листинге 44.4.

Листинг 44.4. Файл компоновки окна item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text"
        android:text=""
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:layout_margin="5px"/>

</LinearLayout>
```

В коде класса `SystemInfoItemActivity` в зависимости от выбранного элемента списка, переданного как параметр, разместим структуру из операторов `if...else` для вывода соответствующих данных. Эту структуру мы пока оставим пустой и будем заполнять постепенно на протяжении этой главы.

Код класса `SystemInfoItemActivity` представлен в листинге 44.5.

Листинг 44.5. Файл класса главного окна приложения `SystemInfoItemActivity.java`

```
package com.samples.os.activitymanager;

import java.util.List;

import android.app.Activity;
import android.app.ActivityManager;
import android.app.ActivityManager.ProcessErrorStateInfo;
import android.app.ActivityManager.RecentTaskInfo;
import android.app.ActivityManager.RunningAppProcessInfo;
import android.app.ActivityManager.RunningServiceInfo;
import android.app.ActivityManager.RunningTaskInfo;
import android.content.pm.ConfigurationInfo;
import android.content.res.Configuration;
import android.os.Bundle;
import android.widget.TextView;

public class SystemInfoItemActivity extends Activity {

    private TextView text;

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

```
text = (TextView)findViewById(R.id.text);

String action = this.getIntent().getAction();
this.setTitle(action);

ActivityManager manager =
    (ActivityManager) getSystemService (ACTIVITY_SERVICE);

if (action.equals(
    getResources().getString(R.string.itemDeviceConfigInfo))) {
    // Информация о конфигурации устройства
}
else if (action.equals(
    getResources().getString(R.string.itemMemoryInfo))) {
    // Информация о памяти
}
else if (action.equals(
    getResources().getString(R.string.itemRunningAppProcesses))) {
    // Информация о выполняющихся процессах
}
else if (action.equals(
    getResources().getString(R.string.itemRunningServices))) {
    // Информация о выполняющихся службах
}
else if (action.equals(
    getResources().getString(R.string.itemRunningTasks))) {
    // Информация о выполняющихся заданиях
}
else if (action.equals(
    getResources().getString(R.string.itemRecentTasks))) {
    // Информация о последних выполненных заданиях
}
else if (action.equals(
    getResources().getString(R.string.itemProcessesInErrorState))) {
    // Информация о процессах в состоянии ошибки
}
}
```

После того как напишете код, на всякий случай проверьте работу приложения, запустив его на эмуляторе Android. Приложение должно выводить на экран список опций, как показано на рис. 44.1.

При выборе опций пока будут открываться пустые окна, но их наполнением мы займемся в следующих разделах этой главы по мере изучения функциональности и информации, предоставляемой этим менеджером.

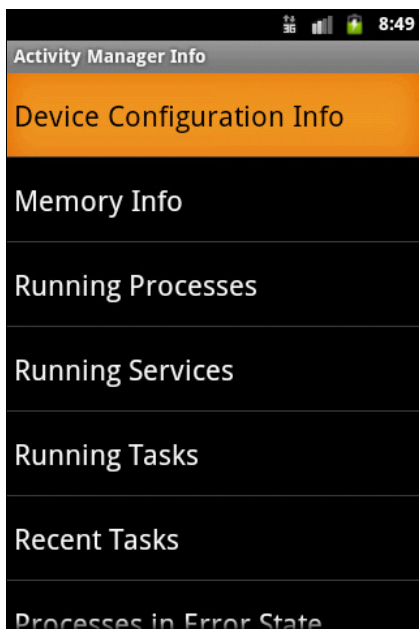


Рис. 44.1. Окно приложения со списком опций

Информация о конфигурации устройства

В классе `ActivityManager` есть метод `getDeviceConfigurationInfo()`, с помощью которого можно получить конфигурацию устройства или определить требования к конфигурации, необходимые для работы приложения на данном устройстве. Этот метод возвращает объект класса `ConfigurationInfo`, который находится в библиотеке `android.content.pm`.

Класс `ConfigurationInfo` инкапсулирует всю информацию об аппаратной конфигурации мобильного устройства. Эта информация необходима для приложений, которые предъявляют специфические требования для своей работы на конкретном мобильном устройстве, например, требуют наличия QWERTY-клавиатуры или шарикового навигатора.

В классе `ConfigurationInfo` определен набор полей, описывающих требования к конфигурации мобильного устройства. Так, например, поле `reqGlEsVersion` представляет версию графического драйвера OPEN GL. Поле `reqInputFeatures` определяет флаги, указывающие на требования к навигатору и клавиатуре, и состоит из комбинации двух битовых флагов:

- ☐ `INPUT_FEATURE_FIVE_WAY_NAV` — указывает, что приложение требует пятипозиционный навигатор;
- ☐ `INPUT_FEATURE_HARD_KEYBOARD` — указывает, что приложение требует наличие физической клавиатуры.

В классе `ConfigurationInfo` также есть три поля, определяющие более детальную конфигурацию для клавиатуры, навигатора и экрана мобильного устройства:

- ☐ `reqKeyboardType` — тип клавиатуры;
- ☐ `reqNavigation` — тип навигатора;
- ☐ `reqTouchScreen` — тип экрана мобильного устройства.

Поле `reqKeyboardType`, определяющее клавиатуру мобильного устройства, может принимать следующие значения:

- ☐ `KEYBOARD_12KEY` — стандартная клавиатура для набора телефонных номеров из 12 клавиш;
- ☐ `KEYBOARD_QWERTY` — полная клавиатура (QWERTY);
- ☐ `KEYBOARD_NOKEYS` — клавиатура отсутствует;
- ☐ `KEYBOARD_UNDEFINED` — тип клавиатуры не определен.

Поле `reqNavigation` определяет тип навигатора, используемого на мобильном устройстве. Навигаторы в мобильных устройствах могут иметь различную конструкцию, в зависимости от модели и производителя, и обычно представляют собой 4 кнопки со стрелками, шариковый манипулятор или мини-джойстик. Поле `reqNavigation`, в зависимости от типа навигатора, может принимать следующие значения:

- ☐ `NAVIGATION_NONAV` — устройство без навигатора;
- ☐ `NAVIGATION_DPAD` — навигатор в виде клавиш со стрелками;
- ☐ `NAVIGATION_TRACKBALL` — навигатор в виде шарика;
- ☐ `NAVIGATION_WHEEL` — навигатор в виде колесика;
- ☐ `NAVIGATION_UNDEFINED` — тип навигатора не определен.

Поле `reqTouchScreen` определяет тип экрана мобильного устройства. На типах экранов, используемых в мобильных устройствах, я бы хотел остановиться подробнее. В настоящее время производители смартфонов почти во всех телефонах устанавливают сенсорные экраны. При этом используются два типа сенсорных экранов: резистивный и емкостной. Существует еще один тип сенсорного экрана — индукционный, но он пока применяется только в планшетных PC.

Резистивный экран работает со стилусом и более дешевый по сравнению с другими типами сенсорных экранов. Резистивный экран состоит из двух прозрачных пластин, между которыми расположен диэлектрик. Обращенные друг к другу поверхности покрыты токопроводящим составом. При нажатии на экран стилусом верхний слой прогибается и касается нижнего, а система определяет координаты точки касания стилусом.

Емкостной экран реагирует на нажатие пальцем и представляет собой стеклянную панель, покрытую проводящим материалом. Палец человека имеет некоторую емкость, на которую реагирует экран. При касании экрана пальцем за счет подключения дополнительной емкости приложенного пальца происходит утечка тока. Величина утечки зависит от расстояния точки касания до токоъемных электродов, которые располагаются на краях экрана, и, таким образом, можно определить координаты точки касания.

В зависимости от типа экрана, поле `reqTouchScreen` может принимать следующие значения:

- ☐ `TOUCHSCREEN_STYLUS` — сенсорный экран резистивного типа;
- ☐ `TOUCHSCREEN_FINGER` — сенсорный экран емкостного типа;

- ❑ `TOUCHSCREEN_NOTOUCH` — обычный экран без сенсорного режима;
- ❑ `TOUCHSCREEN_UNDEFINED` — тип экрана не определен.

Пример работы с объектом `ConfigurationInfo` в нашем приложении для отображения информации о конфигурации мобильного устройства представлен в листинге 44.6 (это дополнение для класса `SystemInfoItemActivity` из листинга 44.5).

Листинг 44.6. Дополнения в классе `SystemInfoItemActivity` для вывода информации о конфигурации устройства

```
if (action.equals(  
    getResources().getString(R.string.itemDeviceConfigInfo)) ) {  
    ConfigurationInfo config = manager.getDeviceConfigurationInfo();  
    text.append("GLes Version:\t" + config.getGLesVersion());  
  
    int keyType = config.reqKeyboardType;  
    text.append("\nKeyboard Type:\t");  
  
    switch (keyType) {  
    case Configuration.KEYBOARD_UNDEFINED:  
        text.append("Undefined");  
        break;  
    case Configuration.KEYBOARD_NOKEYS:  
        text.append("No keys");  
        break;  
    case Configuration.KEYBOARD_QWERTY:  
        text.append("QWERTY");  
        break;  
    case Configuration.KEYBOARD_12KEY:  
        text.append("12 Key");  
        break;  
    }  
  
    int nav = config.reqNavigation;  
    text.append("\nNavigation:\t");  
  
    switch (nav) {  
    case Configuration.NAVIGATION_DPAD:  
        text.append("D pad");  
        break;  
    case Configuration.NAVIGATION_TRACKBALL:  
        text.append("Trackball");  
        break;  
    case Configuration.NAVIGATION_WHEEL:  
        text.append("Wheel");  
        break;  
    case Configuration.NAVIGATION_UNDEFINED:  
        text.append("Undefined");  
        break;  
    }  
}
```

```
int touch = config.reqTouchScreen;
text.append("\nTouch Screen:\t");

switch (touch) {
case Configuration.TOUCHSCREEN_FINGER:
    text.append("Finger");
    break;
case Configuration.TOUCHSCREEN_NOTOUCH:
    text.append("Notouch");
    break;
case Configuration.TOUCHSCREEN_STYLUS:
    text.append("stylus");
    break;
case Configuration.TOUCHSCREEN_UNDEFINED:
    text.append("Undefined");
    break;
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Откройте опцию **Devices Configuration Info**. Приложение выведет информацию о конфигурации мобильного устройства, в нашем случае — конфигурацию эмулятора: версию драйвера OPEN GL, типы клавиатуры, навигатора и экрана, установленные на устройстве. Внешний вид приложения с информацией о конфигурации представлен на рис. 44.2.

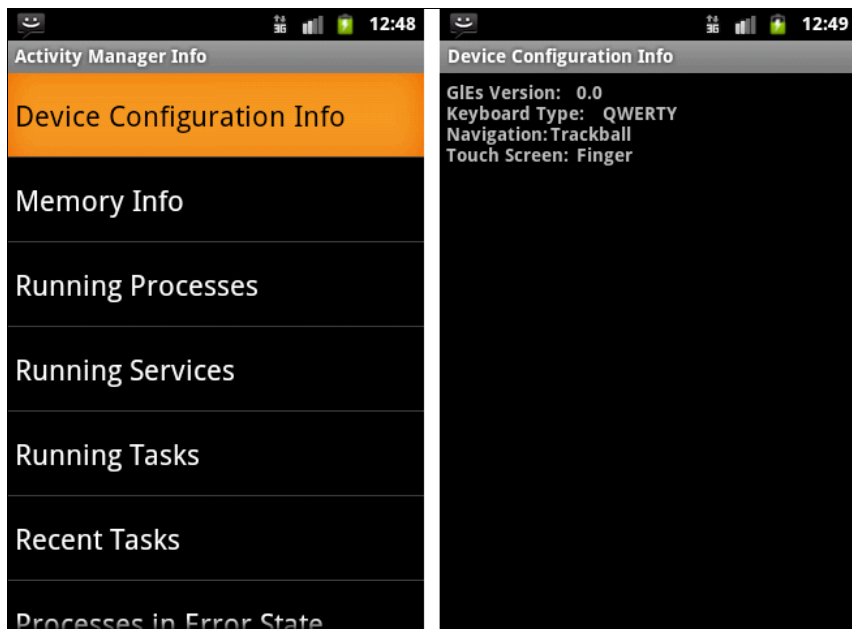


Рис. 44.2. Вывод информации о конфигурации устройства

Информация о системе

Для получения данных о системе, выполняющихся в данный момент времени процессах и заданиях пакет `android.app` предоставляет набор классов:

- ❑ `MemoryInfo` — представляет информацию о доступной памяти;
- ❑ `RecentTaskInfo` — представляет информацию о выполняющихся заданиях;
- ❑ `RunningAppProcessInfo` — представляет информацию о выполняющихся процессах;
- ❑ `RunningServiceInfo` — представляет информацию о выполняющихся в системе службах;
- ❑ `RunningTaskInfo` — представляет выполняющиеся в системе задания;
- ❑ `ProcessErrorStateInfo` — представляет информацию о выполняющихся процессах, находящихся в состоянии ошибки.

Доступ к этим объектам для получения требуемой информации в программном коде можно получить с помощью соответствующих методов класса `ActivityManager`, о чем будет рассказано позже.

Доступная память устройства

Для получения данных об общем количестве памяти и о доступной в текущий момент памяти мобильного устройства в классе `ActivityManager` есть два метода:

- ❑ `getMemoryInfo()` — возвращает объект класса `MemoryInfo` для получения информации об использовании и доступности памяти всеми процессами;
- ❑ `getProcessMemoryInfo()` — возвращает массив объектов `ActivityManager.MemoryInfo` для получения информации об использовании памяти в одном или нескольких процессах. Входным параметром для этого метода служит массив идентификаторов процессов, для которых требуется получить данные об использовании памяти.

В классе `ActivityManager.MemoryInfo` содержатся информационные поля, отображающие количество памяти в байтах:

- ❑ `availMem` — общее количество памяти, доступной в системе;
- ❑ `threshold` — нижний порог доступной памяти, при котором система начнет "убивать" процессы и службы, не являющиеся приоритетными;
- ❑ `lowMemory` — битовое поле, которое устанавливается в `true`, если в системе недостаточно свободной памяти.

Таким образом, чтобы получить объект `MemoryInfo`, необходимо вызвать метод `getMemoryInfo()`, которому в качестве `Out`-параметра надо передать объект `ActivityManager.MemoryInfo`. Метод `getMemoryInfo()` заполняет этот объект `ActivityManager.MemoryInfo` информацией о доступной памяти, и через открытые поля этого класса можно прочесть данные о распределении памяти.

Пример использования объекта `ActivityManager.MemoryInfo` в нашем приложении в коде класса `SystemInfoItemActivity` представлен в листинге 44.7.

Листинг 44.7. Дополнения в классе `SystemInfoItemActivity` для вывода информации о доступной памяти

```
else if (action.equals(
    getResources().getString(R.string.itemMemoryInfo))) {
    // Создаем объект ActivityManager.MemoryInfo
    MemoryInfo memInfo =
        new ActivityManager.MemoryInfo();

    // Передаем созданный объект memInfo
    // в метод getMemoryInfo()
    manager.getMemoryInfo(memInfo);
    text.append("Available Memory:\t" + memInfo.availMem + " B");
    text.append("\nTreshold Memory:\t" + memInfo.threshold + " B");
}
```

Запустите наше приложение на эмуляторе и откройте опцию **Memory Info**. Информация, которую выводит объект `ActivityManager.MemoryInfo` для эмулятора Android, представлена на рис. 44.3.

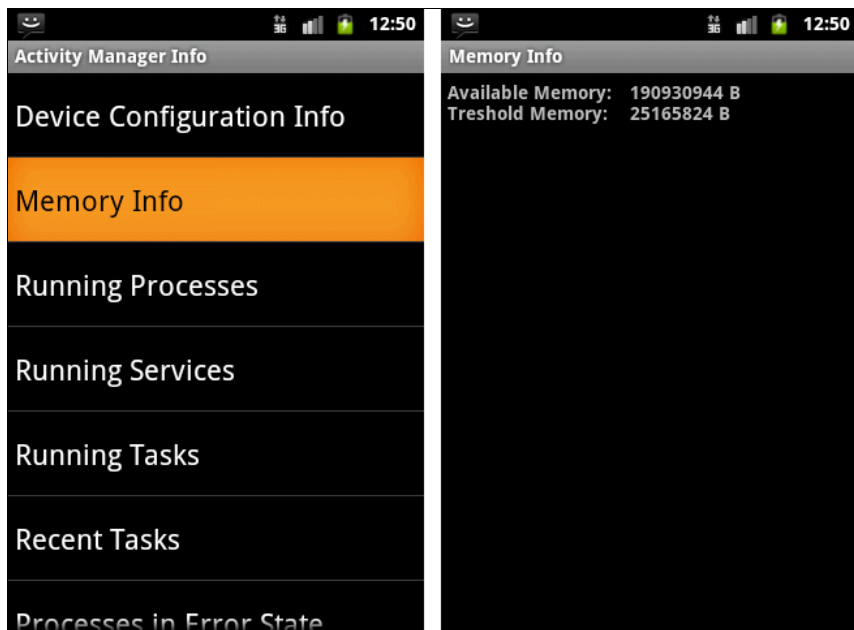


Рис. 44.3. Вывод информации о доступной памяти

Выполняющиеся процессы

С помощью класса `RunningAppProcessInfo` можно получать информацию о работающем в системе процессе. Кстати, это могут быть не только видимые или активные процессы. Пользователь во время работы на мобильном устройстве создает множество процессов,

которые потом не использует, забывая их закрыть. Эти процессы продолжают выполняться до тех пор, пока в системе не возникнет недостатка памяти и система сама не начнет закрывать эти процессы.

Для получения информации о процессах, выполняющихся в системе, в классе `ActivityManager` используется метод `getRunningAppProcesses()`, который возвращает список процессов, выполняющихся в данный момент на устройстве, в виде списка объектов `RunningAppProcessInfo`.

В классе `RunningAppProcessInfo` определено множество полей и констант, характеризующих выполняющийся процесс: `processName` — имя процесса, `pid` — идентификатор процесса PID (Process ID), `pkgList` — список пакетов и другая нужная информация.

Пример работы с объектами `RunningAppProcessInfo` для вывода информации о выполняющихся процессах в нашем приложении представлен в листинге 44.8.

Листинг 44.8. Дополнения в классе `SystemInfoItemActivity` для вывода информации о выполняющихся процессах

```
else if (action.equals(
    getResources().getString(R.string.itemRunningAppProcesses))) {
    List<RunningAppProcessInfo> runningAppProcesses =
        manager.getRunningAppProcesses();
    text.append("Running Processes:");

    for (int i = 0; i < runningAppProcesses.size(); i++) {
        text.append("\n\t" + runningAppProcesses.get(i).processName);
    }
}
```

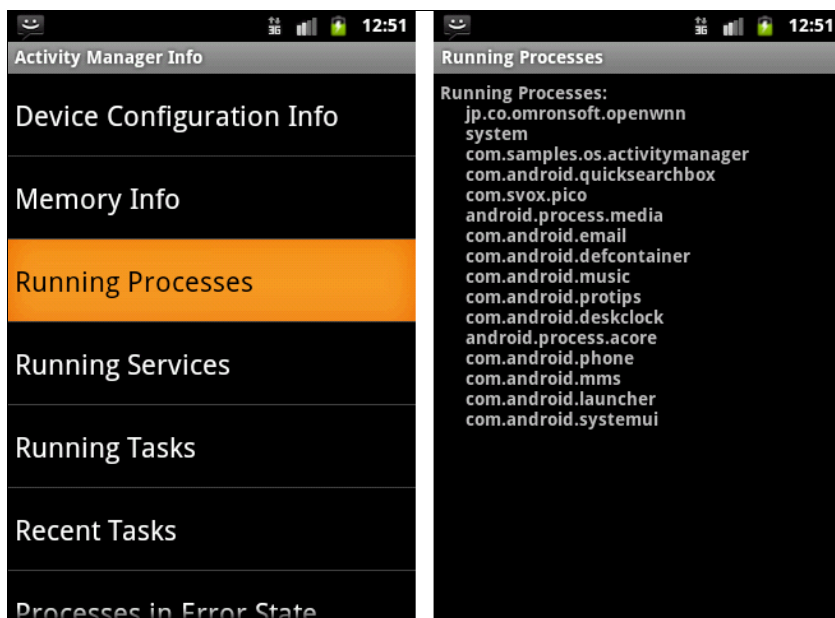


Рис. 44.4. Вывод информации о выполняющихся процессах

Выполните компиляцию приложения, запустите его на эмуляторе и откройте опцию **Running Processes**. На рис. 44.4 представлен возможный вариант списка процессов, выполняющихся в данный момент на мобильном устройстве.

Из-за большого количества процессов, обычно выполняющихся в системе, в окно были выведены только имена процессов, но, при желании, вы можете вывести и остальные поля, определяющие характеристики процессов.

Выполняющиеся службы

Класс `RunningServiceInfo` предоставляет информацию о конкретной службе, в данный момент работающей в системе.

В классе `ActivityManager` есть метод `getRunningServices()`, который возвращает список служб, выполняющихся в данный момент на устройстве, в виде списка объектов `RunningServiceInfo`. Однако, в отличие от рассмотренного ранее метода `getRunningAppProcesses()`, в метод `getRunningServices()` передается входной параметр, определяющий максимальное число выводимых в список служб. Класс `RunningServiceInfo` имеет набор полей, похожих на поля класса `RunningAppProcessInfo`, а также поля, специфические для служб, например `clientCount` — количество клиентов, использующих службу, `foreground` — служба требует выполнения в приоритетном процессе и др.

Пример работы с объектами `RunningServiceInfo` для вывода информации о выполняющихся службах в нашем приложении представлен в листинге 44.9.

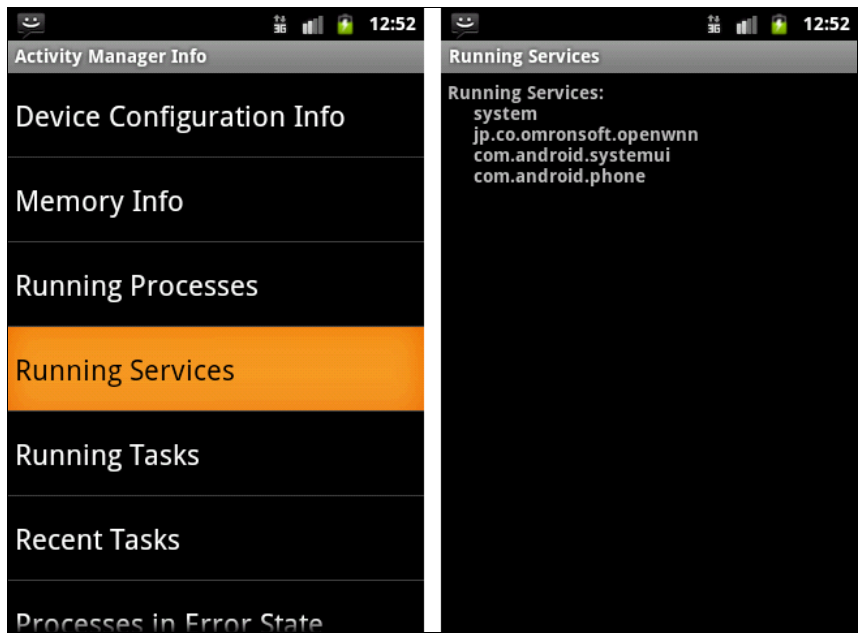


Рис. 44.5. Вывод информации о выполняющихся службах

Листинг 44.9. Дополнения в классе `SystemInfoItemActivity` для вывода информации о выполняющихся службах

```
else if (action.equals(
    getResources().getString(R.string.itemRunningServices))){
    List<RunningServiceInfo> runningServices =
        manager.getRunningServices(20);
    text.append("Running Services:");

    for (int i = 0; i < runningServices.size(); i++) {
        text.append("\n\t" + runningServices.get(i).process);
    }
}
```

Если запустить приложение и открыть опцию **Running Services**, то информация, отображающая работающие в системе службы, будет выглядеть так, как представлено на рис. 44.5.

Выполняющиеся задания

Класс `RunningTaskInfo` предоставляет информацию о конкретном задании, которое в настоящее время выполняется в системе.

Для получения информации о заданиях, выполняющихся в системе, в классе `ActivityManager` используется метод `getRunningTasks()`, который возвращает список выполняющихся заданий в виде списка объектов `RunningTaskInfo`. В этот метод также передается параметр, определяющий максимальное число выводимых в список заданий.

Класс `RunningTaskInfo` имеет набор полей, которые хранят информацию о задании:

- ❑ `id` — идентификатор задания;
- ❑ `baseActivity` — компонент, который должен запускаться первым в этом задании;
- ❑ `description` — описание текущего состояния задания;
- ❑ `numActivities` — общее количество объектов `Activity` в задании;
- ❑ `numRunning` — количество объектов `Activity`, которые в настоящий момент выполняются в данном задании;
- ❑ `topActivity` — компонент, который в данный момент находится вверху стека в данном задании.

Теперь добавим в наше приложение возможность получать информацию о выполняющихся в системе заданиях.

В файл манифеста приложения `AndroidManifest.xml` добавьте разрешение `android.permission.GET_TASKS`. Это разрешение позволяет приложению читать информацию о выполняющихся в данный момент времени заданиях.

Код для работы с объектами `RunningTaskInfo` для вывода информации о выполняющихся в системе заданиях для нашего приложения представлен в листинге 44.10.

Листинг 44.10. Дополнения в классе `SystemInfoItemActivity` для вывода информации о заданиях

```
else if (action.equals(
    getResources().getString(R.string.itemRunningTasks))) {
    List<RunningTaskInfo> runTasks = manager.getRunningTasks(10);

    text.append("Running Tasks:");

    for (int i = 0; i < runTasks.size(); i++) {
        RunningTaskInfo runTask = runTasks.get(i);
        text.append("\nTask ID:\t" + runTask.id);
        text.append("\n\tBase Activity:\n\t\t" +
            runTask.baseActivity.getShortClassName());
        text.append("\n\tNum Activities: " + runTask.numActivities);
        text.append("\n\tNum Running: " + runTask.numRunning);
    }
}
```

При открытии опции **Running Tasks** внешний вид приложения, в окне которого выведена информация о двух выполняемых заданиях, представлен на рис. 44.6.

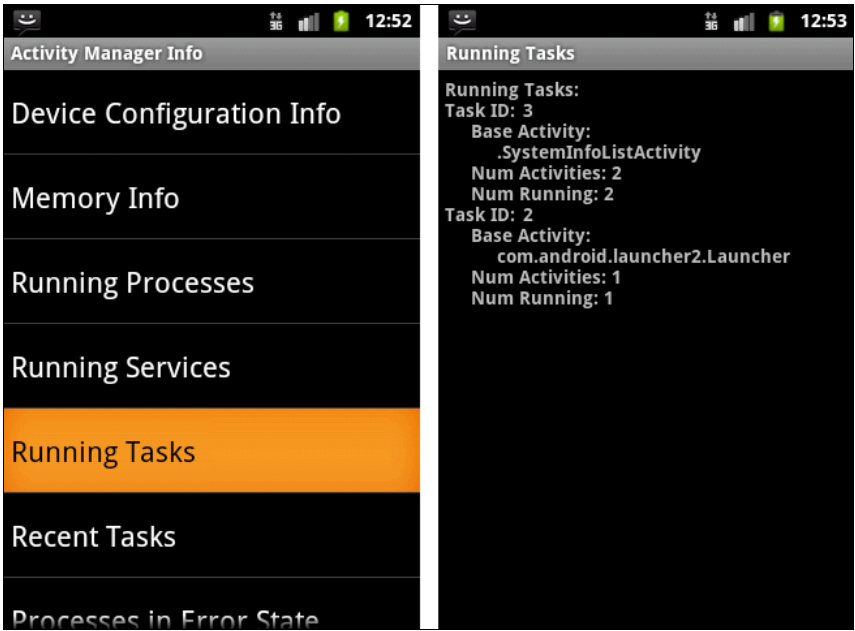


Рис. 44.6. Вывод информации о выполняющихся заданиях

Последние выполненные задания

Класс `RecentTaskInfo` предоставляет информацию о последних заданиях, которые запускал пользователь.

Для получения информации о последних выполненных в системе заданиях в классе `ActivityManager` используется метод `getRecentTasks()`, который возвращает список объектов `RecentTaskInfo`. В отличие от метода `getRunningTasks()`, в метод `getRecentTasks()` передаются два параметра. Первый параметр определяет максимальное число выводимых в список заданий, второй является комбинацией двух битовых флагов:

- ❑ `RECENT_WITH_EXCLUDED` — вернуть все задания;
- ❑ `RECENT_IGNORE_UNAVAILABLE` — вернуть список, в который не включены последние задания, недоступные для пользователей. Этот флаг появился только в версии API Level 11 (Android SDK 3.0).

Класс `RecentTaskInfo` имеет следующий набор полей:

- ❑ `id` — идентификатор задания, если это задание в настоящий момент выполняется;
- ❑ `description` — описание последнего состояния задания;
- ❑ `baseIntent` — базовый объект `Intent`, использовавшийся для запуска этого задания;
- ❑ `origActivity` — имя компонента, из которого было запущено задание.

Пример работы с объектами `RecentTaskInfo` для вывода информации о последних выполненных заданиях представлен в листинге 44.11.

Листинг 44.11. Дополнения в классе `SystemInfoItemActivity` для вывода информации о последних выполненных заданиях

```
else if (action.equals(
    getResources().getString(R.string.itemRecentTasks)) {
    List<RecentTaskInfo> recTasks = manager.getRecentTasks(10, 0);
    text.append("Recent Tasks:");

    for (int i = 0; i < recTasks.size(); i++) {
        RecentTaskInfo recTask = recTasks.get(i);

        text.append("\n\tTask ID:\t" + recTask.id);
        text.append("\n\tBase Intent:\t" + recTask.baseIntent.getAction());

        if (recTask.origActivity != null) {
            text.append("Orig Activity:\t" +
                recTask.origActivity.getShortClassName());
        }
    }
}
```

Если запустить наше приложение на эмуляторе и открыть опцию **Recent Tasks**, можно увидеть, что приложение будет выводить идентификатор задания и базовый объект `Intent`, который создал это задание. Внешний вид приложения с отображаемой информацией представлен на рис. 44.7.

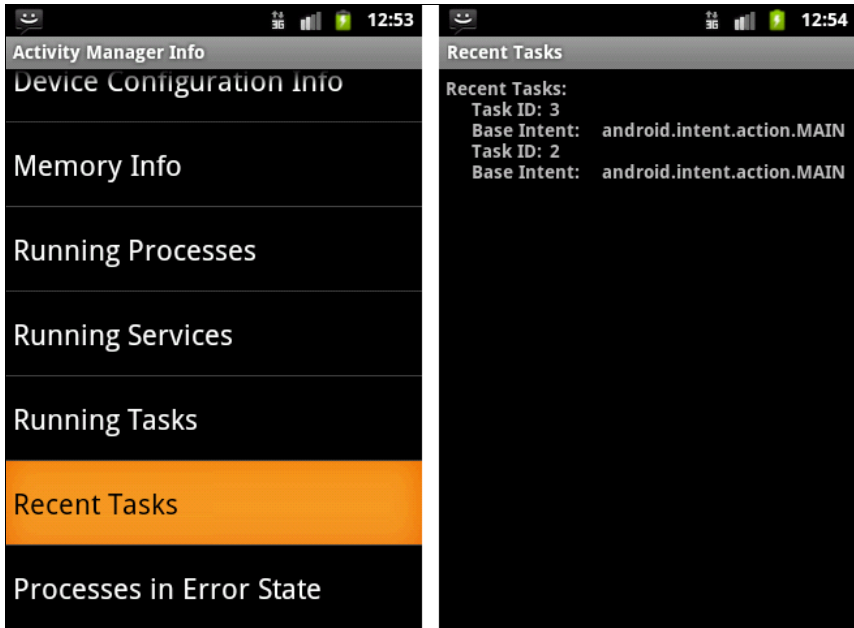


Рис. 44.7. Вывод информации о последних выполненных заданиях

Процессы в состоянии ошибки

Выполняющиеся в системе процессы в некоторых случаях могут быть завершены с ошибкой. Для отображения информации о конкретном процессе используется класс `ProcessErrorStateInfo`.

Для получения всех процессов, находящихся в состоянии ошибки, в классе `ActivityManager` используется метод `getProcessesInErrorState()`, который возвращает список объектов `ProcessErrorStateInfo`.

Класс `ProcessErrorStateInfo` содержит свои специфические поля, отличающиеся от полей класса `RunningAppProcessInfo`. Эти поля позволяют получить информацию, идентифицирующую процесс и описание возникшей ошибки, приведшей к краху процесса:

- ☐ `processName` — имя процесса, в котором произошла ошибка;
- ☐ `pid` — идентификатор процесса, если его нет, выводится 0;
- ☐ `uid` — пользовательский идентификатор (User ID), который был назначен этому процессу. Этот идентификатор не является уникальным, т. к. несколько процессов могут иметь одинаковый UID;
- ☐ `tag` — имя объекта `Activity`, ассоциированного с этой ошибкой, если оно известно;
- ☐ `condition` — условие, из-за которого процесс находится в состоянии ошибки;
- ☐ `shortMsg` — строка с кратким описанием ошибки;
- ☐ `longMsg` — строка с детальным описанием ошибки;
- ☐ `stackTrace` — строка, содержащая трассировку стека с ошибкой.

Пример работы с объектами `ProcessErrorStateInfo` для вывода информации о процессах в состоянии ошибки представлен в листинге 44.12.

Листинг 44.12. Дополнения в классе `SystemInfoItemActivity` для вывода информации о процессах в состоянии ошибки

```
else if (action.equals(  
    getResources().getString(R.string.itemProcessesInErrorState))) {  
    List<ProcessErrorStateInfo> procError =  
        manager.getProcessesInErrorState();  
    text.append("ProcessesInErrorState:");  
  
    if (procError != null) {  
        for (int i = 0; i < procError.size(); i++) {  
            text.append("\n\t" + procError.get(i).processName);  
        }  
    }  
    else {  
        text.append("\tNone");  
    }  
}
```

Выполните компиляцию приложения и откройте опцию **Processes in Error State**. Поскольку в системе не было процессов в состоянии ошибки, приложение ничего не выведет на экран. Внешний вид приложения представлен на рис. 44.8.

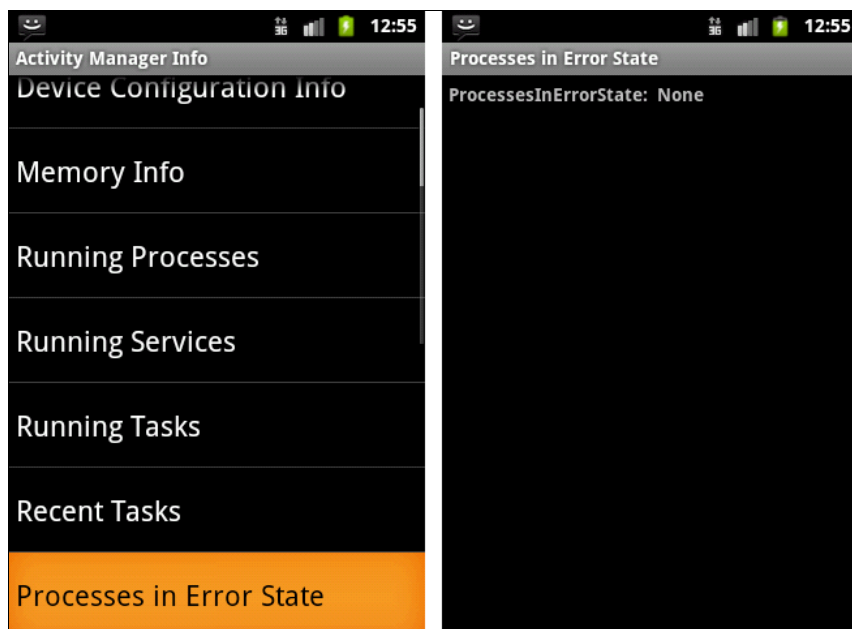


Рис. 44.8. Вывод информации о процессах в состоянии ошибки

ПРИМЕЧАНИЕ

Если процессы в состоянии ошибки отсутствуют, метод `getProcessesInErrorState()` возвращает `null`, а не пустой список, учитывайте это при обработке в программе.

Терминал в системе Android

В системе Android, так же как и других Linux-системах, любую задачу можно выполнить из консоли. Чтобы открыть терминал на мобильном устройстве, запустите окно со списком приложений, зайдите в **Development Tools** и выберите опцию **Terminal Emulator**, как показано на рис. 44.9.

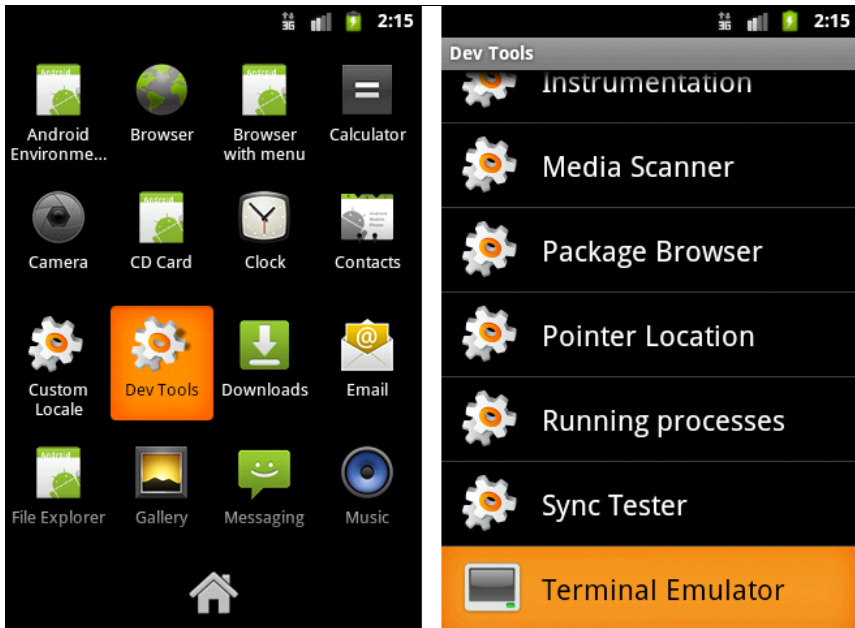


Рис. 44.9. Доступ к терминалу в системе Android

Из окна терминала доступно много различных команд, которые аналогичны командам, используемым в системах Linux. Так же как и в Linux, все команды и ключи чувствительны к регистру. Внешний вид открытого окна терминала, например, с выполненной командой `df` в Android представлен на рис. 44.10.

Команды Linux можно запускать и из кода приложения. Можно также читать в приложении результат выполнения этих команд. Для этого используются два класса из стандартной библиотеки `java.lang`: `Process` и `ProcessBuilder`.

Класс `Process` представляет собой непосредственно процесс. Класс `ProcessBuilder` предоставляет нужную функциональность для управления процессами. Чтобы создать процесс, необходимо сначала создать экземпляр класса `ProcessBuilder`, передав в конструктор имя программы и необходимые аргументы для этой программы. Затем, чтобы начать выполнение программы, необходимо вызвать метод `start()` класса `ProcessBuilder`, например, таким образом:

```
String args = "df";
ProcessBuilder builder = new ProcessBuilder(args);
Process process = builder.start();
```

Результатом будет выполнение Linux-команды `df` без аргументов, которая выведет информацию о смонтированных файловых системах и объеме их дискового пространства на мобильном устройстве.

```
export PATH=/data/local/bin:$PATH
$ export PATH=/data/local/bin:$PATH
$ df
Filesystem      Size  Used  Free  Blks
/dev            125M   32K   125M  4096
/mnt/asec       125M    0K   125M  4096
/mnt/obb        125M    0K   125M  4096
/system         86M   86M    0K   4096
/data           64M   29M   34M  4096
/cache          64M    1M   62M  4096
/mnt/sdcard     1019M   6K  1019M  2048
/mnt/secure/asec: Permission denied
$
```

Рис. 44.10. Открытое окно терминала в Android

Теперь попробуем использовать подобный способ в приложении. Создайте в IDE Eclipse новый проект Android и заполните соответствующие поля в диалоговом окне **New Android Project**:

- ☐ **Project name** — `AndroidOSInfo`;
- ☐ **Application name** — `Read Android OS Info`;
- ☐ **Package name** — `com.samples.hardware.osinfo`;
- ☐ **Create Activity** — `OsInfoListActivity`.

ПРИМЕЧАНИЕ

Полный код приложения находится в каталоге `Ch44_AndroidOSInfo`.

Приложение будет похоже на программу, созданную ранее в этой главе. В нашем приложении будет два Activity — одно для главного окна `OsInfoListActivity`, где будет расположено меню в виде списка, и вспомогательное окно `OsInfoItemActivity` для отображения информации о выполняющихся процессах, заданиях и др.

Код файла манифеста приложения представлен в листинге 44.13.

Листинг 44.13. Файл манифеста приложения `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.samples.hardware.osinfo"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10" />
```

```

<application
    android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".OsInfoListActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".OsInfoItemActivity"
        android:label="@string/app_name">
    </activity>
</application>
</manifest>

```

В коде класса главного окна приложения `OsInfoListActivity` разместим меню в виде списка, в котором будет 4 опции:

- ☐ **OS Version Information** — информация о версии системы;
- ☐ **CPU Information** — информация о процессоре;
- ☐ **Memory Information** — информация о распределении памяти;
- ☐ **Disc Space Information** — информация о распределении дискового пространства.

Для этого списка определим внешние строковые ресурсы, которые поместим в файл ресурсов `strings.xml`, находящийся в каталоге `res/values` проекта. Код файла `strings.xml` представлен в листинге 44.14.

Листинг 44.14. Файл ресурсов `strings.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Read Android OS Info</string>
    <string name="os_version_info">OS Version Information</string>
    <string name="cpu_info">CPU Information</string>
    <string name="memory_info">Memory Information</string>
    <string name="disc_space_info">Disc Space Information</string>
</resources>

```

При выборе элемента из этого списка откроется дополнительное окно, в которое будет выведена соответствующая информация. Код класса `OsInfoListActivity` представлен в листинге 44.15.

Листинг 44.15. Файл класса главного окна приложения `OsInfoListActivity.java`

```

package com.samples.hardware.osinfo;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;

```

```

import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;

public class OsInfoListActivity extends ListActivity {
    private String[] items;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        items = new String[] {
            getResources().getString(R.string.os_version_info),
            getResources().getString(R.string.cpu_info),
            getResources().getString(R.string.memory_info),
            getResources().getString(R.string.disc_space_info)
        };

        setListAdapter(new ArrayAdapter<String>(getApplicationContext(),
            android.R.layout.simple_list_item_1, items));
    }

    @Override
    protected void onListItemClick(
        ListView parent, View v, int position, long id) {
        try {
            String item = items[position];

            Intent intent = new Intent(getApplicationContext(),
                OsInfoItemActivity.class);
            intent.setAction(item);
            startActivity(intent);
        }
        catch (Exception e) {
            Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
        }
    }
}

```

В коде класса `OsInfoItemActivity` в зависимости от выбранного элемента списка, переданного как параметр, будет структура из операторов `if...else` для вывода соответствующих данных. Код класса `OsInfoItemActivity` представлен в листинге 44.16.

Листинг 44.16. Файл класса окна `OsInfoItemActivity.java`

```

package com.samples.hardware.osinfo;

import java.io.IOException;
import java.io.InputStream;

```

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class OsInfoItemActivity extends Activity {

    private TextView text;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.item);

        text = (TextView)findViewById(R.id.text);

        String action = this.getIntent().getAction();
        this.setTitle(action);

        if (action.equals(
            getResources().getString(R.string.cpu_info))) {
            String[] args = {"/system/bin/cat", "/proc/cpuinfo"};
            PrintInfo(args);
        }
        else if (action.equals(
            getResources().getString(R.string.disc_space_info))) {
            String[] args = {"/system/bin/df"};
            PrintInfo(args);
        }
        else if (action.equals(
            getResources().getString(R.string.memory_info))) {
            String[] args = {"/system/bin/cat", "/proc/meminfo"};
            PrintInfo(args);
        }
        else if (action.equals(
            getResources().getString(R.string.os_version_info))) {
            String[] args = {"/system/bin/cat", "/proc/version"};
            PrintInfo(args);
        }
    }

    // Вывод информации о системе на дисплей мобильного устройства
    private void PrintInfo(String[] args)
    {
        try{
            ProcessBuilder builder = new ProcessBuilder(args);

            Process process = builder.start();
            InputStream inputStream = process.getInputStream();
            byte[] b = new byte[4096];
```

```
while(inputStream.read(b) != -1){
    text.append(new String(b));
}
inputStream.close();
}
catch(IOException e){
    Toast.makeText(this, e.toString(), Toast.LENGTH_LONG).show();
}
}
}
```

Выполните компиляцию проекта и запустите его на эмуляторе Android. Посмотрите информацию о системе и оборудовании, которую выдает приложение. Внешний вид приложения, выдающего информацию о памяти мобильного устройства, представлен на рис. 44.11.

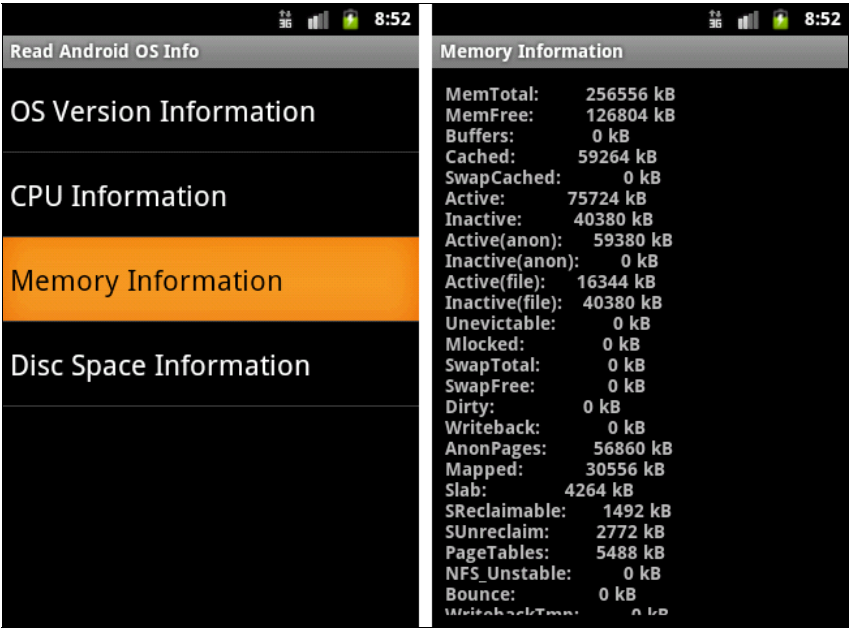


Рис. 44.11. Вывод информации о памяти мобильного устройства

Можете также запустить приложение на реальном мобильном устройстве. Кстати, информация, выводимая приложением на эмуляторе и на реальном телефоне Android, почти не отличается. Дело в том, что эмулятор Android очень точно моделирует реальную архитектуру процессоров ARM и оборудования. Однако такая имитация имеет и свои недостатки — запущенный экземпляр эмулятора Android потребляет много ресурсов, особенно памяти компьютера, а самый "тяжелый" из эмуляторов Android для версии Android 3.0 вообще лучше запускать на компьютере с памятью не менее 2 Гбайт.

Резюме

В этой главе мы изучили возможности, предоставляемые библиотекой Android SDK для получения приложением информации об установленной системе и оборудовании мобильного устройства. Мы научились читать в коде приложения информацию о распределении памяти, выполняющихся службах, запущенных процессах и заданиях. Кроме того, мы узнали о запуске процессов и возможностях использования системных команд Linux из кода приложения и чтения информации, получаемой с помощью этих команд.

Ввиду ограниченного объема книги мы не смогли изучить работу некоторых служб платформы Android, но я надеюсь, что после прочтения этой книги у вас появится желание самостоятельно продолжать более углубленное исследование этой перспективной платформы.

Я надеюсь, что данная книга помогла вам в изучении программирования новой интересной платформы для мобильных устройств. Вы получили навыки создания приложений для этой платформы. Дальше вы можете самостоятельно развиваться и разрабатывать приложения для той предметной области, которая вам нравится. Кто-то захочет создавать игры, а кто-то — приложения для групповой коммуникации пользователей или доступа к сетевым сервисам.

Желаю успехов!

ПРИЛОЖЕНИЕ

Описание электронного архива и установка примеров

Электронный архив

Электронный архив с материалами к книге выложен на FTP издательства, скачать его можно по ссылке **<ftp://ftp.bhv.ru/9785977508803.zip>**. Эта ссылка доступна также со страницы книги на сайте **www.bhv.ru**.

В архиве находятся два каталога: Samples и Resources. В каталоге Samples располагаются файлы проектов, описанных в книге. Каталог Resources состоит из трех подкаталогов:

- ☐ Animation — изображения для анимации (*глава 23*);
- ☐ Images — изображения для отображения в виджетах-списках (*глава 17* и далее по книге);
- ☐ Menu_Icons — значки для создания пользовательских уведомлений (*глава 9*), диалоговых окон (*глава 10*), меню (примеры из *глав 11* и далее по книге).

Большинство изображений взято из ресурсов дистрибутива Android SDK. При желании вы можете использовать собственные изображения.

Установка примеров

В каталоге Samples электронного архива находятся все примеры приложений, рассмотренных в книге. Для их установки на компьютер создайте в Eclipse новое рабочее пространство. Для этого выберите пункты меню **File | Switch Workspace | Other** и в открывшемся диалоговом окне **Workspace Launcher** укажите каталог, в котором вы хотите разместить новое рабочее пространство для примеров (рис. П1).

После создания рабочего пространства необходимо заново создать ссылку на каталог с распакованным Android SDK. Для этого выберите пункт меню **Window | Preferences**. В открывшемся окне **Preferences** выберите на левой панели пункт **Android**. В поле **SDK Location** на правой панели окна необходимо указать каталог, в котором расположен Android SDK. Для этого нажмите кнопку **Browse** и установите путь к каталогу Android SDK, как показано на рис. П2.

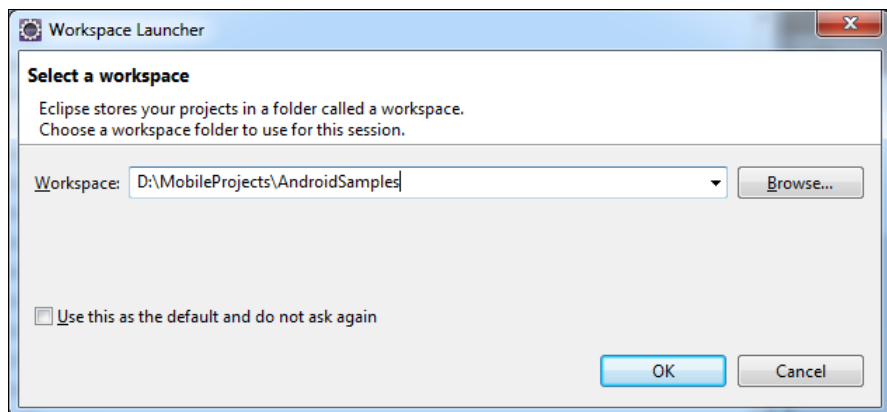


Рис. П1. Создание рабочего пространства в Eclipse

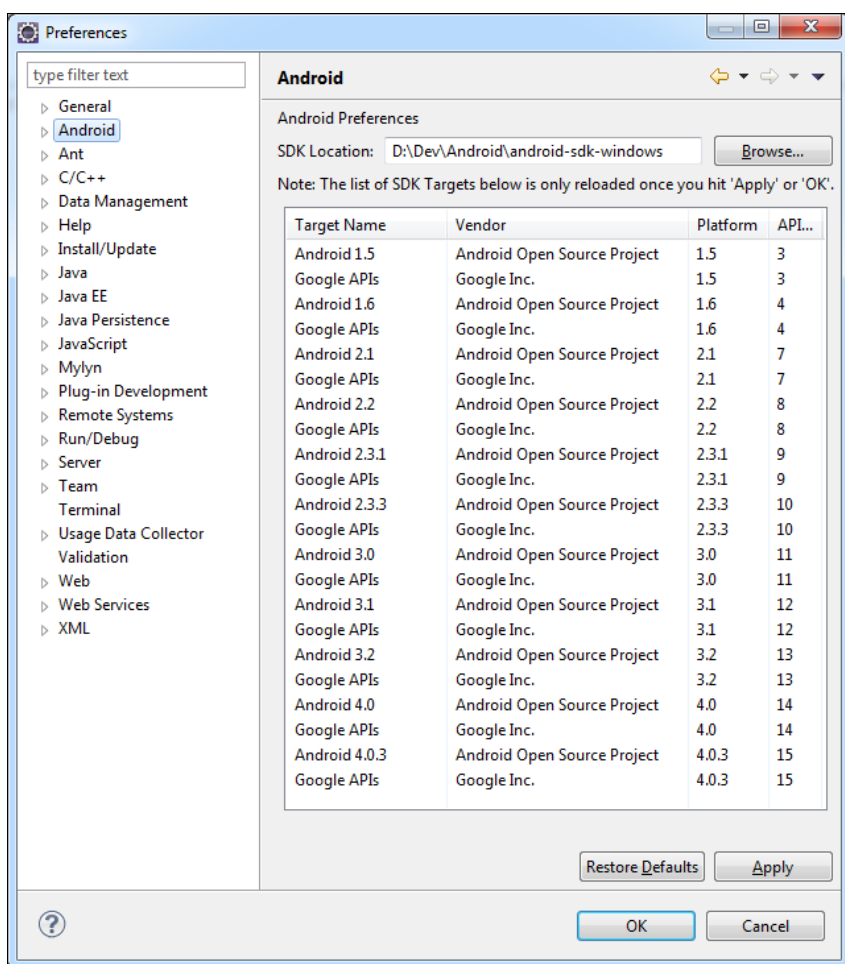


Рис. П2. Подключение Android SDK

Теперь, когда рабочее пространство для примеров создано и настроено, можно импортировать проекты из архива. Для этого выберите пункт меню **File | Import** и в открывшемся диалоговом окне **Import** выберите опцию **General/Existing Projects into Workspace**, как показано на рис. ПЗ. Нажмите кнопку **Next**.

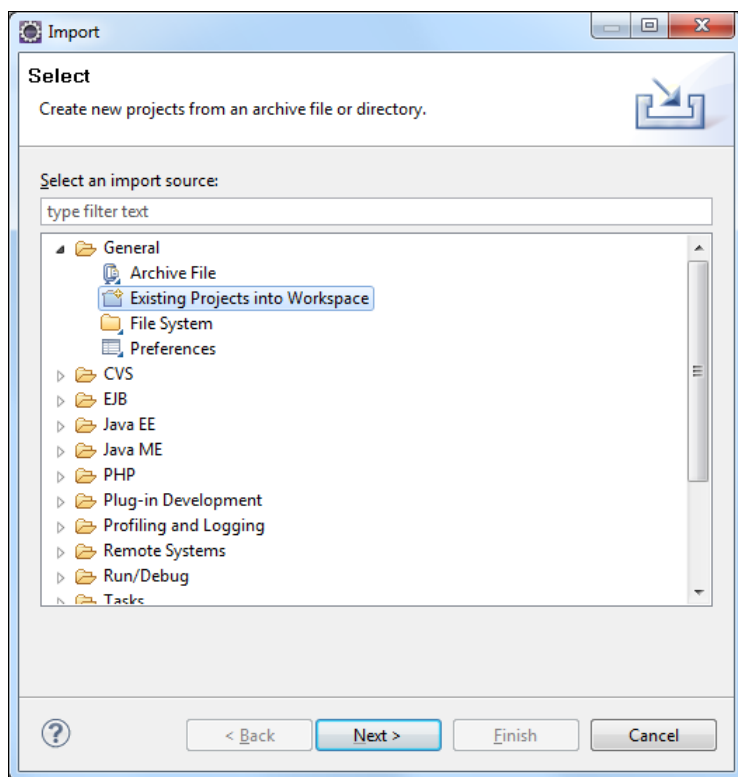


Рис. ПЗ. Окно мастера импорта файлов

В следующем окне выберите **Select archive file** и укажите путь к архиву **AndroidSamples.zip**. Список **Projects** отобразит все проекты, находящиеся в каталоге **Samples**. Для импорта всех проектов из каталога нажмите кнопку **Select All**, а также установите флажок **Copy projects into workspace** для копирования файлов из папки в созданный вами каталог для рабочего пространства, как показано на рис. П4.

Все проекты должны быть импортированы из архива в ваш рабочий каталог без ошибок (рис. П5).

Обратите внимание на имена проектов: все проекты содержат в названии префикс, указывающий на главу книги, например **Ch09_CustomDialog**, в то время как в тексте книги этот проект имеет название **CustomDialog**. Это сделано для более удобной группировки проектов в рабочем пространстве по соответствующим главам книги.

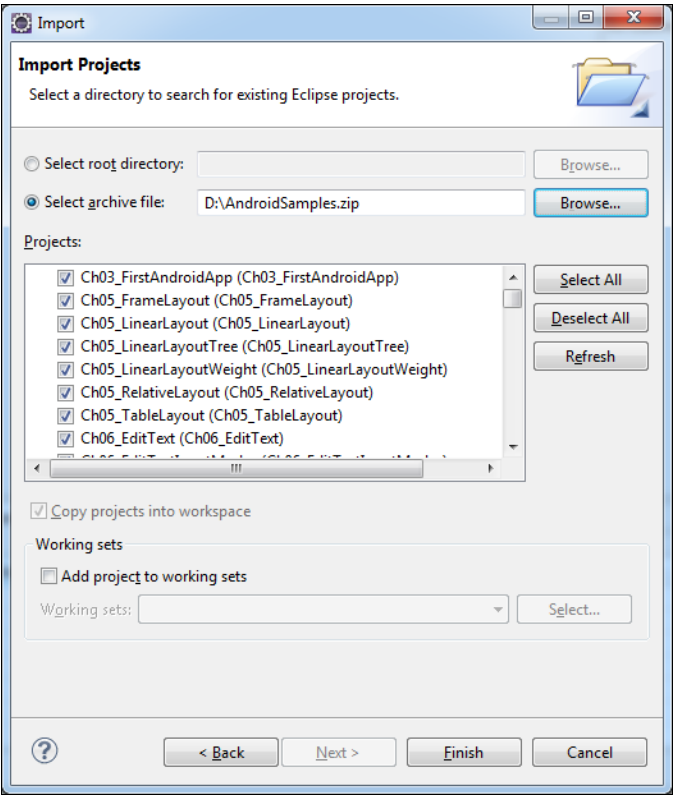


Рис. П4. Окно выбора импортируемого каталога и проектов

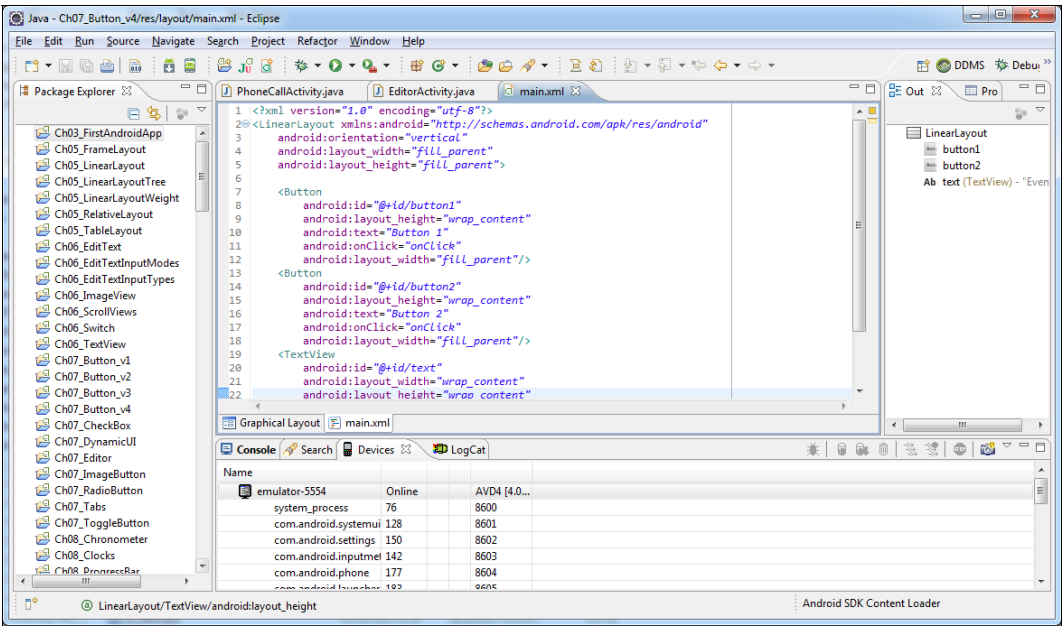


Рис. П5. Рабочее пространство с импортированными проектами

Предметный указатель

A

AbsListView 326
Accessibility Service 478
Account Service 479
Action Bar 509
Activity 31
Activity Service 478
AdapterView 325, 326
Alarm Service 478
AnalogClock 162
Android Development Tools 35–37
Android Power Management 28
Android SDK 36, 37
Android Virtual Device 43, 305
AndroidManifest.xml 64
Application Launcher 425
AppWidgetManager 492
ArrayAdapter 325
Audio Service 478
AVD Manager 37, 43–45

B

Battery Manager 479
Broadcast Intent 483, 767
Broadcast Receiver 31, 481, 483
Bundle 239
Button 126

C

CheckBox 126, 136, 137
CheckBoxPreference 404
Chronometer 162, 163
Clipboard Service 478

Connectivity Service 479
Content Provider 31
Context Menu 205
Cursor 325, 387

D

Dalvik Debug Monitor Service 41
Dalvik Virtual Machine 29
DDMC (Dalvik Debug Monitor Server) 307
DDMS 39, 73, 557, 561
Device_policy Service 479
DigitalClock 162
Download Service 479
Draw 9-patch 42
Dropbox Service 478

E

Eclipse 35–39, 41, 43, 51
EditText 110
EditTextPreference 404
Explorer View 103

F

Forward Geocoding 663
Frame animation 443
FrameLayout 88, 119

G

Gallery 325, 353
Geocoding 663
GPS (Global Positioning System) 648
GridView 325, 343, 347

H

Hierarchy Viewer 42, 101
HorizontalScrollView 119

I

ImageButton 126, 141
ImageView 121
Input Method Service 478
Intent 32
IntentFilter 232
Iterator 392

J

Java Runtime Environment 36

K

Keyguard Service 478

L

Layout Editor 87
Layout Inflater Service 477
Layout View 101
layoutopt 42
layouts 84
LinearLayout 88, 127
ListActivity 328
ListPreference 404
ListView 325, 328, 329
Location Service 479
Logcat 75
Loupe View 103

M

mkscard 42

N

NFC Service 479
Normal View 103
Notification Service 478

O

onDraw() 438

P

PaintDrawable 280
Pixel Perfect View 101, 103
Power Service 479
Preferences 403
ProgressBar 152
Properties View 101

R

RadioButton 126, 134
RatingBar 155, 162
RelativeLayout 88, 99
Request Layout 102
Reverse Geocoding 663
RingtonePreference 404

S

ScrollView 119
Search Service 478
SecurityException 473
SeekBar 155, 156
Sensor Service 479
Sensor Simulator 755
Service 31
SharedPreferences 405
SimpleAdapter 327, 336
SimpleCursorAdapter 336
SlidingDrawer 325, 356
SMS (Short Message Service) 573
Spinner 325, 361
sqlite3 42
SQLiteDatabase 376
SQLiteOpenHelper 375
Storage Service 479
String 111

T

TabHost 143
TableLayout 88, 95
TabWidget 143
Telephony Service 479
Text To Speech 528
TextView 105, 106, 121
Toast Notification 167
ToggleButton 126, 139
Traceview 42
Tween Animation 443, 447, 453, 457

U

UI Mode Service 478
URI 385

V

Vibrator Service 479

View 83, 106
ViewGroup 83

W

Wallpaper Service 477
Wifi Service 479
Window Service 478

A

Адаптеры данных 326
Активный процесс 228
Активы 279, 300
Анимация 280
Атрибут
◊ android:name 492
◊ android:resource 492
Атрибуты 446

Б

База данных
◊ Browser 372
◊ CallLog 372
◊ ContactsContract 372
◊ Mediastore 372
◊ Settings 372
◊ SQLite 371
◊ UserDictionary 372
Базовая станция сотовой связи 539
Библиотека
◊ Bionic 29
◊ Google API 647, 681
◊ Text To Speech 529
Браузер WebKit 595, 601

В

Виджет 105
Видимый процесс 229

Виртуальная машина Dalvik 29
Всплывающее уведомление 167

Г

Горячие клавиши 206, 207
Графика NinePatch 280
Графические ресурсы 280
Графический интерфейс пользователя 83
Графический примитив 447, 449
Группа
◊ меню 222
◊ представлений 83

Д

Действие
◊ ACTION_BATTERY_LOW 769
◊ ACTION_BATTERY_OKAY 769
◊ ACTION_IMAGE_CAPTURE 709
◊ ACTION_PICK_WIFI_NETWORK 625
◊ ACTION_POWER_CONNECTED 769
◊ ACTION_POWER_DISCONNECTED 769
◊ ACTION_POWER_USAGE_SUMMARY 775
◊ ACTION_VIDEO_CAPTURE 709
◊ ACTION_WIFI_IP_SETTINGS 625
◊ ACTION_WIFI_SETTINGS 625
◊ ACTION_WIRELESS_SETTINGS 625
◊ INTENT_ACTION_STILL_IMAGE_CAMERA 709
◊ INTENT_ACTION_VIDEO_CAMERA 709

- ◇ NETWORK_STATE_CHANGED_ACTION 618
- ◇ SCAN_RESULTS_AVAILABLE_ACTION 637
- ◇ SUPPLICANT_CONNECTION_CHANGE_ACTION 619
- ◇ SUPPLICANT_STATE_CHANGED_ACTION 619
- ◇ WIFI_STATE_CHANGED_ACTION 618

Деятельность 229

Диалог 175

Документ XML 293

Дополнение

- ◇ EXTRA_SUPPLICANT_CONNECTED 619

Дополнения 240

Драйвер IPC 28

Ж

Журнал событий 76

З

Заголовок меню 206

Запрос 391

И

Идентификатор

- ◇ BSSID 619, 627

- ◇ SSID 627

Идентификатор

- ◇ группы 206

- ◇ пункта меню 206

- ◇ ресурса представления 327

Индекс ключа WEP 633

Интерфейс

- ◇ Camera.PictureCallback 723

- ◇ Camera.ShutterCallback 723

- ◇ Iterator 392

- ◇ LocationListener 658

- ◇ OnClickListener 125

- ◇ OnCreateContextMenuListener 125

- ◇ OnFocusChangeListener 125

- ◇ OnInitListener 530

- ◇ OnKeyListener 125

- ◇ OnLongClickListener 125

- ◇ OnTouchListener 125

- ◇ SensorEventListener 737

- ◇ SurfaceHolder 712

- ◇ TextBasedSmsColumns 590

- ◇ WindowManager 757

К

Карта памяти 305

Каталог

- ◇ Conversations 585

- ◇ Draft 585

- ◇ Failed 585

- ◇ Inbox 585

- ◇ Outbox 585

- ◇ Queued 585

- ◇ Sent 585

- ◇ Undelivered 585

- ◇ для хранения ключей 679

- ◇ ресурсов 58

Класс

- ◇ ActionBar 510

- ◇ ActivityManager 787

- ◇ AlarmManager 517

- ◇ AlertDialog 177

- ◇ AlphaAnimation 444

- ◇ AnalogClock 162

- ◇ AnimationDrawable 461, 463, 465

- ◇ AnimationSet 443, 445

- ◇ AppWidgetProvider 492

- ◇ AppWidgetProviderInfo 492

- ◇ ArcShape 432, 434

- ◇ ArrayAdapter 327

- ◇ ArrayAdapter<T> 326

- ◇ ArrayList 327

- ◇ AssetManager 279, 301, 302

- ◇ AutoCompleteTextView 364

- ◇ AutoFocusCallback 728

- ◇ BatteryManager 767

- ◇ BroadcastReceiver 481–485, 569, 637, 767

- ◇ Bundle 239

- ◇ Button 126

- ◇ Calendar 197

- ◇ Camera 701

- ◇ Camera.Parameters 702

- ◇ CdmaCellLocation 540

- ◇ CellLocation 539

- ◇ CheckBox 136, 137

- ◇ CheckBoxPreference 404

- ◇ Chronometer 162, 163

- ◇ ClipboardManager 525

- ◇ ConfigurationInfo 793

- ◇ ConnectivityManager 595
- ◇ ContentProvider 383, 385, 391
- ◇ ContentResolver 386, 393, 394
- ◇ ContentValues 388, 389, 393
- ◇ Context 310, 327
- ◇ Criteria 649, 653
- ◇ Cursor 387, 590
- ◇ CursorAdapter 325, 326
- ◇ DatePickerDialog 193
- ◇ DialogFragment 261
- ◇ DigitalClock 162
- ◇ Display 757
- ◇ DisplayMetrics 758
- ◇ Drawable 427–429, 432, 434, 438
- ◇ EditText 105
- ◇ EditTextPreference 404
- ◇ Environment 307
- ◇ File 307
- ◇ FileInputStream 310
- ◇ FileOutputStream 311
- ◇ Fragment 261
- ◇ FragmentTransaction 266
- ◇ FrameLayout 119
- ◇ Gallery 347, 353
- ◇ GeoPoint 684
- ◇ GridView 343
- ◇ GsmCellLocation 539
- ◇ HashMap 337
- ◇ HorizontalScrollView 119
- ◇ ImageButton 141
- ◇ ImageView 121
- ◇ Intent 481, 482, 485
- ◇ IntentFilter 232, 637
- ◇ LayoutInflater 170, 719
- ◇ List<T> 327
- ◇ ListActivity 328
- ◇ ListAdapter 328
- ◇ ListFragment 261
- ◇ ListPreference 404
- ◇ ListView 328
- ◇ Locale 664
- ◇ Location 658
- ◇ LocationManager 647
- ◇ LocationProvider 649
- ◇ Log 76
- ◇ Map 327, 388, 393
- ◇ MapActivity 683
- ◇ MapController 683
- ◇ MapView 682
- ◇ MemoryInfo 797
- ◇ Menuitem 206
- ◇ MultiAutoCompleteTextView 367
- ◇ NetworkInfo 596
- ◇ Notification 502
- ◇ NotificationManager 501
- ◇ PaintDrawable 280
- ◇ Path 433
- ◇ PhoneStateListener 549
- ◇ PowerManager 779
- ◇ Preference 606
- ◇ PreferenceFragment 261
- ◇ Process 806
- ◇ ProcessBuilder 806
- ◇ ProcessErrorStateInfo 797, 804
- ◇ ProgressBar 152
- ◇ ProgressDialog 189
- ◇ R 62, 279, 281
- ◇ RadioButton 134
- ◇ RatingBar 155, 159
- ◇ RecentTaskInfo 797, 802
- ◇ RectShape 432, 438
- ◇ ResourceCursorAdapter 326
- ◇ RingtonePreference 404
- ◇ RotateAnimation 444
- ◇ RoundRectShape 432, 433
- ◇ RunningAppProcessInfo 797
- ◇ RunningServiceInfo 797, 800
- ◇ RunningTaskInfo 797, 801
- ◇ ScaleAnimation 444
- ◇ ScrollView 119
- ◇ SeekBar 155, 156, 692
- ◇ Sensor 734
- ◇ SensorManager 735
- ◇ Service 518
- ◇ Service() 471, 472
- ◇ ServiceState 552
- ◇ Settings 625
- ◇ ShapeDrawable 427, 432–434, 438–440
- ◇ SimpleAdapter 326
- ◇ SimpleCursorAdapter 326
- ◇ SlidingDrawer 347, 356
- ◇ SmsManager 574
- ◇ SmsMessage 581
- ◇ Spinner 361
- ◇ SQLiteDatabase 376
- ◇ SQLiteOpenHelper 375
- ◇ String 111
- ◇ SurfaceView 712

Класс (*прод.*)

- ◇ TabHost 143
- ◇ TabWidget 143
- ◇ TelephonyManager 537
- ◇ TextToSpeech 529
- ◇ TextView 105
- ◇ TimePickerDialog 197
- ◇ Toast 170
- ◇ ToggleButton 139
- ◇ TrafficStats 599
- ◇ TransitionDrawable 429
- ◇ TranslateAnimation 444
- ◇ Typeface 302, 303
- ◇ View 84, 440
- ◇ ViewGroup 84
- ◇ WakeLock 780
- ◇ WallpaperManager 489
- ◇ WebView 602
- ◇ WebViewFragment 261
- ◇ WifiConfiguration 632
- ◇ WifiInfo 627
- ◇ WifiManager 617

Ключ Maps API Key 679

Команды

- ◇ Linux 806
- ◇ анимации 444

Компоновка 83

Константа

- ◇ ACCURACY_COARSE 653
- ◇ ACCURACY_FINE 653
- ◇ ACCURACY_HIGH 653
- ◇ ACCURACY_LOW 653
- ◇ ACCURACY_MEDIUM 653
- ◇ ACQUIRE_CAUSES_WAKEUP 780
- ◇ ACTION_CALL 255
- ◇ BATTERY_HEALTH_DEAD 768
- ◇ BATTERY_HEALTH_GOOD 768
- ◇ BATTERY_HEALTH_OVER_VOLTAGE 768
- ◇ BATTERY_HEALTH_OVERHEAT 768
- ◇ BATTERY_HEALTH_UNKNOWN 768
- ◇ BATTERY_HEALTH_UNSPECIFIED_FAILURE 768
- ◇ BATTERY_PLUGGED_AC 768
- ◇ BATTERY_PLUGGED_USB 768
- ◇ BATTERY_STATUS_CHARGING 768
- ◇ BATTERY_STATUS_DISCHARGING 768
- ◇ BATTERY_STATUS_FULL 768

- ◇ BATTERY_STATUS_NOT_CHARGING 768
- ◇ BATTERY_STATUS_UNKNOWN 768
- ◇ CALL_STATE_IDLE 540
- ◇ CALL_STATE_OFFHOOK 540
- ◇ CALL_STATE_RINGING 540
- ◇ CHOICE_MODE_MULTIPLE 335
- ◇ CHOICE_MODE_SINGLE 334
- ◇ CLIPBOARD_SERVICE 525
- ◇ CONNECTIVITY_SERVICE 595
- ◇ DEFAULT_ALL 502
- ◇ DEFAULT_LIGHTS 502
- ◇ DEFAULT_SOUND 502
- ◇ DEFAULT_VIBRATE 502
- ◇ DENSITY_HIGH 758
- ◇ DENSITY_LOW 758
- ◇ DENSITY_MEDIUM 758
- ◇ DENSITY_XHIGH 758
- ◇ DIRECTORY_ALARMS 307
- ◇ DIRECTORY_DCIM 307
- ◇ DIRECTORY_DOWNLOADS 307
- ◇ DIRECTORY_MOVIES 307
- ◇ DIRECTORY_MUSIC 307
- ◇ DIRECTORY_NOTIFICATIONS 308
- ◇ DIRECTORY_PICTURES 308
- ◇ DIRECTORY_PODCASTS 308
- ◇ DIRECTORY_RINGTONES 308
- ◇ ELAPSED_REALTIME 518
- ◇ ELAPSED_REALTIME_WAKEUP 518
- ◇ ERROR 530
- ◇ FULL_WAKE_LOCK 780
- ◇ GPS_PROVIDER 649
- ◇ GRAVITY_DEATH_STAR_I 749
- ◇ GRAVITY_MERCURY 749
- ◇ GRAVITY_MOON 749
- ◇ GRAVITY_VENUS 749
- ◇ KEYBOARD_12KEY 794
- ◇ KEYBOARD_NOKEYS 794
- ◇ KEYBOARD_QWERTY 794
- ◇ KEYBOARD_UNDEFINED 794
- ◇ LENGTH_LONG 168
- ◇ LENGTH_SHORT 168
- ◇ LIGHT_CLOUDY 739
- ◇ LIGHT_FULLMOON 739
- ◇ LIGHT_NO_MOON 739
- ◇ LIGHT_OVERCAST 739
- ◇ LIGHT_SHADE 739
- ◇ LIGHT_SUNLIGHT 739
- ◇ LIGHT_SUNLIGHT_MAX 739

- ◇ LIGHT_SUNRISE 739
- ◇ LISTEN_CALL_FORWARDING_INDICATOR 550
- ◇ LISTEN_CALL_STATE 550
- ◇ LISTEN_CELL_LOCATION 550
- ◇ LISTEN_DATA_ACTIVITY 551
- ◇ LISTEN_DATA_CONNECTION_STATE 551
- ◇ LISTEN_MESSAGE_WAITING_INDICATOR 551
- ◇ LISTEN_NONE 551
- ◇ LISTEN_SERVICE_STATE 551
- ◇ LISTEN_SIGNAL_STRENGTHS 551
- ◇ LOCATION_SERVICE 647
- ◇ MAGNETIC_FIELD_EARTH_MAX 753
- ◇ MAGNETIC_FIELD_EARTH_MIN 753
- ◇ MEDIA_BAD_REMOVAL 310
- ◇ MEDIA_CHECKING 310
- ◇ MEDIA_MOUNTED 310
- ◇ MEDIA_MOUNTED_READ_ONLY 310
- ◇ MEDIA_NOFS 310
- ◇ MEDIA_REMOVED 310
- ◇ MEDIA_SHARED 310
- ◇ MEDIA_UNMOUNTABLE 310
- ◇ MEDIA_UNMOUNTED 310
- ◇ NAVIGATION_DPAD 794
- ◇ NAVIGATION_NONAV 794
- ◇ NAVIGATION_TRACKBALL 794
- ◇ NAVIGATION_UNDEFINED 794
- ◇ NAVIGATION_WHEEL 794
- ◇ NETWORK_PROVIDER 649
- ◇ NETWORK_TYPE_1xRTT 538
- ◇ NETWORK_TYPE_CDMA 538
- ◇ NETWORK_TYPE_EDGE 538
- ◇ NETWORK_TYPE_EHRPD 538
- ◇ NETWORK_TYPE_EVDO_0 538
- ◇ NETWORK_TYPE_EVDO_A 538
- ◇ NETWORK_TYPE_EVDO_B 538
- ◇ NETWORK_TYPE_GPRS 538
- ◇ NETWORK_TYPE_HSDPA 539
- ◇ NETWORK_TYPE_HSPA 539
- ◇ NETWORK_TYPE_HSUPA 539
- ◇ NETWORK_TYPE_IDEN 539
- ◇ NETWORK_TYPE_LTE 538
- ◇ NETWORK_TYPE_UMTS 539
- ◇ NETWORK_TYPE_UNKNOWN 539
- ◇ NO_REQUIREMENT 653
- ◇ ON_AFTER_RELEASE 780
- ◇ PARTIAL_WAKE_LOCK 780
- ◇ PASSIVE_PROVIDER 650
- ◇ PHONE_TYPE_CDMA 538
- ◇ PHONE_TYPE_GSM 538
- ◇ PHONE_TYPE_NONE 538
- ◇ PHONE_TYPE_SIP 538
- ◇ POWER_HIGH 653
- ◇ POWER_LOW 653
- ◇ POWER_MEDIUM 653
- ◇ QUEUE_ADD 529
- ◇ QUEUE_FLUSH 530
- ◇ RECENT_IGNORE_UNAVAILABLE 803
- ◇ RECENT_WITH_EXCLUDED 803
- ◇ RESULT_CANCELED 241
- ◇ RESULT_ERROR_GENERIC_FAILURE 575
- ◇ RESULT_ERROR_NO_SERVICE 575
- ◇ RESULT_ERROR_NULL_PDU 575
- ◇ RESULT_ERROR_RADIO_OFF 575
- ◇ RESULT_FIRST_USER 241
- ◇ RESULT_OK 241, 575
- ◇ RTC 518
- ◇ RTC_WAKEUP 518
- ◇ SCREEN_BRIGHT_WAKE_LOCK 780
- ◇ SCREEN_DIM_WAKE_LOCK 780
- ◇ SENSOR_DELAY_FASTEST 738
- ◇ SENSOR_DELAY_GAME 738
- ◇ SENSOR_DELAY_NORMAL 738
- ◇ SENSOR_DELAY_UI 738
- ◇ SENSOR_STATUS_ACCURACY_HIGH 737
- ◇ SENSOR_STATUS_ACCURACY_LOW 737
- ◇ SENSOR_STATUS_ACCURACY_MEDIUM 737
- ◇ SENSOR_STATUS_UNRELIABLE 737
- ◇ SHOW_AS_ACTION_IF_ROOM 512
- ◇ SHOW_AS_ACTION_WITH_TEXT 516
- ◇ SIM_STATE_ABSENT 547
- ◇ SIM_STATE_NETWORK_LOCKED 547
- ◇ SIM_STATE_PIN_REQUIRED 547
- ◇ SIM_STATE_PUK_REQUIRED 547
- ◇ SIM_STATE_READY 547
- ◇ SIM_STATE_UNKNOWN 547
- ◇ simple_list_item_multiple_choice 335
- ◇ simple_list_item_single_choice 334
- ◇ STANDARD_GRAVITY 749
- ◇ STATE_EMERGENCY_ONLY 552
- ◇ STATE_IN_SERVICE 552
- ◇ STATE_OUT_OF_SERVICE 553
- ◇ STATE_POWER_OFF 553
- ◇ STATUS_ON_ICC_FREE 581
- ◇ STATUS_ON_ICC_READ 581

Константа (*prod.*)

- ◇ STATUS_ON_ICC_SEND 581
- ◇ STATUS_ON_ICC_UNREAD 581
- ◇ STATUS_ON_ICC_UNSENT 581
- ◇ SUCCESS 530
- ◇ TOUCHSCREEN_FINGER 794
- ◇ TOUCHSCREEN_NOTOUCH 795
- ◇ TOUCHSCREEN_STYLUS 794
- ◇ TOUCHSCREEN_UNDEFINED 795
- ◇ TYPE_ACCELEROMETER 734
- ◇ TYPE_ALL 734
- ◇ TYPE_GRAVITY 734
- ◇ TYPE_GYROSCOPE 734
- ◇ TYPE_LIGHT 734
- ◇ TYPE_LINEAR_ACCELERATION 734
- ◇ TYPE_MAGNETIC_FIELD 734
- ◇ TYPE_ORIENTATION 734
- ◇ TYPE_PRESSURE 734
- ◇ TYPE_PROXIMITY 734
- ◇ TYPE_ROTATION_VECTOR 734
- ◇ TYPE_TEMPERATURE 734
- ◇ WIFI_STATE_DISABLED 618
- ◇ WIFI_STATE_DISABLING 618
- ◇ WIFI_STATE_ENABLED 618
- ◇ WIFI_STATE_ENABLING 618
- ◇ WIFI_STATE_UNKNOWN 618
- ◇ WINDOW_SERVICE 757

Контейнерные виджеты 325

Контекст

- ◇ Activity 450
- ◇ приложения 167
- Контекстное меню 205, 215

Л

Локализованные строковые ресурсы 423

Локализованный строковый файл 423

М

Манифест 63, 297, 298

- ◇ приложения 299
- Меню 205, 280, 289, 291, 292, 297
- ◇ выбора опций 205
- ◇ со значками 205, 210

Метод

- ◇ acquire(), класс PowerManager 780
- ◇ allowedAuthAlgorithms() 633
- ◇ allowedGroupCiphers() 633
- ◇ allowedKeyManagement() 633
- ◇ allowedPairwiseCiphers() 633
- ◇ allowedProtocols() 633
- ◇ animateTo() 683
- ◇ beginTransaction() 266
- ◇ bindService() 471
- ◇ calculateSignalLevel() 642
- ◇ cancel(), класс AlarmManager 517
- ◇ cancel(), класс NotificationManager 501
- ◇ cancelAll(), класс NotificationManager 501
- ◇ commit() 266
- ◇ compareSignalLevel() 642
- ◇ delete() 384, 385, 389, 390, 393, 394
- ◇ displayZoomControls() 690
- ◇ doNegativeClick() 274
- ◇ doPositiveClick() 274
- ◇ getAccuracy() 649, 653, 658
- ◇ getActionBar() 509
- ◇ getActiveNetworkInfo() 595
- ◇ getAddressLine() 664
- ◇ getAllNetworkInfo() 595
- ◇ getAllProviders() 649
- ◇ getAltitude() 658
- ◇ getBaseStationId() 540
- ◇ getBaseStationLatitude() 540
- ◇ getBaseStationLongitude() 540
- ◇ getBearingAccuracy() 653
- ◇ getBestProvider() 652
- ◇ getBSSID() 627
- ◇ getCallState() 540
- ◇ getCdmaDbm() 551
- ◇ getCdmaEcio() 552
- ◇ getCellLocation() 539
- ◇ getCid() 539
- ◇ getColumnName() 590
- ◇ getCountry() 664
- ◇ getCountryCode() 663
- ◇ getCountryName() 663
- ◇ getDataDirectory() 307
- ◇ getDefault() 575
- ◇ getDefaultDisplay() 757
- ◇ getDefaultSensor() 735
- ◇ getDeviceConfigurationInfo() 793
- ◇ getDhcpInfo() 627
- ◇ getDisplayCountry() 664
- ◇ getDisplayId() 758
- ◇ getDisplayLanguage() 664
- ◇ getDisplayMessageBody() 581
- ◇ getDisplayName() 664

- ◇ getDisplayOriginatingAddress() 581
- ◇ getDownloadCacheDirectory() 307
- ◇ getEvdoDbm() 552
- ◇ getEvdoEcio() 552
- ◇ getEvdoSnr() 552
- ◇ getExternalStorageDirectory() 307
- ◇ getExternalStorageState() 310
- ◇ getFeatureName() 664
- ◇ getFromLocationName() 672
- ◇ getGsmBitErrorRate() 552
- ◇ getGsmSignalStrength() 552
- ◇ getHeight() 758
- ◇ getHolder() 712
- ◇ getHorizontalAccuracy() 653
- ◇ getIndexOnIcc() 581
- ◇ getInstance() 489
- ◇ getIsManualSelection() 553
- ◇ getISO3Country() 664
- ◇ getISO3Language() 664
- ◇ getItemId() 206
- ◇ getLac() 539
- ◇ getLanguage() 530, 664
- ◇ getLastKnownLocation() 658
- ◇ getLatitude() 658
- ◇ getLatitudeE6() 684
- ◇ getLinkSpeed() 627
- ◇ getLocality() 664
- ◇ getLongitude() 658
- ◇ getLongitudeE6() 684
- ◇ getMaximumRange() 734
- ◇ getMaxZoomLevel() 682
- ◇ getMemoryInfo() 797
- ◇ getMessageBody() 581
- ◇ getMessageClass() 581
- ◇ getMinDelay() 734
- ◇ getMobileRxBytes() 599
- ◇ getMobileRxPackets() 599
- ◇ getMobileTxBytes() 599
- ◇ getMobileTxPackets() 599
- ◇ getName() 649, 734
- ◇ getNetworkId() 540
- ◇ getNetworkInfo() 595
- ◇ getNetworkType() 538
- ◇ getOnChronometerTickListener() 163
- ◇ getOperatorAlphaLong() 553
- ◇ getOperatorAlphaShort() 553
- ◇ getOperatorNumeric() 553
- ◇ getOriginatingAddress() 581
- ◇ getParameters(), класс Camera 702
- ◇ getPdu() 581
- ◇ getPhone() 664
- ◇ getPhoneType() 537
- ◇ getPosition() 392
- ◇ getPostalCode() 664
- ◇ getPower() 734
- ◇ getPowerRequirement() 649, 653
- ◇ getProcessesInErrorState() 804
- ◇ getProcessMemoryInfo() 797
- ◇ getProtocolIdentifier() 581
- ◇ getProvider() 649
- ◇ getProviders() 649
- ◇ getPsc() 540
- ◇ getRefreshRate() 758
- ◇ getResolution() 734
- ◇ getRoaming() 553
- ◇ getRootDirectory() 307
- ◇ getRssi() 642
- ◇ getRunningAppProcesses() 799
- ◇ getRunningServices() 800
- ◇ getRunningTasks() 801
- ◇ getSensorList() 735
- ◇ getServiceCenterAddress() 581
- ◇ getSimCountryIso() 546
- ◇ getSimOperator() 546
- ◇ getSimOperatorName() 546
- ◇ getSimSerialNumber() 546
- ◇ getSimState() 546, 547
- ◇ getSpeedAccuracy() 653
- ◇ getSSID() 627
- ◇ getState() 552
- ◇ getStatus() 581
- ◇ getStatusOnIcc() 581
- ◇ getSubAdminArea() 663
- ◇ getSubtype() 596
- ◇ getSubtypeName() 596
- ◇ getSupportedColorEffects() 704
- ◇ getSupportedFlashModes() 705
- ◇ getSupportedFocusModes() 705
- ◇ getSupportedJpegThumbnailSizes() 705
- ◇ getSupportedPictureFormats() 705
- ◇ getSupportedPictureSizes() 705
- ◇ getSupportedPreviewFormats() 705
- ◇ getSupportedPreviewFpsRange() 705
- ◇ getSupportedPreviewSizes() 705
- ◇ getSupportedSceneModes() 705
- ◇ getSupportedVideoSizes() 705
- ◇ getSupportedWhiteBalance() 705
- ◇ getSystemId() 540

Метод (*мод.*)

- ◇ getSystemService() 537
- ◇ getText() 110
- ◇ getText(), класс ClipboardManager 525
- ◇ getTimestampMillis() 581
- ◇ getTotalRxBytes() 599
- ◇ getTotalTxBytes() 599
- ◇ getTotalTxPackets() 599
- ◇ getType() 384, 596
- ◇ getTypeName() 596
- ◇ getUidRxBytes() 599
- ◇ getUidTxBytes() 599
- ◇ getUserData() 581
- ◇ getVendor() 734
- ◇ getVersion() 734
- ◇ getVerticalAccuracy() 653
- ◇ getWidth() 758
- ◇ getZoomLevel() 682
- ◇ goToSleep() 779
- ◇ hasMonetaryCost() 649
- ◇ hasText(), класс ClipboardManager 526
- ◇ insert() 383, 385, 388, 393
- ◇ invalidate() 438
- ◇ isAltitudeRequired() 653
- ◇ isAvailable() 596
- ◇ isBearingRequired() 653
- ◇ isConnected() 596
- ◇ isConnectedOrConnecting() 596
- ◇ isCostAllowed() 653
- ◇ isDirectory() 307
- ◇ isFile() 307
- ◇ isLanguageAvailable() 530
- ◇ isNetworkRoaming() 540
- ◇ isRoaming() 596
- ◇ isSatellite() 682
- ◇ isScreenOn() 779
- ◇ isSpeaking() 529
- ◇ isSpeedRequired() 653
- ◇ isStreetView() 682
- ◇ isTraffic() 682
- ◇ loadData() 605
- ◇ loadDataWithBaseUrl() 605
- ◇ loadUrl() 602
- ◇ makeText() 167
- ◇ meetsCriteria() 649
- ◇ moveToFirst() 392
- ◇ moveToLast() 392
- ◇ moveToNext() 392
- ◇ moveToPosition() 392
- ◇ moveToPrevious() 392
- ◇ newWakeLock() 780
- ◇ notify(), класс NotificationManager 501
- ◇ onActivityCreated() 262
- ◇ onActivityResult() 241
- ◇ onAttach() 262
- ◇ onCallForwardingIndicatorChanged() 550
- ◇ onCallStateChanged() 549
- ◇ onCellLocationChanged() 549, 552
- ◇ onContextItemSelected() 216
- ◇ onCreate() 230, 262, 375, 471
- ◇ onCreateDialog() 176, 190
- ◇ onCreateOptionsMenu() 206
- ◇ onCreateView() 262
- ◇ onDataActivity() 550
- ◇ onDataConnectionStateChanged() 550
- ◇ onDestroy() 230, 262, 471
- ◇ onDestroyView() 262
- ◇ onDetach() 262
- ◇ onItemSelected() 346
- ◇ onListItemClick() 591
- ◇ onLocationChanged() 658
- ◇ onMessageWaitingIndicatorChanged() 550
- ◇ onNothingSelected() 346
- ◇ onOptionsItemSelected() 206, 211, 213, 214, 221, 223, 225
- ◇ onPause() 230, 262
- ◇ onPrepareDialog() 177
- ◇ onProgressChanged() 156
- ◇ onProviderDisabled() 658
- ◇ onProviderEnabled() 658
- ◇ onRebind() 472
- ◇ onReceive() 481–485
- ◇ onRestart() 230
- ◇ onRestoreInstanceState() 239
- ◇ onResume() 230, 262
- ◇ onSaveInstanceState() 239
- ◇ onSensorChanged() 737
- ◇ onServiceStateChanged() 550, 552
- ◇ onSignalStrengthsChanged() 550, 551
- ◇ onStart() 230, 262, 471
- ◇ onStartTrackingTouch() 156
- ◇ onStatusChanged() 658
- ◇ onStop() 230, 262
- ◇ onStopTrackingTouch() 156
- ◇ onUpgrade() 375
- ◇ open(), класс Camera 701
- ◇ openFileInput() 312
- ◇ openFileOutput() 312

- ◇ preSharedKey() 633
- ◇ query() 383, 385, 387, 389, 390
- ◇ reboot() 779
- ◇ registerListener() 738
- ◇ release(), класс Camera 701
- ◇ release(), класс PowerManager 780
- ◇ requestLocationUpdates() 659
- ◇ requestSingleUpdate() 659
- ◇ requiresCell() 649
- ◇ requiresNetwork() 649
- ◇ requiresSatellite() 649
- ◇ sendBroadcast() 481
- ◇ sendMultipartTextMessage() 576
- ◇ set(), класс AlarmManager 517
- ◇ setAccuracy() 653
- ◇ setBuiltInZoomControls() 606, 690
- ◇ setCenter() 683
- ◇ setCheckable() 222
- ◇ setChoiceMode() 334
- ◇ setCursiveFontFamily() 606
- ◇ setDefaultFixedFontSize() 606
- ◇ setDefaultFontSize() 606
- ◇ setDefaultZoom() 606
- ◇ setDisplayZoomControls() 606
- ◇ setFixedFontFamily() 606
- ◇ setInexactRepeating 517
- ◇ setItems() 184
- ◇ setLanguage() 530
- ◇ setListAdapter() 328
- ◇ setMinimumFontSize() 606
- ◇ setMinimumLogicalFontSize() 606
- ◇ setMultiChoiceItems() 187
- ◇ setOnChronometerTickListener() 163
- ◇ setOnClickListener() 129
- ◇ setOnItemSelectedListener() 346
- ◇ setParameters(), класс Camera 702
- ◇ setPowerRequirement() 653
- ◇ setReferenceCounted(), класс PowerManager 781
- ◇ setRepeating() 517
- ◇ setSatellite() 682
- ◇ setShowAsAction 512
- ◇ setSingleChoiceItems() 183
- ◇ setStreetView() 682
- ◇ setSupportZoom() 606
- ◇ setText(), класс ClipboardManager 525
- ◇ setTextSize() 606
- ◇ setTime(), класс AlarmManager 517
- ◇ setTimeZone(), класс AlarmManager 517

- ◇ setTraffic() 682
- ◇ setZoom() 683
- ◇ showDialog() 176
- ◇ shutdown(), класс TextToSpeech 529
- ◇ speak() 529
- ◇ startActivity() 668
- ◇ startScan() 636
- ◇ stop(), класс TextToSpeech 529
- ◇ stopAnimation() 683
- ◇ stopService() 473
- ◇ supportsAltitude() 649
- ◇ supportsSpeed() 649
- ◇ surfaceChanged() 712
- ◇ surfaceCreated() 712
- ◇ surfaceDestroyed() 712
- ◇ synthesizeToFile() 529
- ◇ takePicture() 723
- ◇ update() 384, 385, 389, 390, 393
- ◇ wepKeys() 633
- ◇ wepTxKeyIndex() 633
- ◇ zoomIn() 683
- ◇ zoomInFixing() 683
- ◇ zoomOutFixing() 683
- ◇ zoomToSpan() 684
- Метрики дисплея 758

O

Объект Intent 564

P

Пакет

- ◇ android.app 787
- ◇ android.location 647, 658
- ◇ android.net 595
- ◇ android.net.http 595
- ◇ android.net.wifi 617
- ◇ android.os 307
- ◇ android.telephony 537
- ◇ android.view 757
- ◇ java.io.File 307
- ◇ java.net 595

Параметр

- ◇ Azimuth 744
- ◇ EXTRA_BSSID 619
- ◇ EXTRA_HEALTH 768
- ◇ EXTRA_ICON_SMALL 768
- ◇ EXTRA_LEVEL 767

Параметр (прод.)

- ◇ EXTRA_NETWORK_INFO 618
- ◇ EXTRA_NEW_STATE 619
- ◇ EXTRA_PRESENT 767
- ◇ EXTRA_PREVIOUS_WIFI_STATE 618
- ◇ EXTRA_SCALE 767
- ◇ EXTRA_SUPPLICANT_ERROR 619
- ◇ EXTRA_TECHNOLOGY 768
- ◇ EXTRA_TEMPERATURE 767
- ◇ EXTRA_VOLTAGE 767
- ◇ Lateral 749
- ◇ Longitudinal 749
- ◇ Pitch 744
- ◇ Roll 744
- ◇ Vertical 749
- Перерисовка 438, 440
- Плагин ADT 39, 57
- Плотность пикселей 758
- Подменю 219
- Позиционирование курсора 392
- Покадровая анимация 461
- Поле
 - ◇ _id, интерфейс TextBasedSmsColumns 590
 - ◇ address, интерфейс TextBasedSmsColumns 590
 - ◇ availMem 797
 - ◇ baseActivity 801
 - ◇ baseIntent 803
 - ◇ body, интерфейс TextBasedSmsColumns 590
 - ◇ condition, класс ProcessErrorStateInfo 804
 - ◇ date, интерфейс TextBasedSmsColumns 590
 - ◇ densityDpi 758
 - ◇ description, класс RecentTaskInfo 803
 - ◇ description, класс RunningTaskInfo 801
 - ◇ dns1 627
 - ◇ dns2 628
 - ◇ error_code, интерфейс TextBasedSmsColumns 590
 - ◇ flags, класс Notification 503
 - ◇ frequency 637
 - ◇ gateway 627
 - ◇ heightPixels 758
 - ◇ id, класс RecentTaskInfo 803
 - ◇ id, класс RunningTaskInfo 801
 - ◇ interval 518
 - ◇ ipAddress 627
 - ◇ level 637
 - ◇ locked, интерфейс TextBasedSmsColumns 590
 - ◇ longMsg, класс ProcessErrorStateInfo 804
 - ◇ lowMemory 797
 - ◇ netmask 628
 - ◇ numActivities 801
 - ◇ numRunning 801
 - ◇ operation 518
 - ◇ origActivity 803
 - ◇ person, интерфейс TextBasedSmsColumns 590
 - ◇ pid, класс ProcessErrorStateInfo 804
 - ◇ priority, класс WifiConfiguration 633
 - ◇ processName, класс ProcessErrorStateInfo 804
 - ◇ protocol, интерфейс TextBasedSmsColumns 590
 - ◇ read, интерфейс TextBasedSmsColumns 590
 - ◇ reply_path_present 590
 - ◇ reqGlesVersion 793
 - ◇ reqInputFeatures 793
 - ◇ reqKeyboardType 794
 - ◇ reqNavigation 794
 - ◇ reqTouchScreen 794
 - ◇ scaledDensity 758
 - ◇ serverAddress 628
 - ◇ service_center, интерфейс TextBasedSmsColumns 590
 - ◇ shortMsg, класс ProcessErrorStateInfo 804
 - ◇ stackTrace 804
 - ◇ status, интерфейс TextBasedSmsColumns 590
 - ◇ status, класс WifiConfiguration 633
 - ◇ subject 590
 - ◇ tag, класс ProcessErrorStateInfo 804
 - ◇ thread_id, интерфейс TextBasedSmsColumns 590
 - ◇ threshold 797
 - ◇ topActivity 801
 - ◇ triggerAtTime 518
 - ◇ type, интерфейс TextBasedSmsColumns 590
 - ◇ uid, класс ProcessErrorStateInfo 804
 - ◇ widthPixels 758
 - ◇ xdpi 758
 - ◇ ydpi 758
 - ◇ capabilities 637
- Полосы прокрутки 119
- Представление 83, 440
- Преобразования 444
- Программный стек 27, 28
- Протокол
 - ◇ аутентификации 633
 - ◇ шифрования 633

Процесс 227
Пункт меню 222
Пустой процесс 229

Р

Разрешение

- ◇ ACCESS_COARSE_LOCATION 541, 552, 650
- ◇ ACCESS_FINE_LOCATION 650
- ◇ ACCESS_WIFI_STATE 618
- ◇ BATTERY_STATS 769
- ◇ CALL_PHONE 564
- ◇ CALL_PRIVILEGED 564
- ◇ CAMERA 702
- ◇ CHANGE_WIFI_STATE 618
- ◇ GET_TASKS 801
- ◇ INTERNET 603
- ◇ MODIFY_PHONE_STATE 564
- ◇ PROCESS_OUTGOING_CALLS 564, 570
- ◇ READ_PHONE_STATE 541, 547, 564
- ◇ RECEIVE_SMS 576
- ◇ SEND_SMS 576
- ◇ WAKE_LOCK 781
- ◇ WRITE_SETTINGS 762
- ◇ WRITE_SMS 576

Разрешения 256

Расширенное меню 205, 212, 215

Режим отладки USB 77

Ресурс 423

Ресурсы 279

Рисование на канве 438

Роуминг 540

С

Связывание данных 326

Сервис

- ◇ Geocoding 663
- ◇ Google Maps 679

Сервисный процесс 229

Система

- ◇ глобального позиционирования 648
- ◇ управления энергопотреблением 28

Системная библиотека C 29

Системный идентификатор ресурса 327

Сканирование точек доступа 636

Состояние соединения 618

Статическое связывание данных 327

Стек Activity 239

Стили 280, 296
Стиль Theme.Holo.NoActionBar 509

Т

Текстовое поле с автозаполнением 364

Темы 279, 280, 296–298, 300

Терминал 806

Технология

- ◇ 1xRTT 538
- ◇ CDMA 538
- ◇ EDGE 538
- ◇ EHRPD 538
- ◇ EVDO_0 538
- ◇ GPRS 538
- ◇ GSM 538
- ◇ HSDPA 539
- ◇ HSPA 539
- ◇ HSUPA 539
- ◇ IDEN 539
- ◇ LTE 538
- ◇ SIP 538
- ◇ UMTS 539

Тип экрана 794

Точка доступа 617

У

Уведомление 167

Уровень

- ◇ API 39
- ◇ библиотек и среды выполнения 27
- ◇ каркаса приложений 27
- ◇ приложений 27
- ◇ принимаемого сигнала 551
- ◇ ядра 27

Ф

Файл

- ◇ R.java 61
- ◇ манифеста 64
 - приложения 541

Флаг

- ◇ DEFAULT_LIGHTS 502
- ◇ DEFAULT_SOUND 502
- ◇ DEFAULT_VIBRATE 502
- ◇ FLAG_AUTO_CANCEL 503
- ◇ FLAG_FOREGROUND_SERVICE 503
- ◇ FLAG_HIGH_PRIORITY 503
- ◇ FLAG_INSISTENT 503

Флаг (*прод.*)

- ◇ FLAG_NO_CLEAR 503
- ◇ FLAG_ONGOING_EVENT 503
- ◇ FLAG_ONLY_ALERT_ONCE 503
- ◇ FLAG_SHOW_LIGHTS 503
- ◇ INPUT_FEATURE_FIVE_WAY_NAV 793
- ◇ INPUT_FEATURE_HARD_KEYBOARD 793

Фоновый процесс 229

Фрагмент 261

Функциональные библиотеки C/C++ 29

Ш

Шифрование AES 633

Э

Экран

- ◇ емкостной 794
- ◇ резистивный 794
- ◇ сенсорный 794

Элемент

- ◇ appwidget-provider 493
 - ◇ fragment 263
- Эмулятор Android 77, 545, 561

Я

Ядро Android 28