АЛЕКСЕЙ ГОЛОЩАПОВ

bhv®

Google Android

системные компоненты и сетевые коммуникации

ДОСТУП К СИСТЕМНЫМ КОМПОНЕНТАМ

ЛОКАЛЬНЫЕ СЛУЖБЫ

СЕТЕВЫЕ КОММУНИКАЦИИ

СЕРВИСЫ GOOGLE

РАБОТА С ОБОРУДОВАНИЕМ МОБИЛЬНОГО ТЕЛЕФОНА

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Материалы на www.bhv.ru

Алексей Голощапов

Google Android

системные компоненты и сетевые коммуникации

Санкт-Петербург
«БХВ-Петербург»
2012

УДК 681.3.06
ББК 32.973.26-018.2
Г61

Голощапов А. Л.

Г61 Google Android: системные компоненты и сетевые коммуникации. — СПб.: БХВ-Петербург, 2012. — 384 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0666-3

Книга посвящена разработке программ для мобильных устройств под управлением операционной системы Android. Рассматривается создание приложений с использованием системных компонентов и служб Android, управление сетевыми соединениями и коммуникация через сотовую сеть, мобильный Интернет, Wi-Fi. Описана работа с оборудованием мобильного устройства Android: встроенными датчиками, картой памяти, видеокамерой, дисплеем, управление энергопотреблением телефона. Показано использование сетевых сервисов Google в пользовательских приложениях: определение координат, навигация, Geocoding, карты Google Map. Рассматриваемые в книге примеры приложений можно скачать по ссылке: ftp://85.249.45.166/9785977506663.zip и на странице книги на сайте www.bhv.ru.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Нина Седых*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.08.11.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 30,96.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12.

ISBN 978-5-9775-0666-3

Оглавление

Введение **1**

На кого рассчитана эта книга 1
Краткое описание глав 2
Исходные коды примеров 5
Благодарности 5

ЧАСТЬ I. ОБЩИЕ СВЕДЕНИЯ **7**

Глава 1. Подключение мобильного устройства для тестирования и отладки приложений **9**

Среда разработки и Android SDK 9
Настройка мобильного телефона для отладки приложений 10
Установка режима отладки через USB 11
Установка драйвера USB 12
Взаимодействие мобильного телефона с DDMS 12
Запуск и инсталляция проекта на мобильный телефон из IDE Eclipse 13
Резюме 14

Глава 2. Доступ к системным компонентам и сетевым сервисам **15**

Компоненты системы Android 15
Системные службы 15
Объекты *Intent* 18
Intent-фильтры 18
Системные контент-провайдеры 19
Встроенные базы данных 19
Установка требуемых разрешений в приложении 20
Резюме 21

ЧАСТЬ II. БАЗОВЫЕ ФУНКЦИИ ТЕЛЕФОНА И СЕТЕЙ СОТОВОЙ СВЯЗИ 23

Глава 3. Получение информации о телефоне и сети сотовой связи 25

Информация о телефоне 25
Определение типа телефона и сети сотовой связи 25
Определение базовой станции сотовой связи 27
Определение состояния вызова 28
Получение информации о роуминге 28
Использование класса *TelephonyManager* в приложении 28
Доступ к SIM-карте 34
Состояние SIM-карты 35
Доступ к SIM-карте из приложения 35
Перехват изменений состояния параметров телефона 37
Запуск и остановка прослушивания изменений состояния сотовой сети 38
Изменение уровня сигнала 39
Изменение базовой станции сотовой связи 40
Мониторинг состояния подключения к сервису 40
Приложение для прослушивания изменений состояния сотовой сети 41
Использование эмулятора для тестирования приложений 46
Резюме 48

Глава 4. Обработка телефонных вызовов 49

Использование эмулятора для тестирования обработки телефонных вызовов 49
Имитация телефонного вызова из DDMS 49
Имитация телефонного вызова между двумя эмуляторами Android 51
Установка разрешений 52
Использование объектов *Intent* для создания телефонных вызовов 53
Вызов телефонного абонента из приложения 54
Перехват исходящих звонков 58
Резюме 61

Глава 5. Отправка и получение SMS приложением 62

Использование эмулятора для отправки SMS 62
Отправка SMS из приложения 64
Отправка SMS с данными 65
Деление SMS на фрагменты 65
Установка разрешений для работы SMS 65
Приложение для отправка SMS 66
Структура SMS-сообщения 70
Перехват входящих SMS-сообщений приложением 71

Хранение SMS на мобильном устройстве ... 74
Доступ к каталогам SMS ... 74
Доступ к полям SMS-сообщения ... 79
Резюме ... 82

ЧАСТЬ III. СЕТЕВЫЕ КОММУНИКАЦИИ ... 83

Глава 6. Мобильный Интернет ... 85

Создание сетевых соединений ... 85
Менеджер сетевых соединений ... 85
Характеристики мобильной сети ... 86
Получение информации о сети в приложении ... 86
Мониторинг сетевого трафика ... 89
Получение информации о трафике ... 89
Приложение для мониторинга сетевого трафика ... 90
Встроенный браузер ... 92
Виджет *WebView* ... 93
Использование виджета *WebView* ... 93
Загрузка данных в виджет *WebView* ... 95
Сохранение пользовательских настроек ... 97
Резюме ... 106

Глава 7. Управление Wi-Fi соединениями ... 107

Управление соединением Wi-Fi ... 107
Менеджер Wi-Fi соединений ... 107
Разрешения ... 108
Состояние соединения ... 108
Отслеживание состояния соединения ... 108
Управление подключением Wi-Fi и отслеживание состояния соединения из приложения ... 110
Управление настройками Wi-Fi соединения ... 115
Характеристики соединения ... 118
IP-адресация ... 118
Получение информации о сети Wi-Fi в приложении ... 119
Конфигурация Wi-Fi соединения ... 123
Сканирование точек доступа ... 128
Мониторинг уровня сигнала и скорости передачи данных в приложении ... 133
Резюме ... 137

ЧАСТЬ IV. МЕСТОПОЛОЖЕНИЕ И НАВИГАЦИЯ 139

Глава 8. Определение местоположения 141

Использование Google API в эмуляторе 141
Сервисы и провайдеры местоположения 141
Типы провайдеров местоположения 142
Разрешения для работы с провайдерами местоположения 144
Приложение для поиска доступных провайдеров 144
Определение лучшего провайдера 146
Критерии для определения лучшего провайдера 147
Поиск и определение лучшего привайдера в приложении 148
Использование эмулятора Android для тестирования приложений 151
Определение координат 152
Обновление местоположения 153
Приложение для мониторинга изменений координат и состояния провайдера 154
Резюме 157

Глава 9. Сервис Geocoding 158

Использование Geocoding 158
Reverse Geocoding 159
Отображение местоположения на карте 163
Forward Geocoding 168
Резюме 173

Глава 10. Использование карт Google Maps в приложениях 174

Получение ключа Maps API Key 174
Базовые классы 176
Виджет *MapView* 177
Класс *MapActivity* 178
Класс *MapController* 178
Класс *GeoPoint* 179
Использование *MapView* в приложении 180
Управление масштабированием карты 185
Добавление маркера 186
Изменение масштаба карты с помощью виджета *SeekBar* 187
Резюме 194

ЧАСТЬ V. РАБОТА С ОБОРУДОВАНИЕМ МОБИЛЬНОГО УСТРОЙСТВА 195

Глава 11. Карта памяти и файловая система 197

Подключение карты памяти в эмуляторе 197
Файловая система Android 198
Стандартные директории Android 199
Проверка состояния карты памяти 202
Сохранение и чтение файлов с SD-карты 203
Резюме 212

Глава 12. Использование видеокамеры 213

Работа с камерой в приложении 213
  Параметры камеры 214
  Получение параметров камеры в приложении 214
Поддержка различных режимов камерой 216
Использование объектов *Intent* для открытия камеры 221
Встраивание камеры в приложения 224
Управление работой камеры 227
Добавление оверлеев 231
Захват изображения 236
Использование автофокуса 241
Резюме 246

Глава 13. Встроенные датчики 247

Библиотека для работы с датчиками 247
  Управление датчиками 247
  Поиск доступных датчиков на мобильном устройстве 249
  Отслеживание изменений измеряемых датчиками значений 251
Работа с датчиками в приложении 252
  Датчик освещенности 252
  Датчик расстояния 256
  Датчик ориентации 258
  Акселерометр 263
  Датчик уровня магнитного поля 267
  Другие датчики, доступные на мобильных устройствах Android 269
Имитация работы сенсоров для эмулятора Android 269
Резюме 271

Глава 14. Управление дисплеем ... 272

Доступ к дисплею мобильного устройства ... 272
Менеджер окон ... 272
Параметры дисплея мобильного устройства ... 272
Управление яркостью экрана ... 276
Резюме ... 281

Глава 15. Доступ к аккумуляторной батарее ... 282

Менеджер источника питания ... 282
Отображение статистики использования батареи ... 290
Резюме ... 292

Глава 16. Управление энергопотреблением телефона ... 293

Менеджер энергопотребления ... 293
Управление энергопотреблением и блокировки ... 294
Резюме ... 300

ЧАСТЬ V. СИСТЕМНЫЕ СЕРВИСЫ ... 301

Глава 17. Получение информации о системе ... 303

Класс *ActivityManager* ... 303
Информация о конфигурации устройства ... 309
Информация о системе ... 313
Доступная память устройства ... 313
Выполняющиеся процессы ... 315
Выполняющиеся службы ... 316
Выполняющиеся задания ... 317
Последние выполненные задания ... 319
Процессы в состоянии ошибки ... 320
Терминал в системе Android ... 322
Резюме ... 328

Глава 18. Управление пользовательскими уведомлениями ... 329

Менеджер уведомлений ... 329
Создание уведомления ... 330
Резюме ... 335

Глава 19. Создание пользовательских оповещений 336

Менеджер оповещений 336
Использование оповещений 337
Резюме 343

Глава 20. Буфер обмена и API для работы с текстом 344

Менеджер буфера обмена 344
Синтез речи на основе текста 348
Резюме 353

Приложение. Установка примеров 355

Литература и веб-ресурсы 359

Предметный указатель 361

Введение

Книга, которую вы держите в руках, является логическим продолжением книги "Google Android: программирование для мобильных устройств", изданной БХВ-Петербург. С помощью этой книги вы научитесь создавать приложения, использующие возможности системы Android для доступа к сетевым коммуникациям, удаленным сервисам, а также локальным службам и "железу" мобильного телефона с системой 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.

На кого рассчитана эта книга

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. Читатель также должен уметь создавать приложения с графическим пользовательским интерфейсом под Android, поскольку такие приложения мы будем создавать на протяжении всей книги, и, желательно, иметь хотя бы базовый опыт по созданию служб и приложений для доступа к данным в системе Android.

Еще одно требование, предъявляемое к читателю, — наличие мобильного телефона с системой Android. Большинство примеров приложений, создаваемых в этой книге, 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 2.3.3 (API Level 10). Однако совсем необязательно иметь мобильное устройство с версией Android 2.3.3.

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.

Краткое описание глав

Книга состоит из 6 частей, которые содержат 20 глав, и одного приложения. Далее приводится краткое описание каждой из глав.

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

- *Глава 1. Подключение мобильного устройства для тестирования и отладки приложений*

  Описываются общие требования к компьютеру разработчика и среде разработки, настройка и подключение мобильного устройства к компьютеру для отладки и тестирования приложений. Рассматриваются установка необходимых драйверов, организация взаимодействия мобильного устройства с инструментами, входящими в состав Android SDK, и со средой разработки Eclipse.
- *Глава 2. Доступ к системным компонентам и сетевым сервисам*

  В этой главе даются сведения о системных компонентах и службах Android. Их широкие функциональные возможности можно применять в собственных приложениях. Описываются общие принципы доступа приложений к системным компонентам, локальным службам на платформе Android и сетевым сервисам, предоставляемым Google для пользователей мобильных телефонов.

Вторая часть книги — "Базовые функции телефона и сетей сотовой связи". Эта часть описывает работу с основными функциями мобильного телефона Android и сетями сотовой связи: управление телефонными вызовами, отправка и получение SMS-сообщений.

- *Глава 3. 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 4. 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 5. Отправка и получение SMS приложением*

  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.

Третья часть книги — "Сетевые коммуникации". Здесь рассматривается работа с мобильным Интернетом и сетями Wi-Fi: подключение к сетевым коммуникациям мобильного устройства, получение информации о конфигурации, управление сетевыми подключениями из кода приложения.

- *Глава 6. 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 7. 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.

Четвертая часть книги — "Местоположение и навигация". В этой части рассматривается работа с провайдерами местоположения и с сетевыми сервисами Google Maps и Geocoding: получение информации о местоположении, встраивание карт в собственные приложения, поиск объектов с помощью карт Google Maps и сопоставление адреса объекта и его географических координат.

- *Глава 8. 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 9. 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 10. Использование карт Google Maps в приложениях*

  В этой главе рассматривается использование внешних библиотек Google API в своих приложениях. Рассматриваются 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.

Пятая часть книги — "Работа с оборудованием мобильного устройства". 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 11. Карта памяти и файловая система*

  В этой главе даются сведения о получении доступа к карте памяти, информации о ее состоянии. Также рассматриваются организация файловой системы на карте

памяти и мобильном устройстве, создание каталогов, сохранение и чтение файлов с карты памяти.

- *Глава 12. Использование видеокамеры*
  Описываются доступ к видеокамере мобильного устройства, использование и управление камерой в приложениях: управление автофокусом, масштабирование изображений, сохранение видео и фотографий.
- *Глава 13. 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 14. 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 15. 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 16. 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.

Шестая часть книги — "Системные сервисы". В этой части 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 17. 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.
- *Глава 18. Управление пользовательскими уведомлениями*
  В этой главе описывается использование менеджера уведомлений для создания приложений, осуществляющих интерактивное взаимодействие с пользователем мобильного устройства.

- *Глава 19. Создание пользовательских 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 20. Буфер обмена и API для работы с текстом*
  Глава посвящена использованию менеджера системного буфера обмена в приложениях для копирования и вставки текста, а также применению Text To Speech — библиотеки для синтеза речи на основе текста.

Исходные коды примеров

По ссылке **ftp://85.249.45.166/9785977506663.zip** можно скачать архив диска с исходными кодами примеров, приведенных в книге[1]. Установка примеров описана в *приложении*. Все примеры используют функциональность Android SDK версии 2.3.3 или ниже и скомпилированы в среде Eclipse.

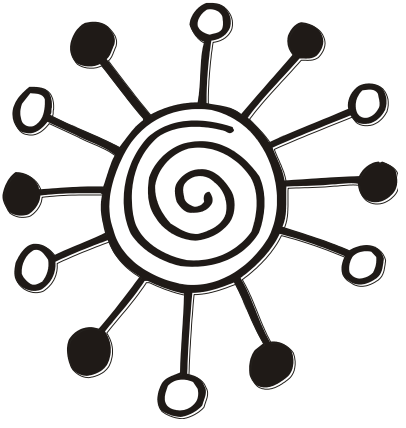
Книга содержит полные исходные коды всех программ, имеющихся на диске, однако некоторые листинги программ для экономии места и во избежание ненужного дублирования информации содержат только изменения программного кода относительно предыдущих листингов. Такое сокращение позволяет не только экономить место в книге, но и улучшить понимание программного кода, делая акцент только на новой функциональности. На диске также находятся файлы ресурсов, используемые в примерах приложений, приведенных в книге.

Все примеры приложений сделаны по возможности компактными и реализуют 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 на диске структурированы по темам и позволяют, при необходимости, легко отыскать реализацию нужной вам функциональности.

Благодарности

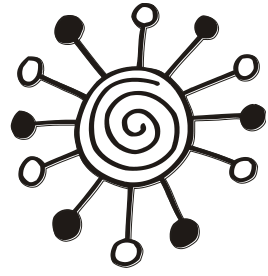
В первую очередь хочу поблагодарить своих родных и близких за оказанную моральную поддержку в процессе написания книги, а также всех сотрудников издательства "БХВ-Петербург", которые помогали мне в ее создании и без помощи которых эта книга не увидела бы свет и не попала бы в руки читателей.

[1] Ссылка доступна также со страницы книги на сайте **www.bhv.ru**.



ЧАСТЬ I

ОБЩИЕ СВЕДЕНИЯ



Глава 1

Подключение мобильного устройства для тестирования и отладки приложений

Прежде чем мы приступим к разработке приложений, представленных в этой книге, я бы хотел обратить ваше внимание на инструменты, требуемые для разработки приложений под Android. В этой главе описываются настройка и подключение мобильного телефона Android для тестирования и отладки приложений, которые мы будем разрабатывать в процессе работы с книгой.

Среда разработки и Android SDK

Поскольку вы уже наверняка знакомы с разработкой приложений под Android, на вашем рабочем компьютере должны быть установлены необходимые инструменты. Для работы нам понадобятся последняя версия среды разработки Eclipse (на момент написания книги — Eclipse Helios 3.6.2) и плагин ADT (Android Developer Tools) для нее. Версия плагина ADT — 10.0.1 или выше, если к моменту, когда вы будете читать эту книгу, выйдет следующая.

Также для работы с книгой у вас должны быть установлены библиотеки Android SDK версии 2.3.3 (API Level 10). Версия Android SDK 3.0 (API Level 11) не обязательно должна присутствовать в вашей инсталляции, поскольку мы будем рассматривать создание приложений только для смартофонов Android, а работа с планшетными PC — это тема для другой книги.

Если вы давно не обновляли свою инсталляцию Android SDK, проверьте установленные у вас библиотеки. Откройте SDK Manager в каталоге, где у вас установлен Android SDK, и в окне **Android SDK and AVD Manager** откройте узел **Installed Packages**. На правой панели отобразятся установленные пакеты, как показано на рис. 1.1.

Кроме Android SDK версии 2.3.3, нам понадобятся библиотеки Google API для работы с сетевыми сервисами Google. Эти библиотеки не входят в состав Android SDK и устанавливаются отдельно. У вас должны быть инсталлированы библиотеки до Google API Level 10 включительно (рис. 1.1).

Если у вас не установлены требуемые библиотеки, обязательно установите их, они понадобятся нам для работы с примерами приложений из этой книги. Следующий шаг, который необходимо сделать, — заняться настройкой мобильного телефона.

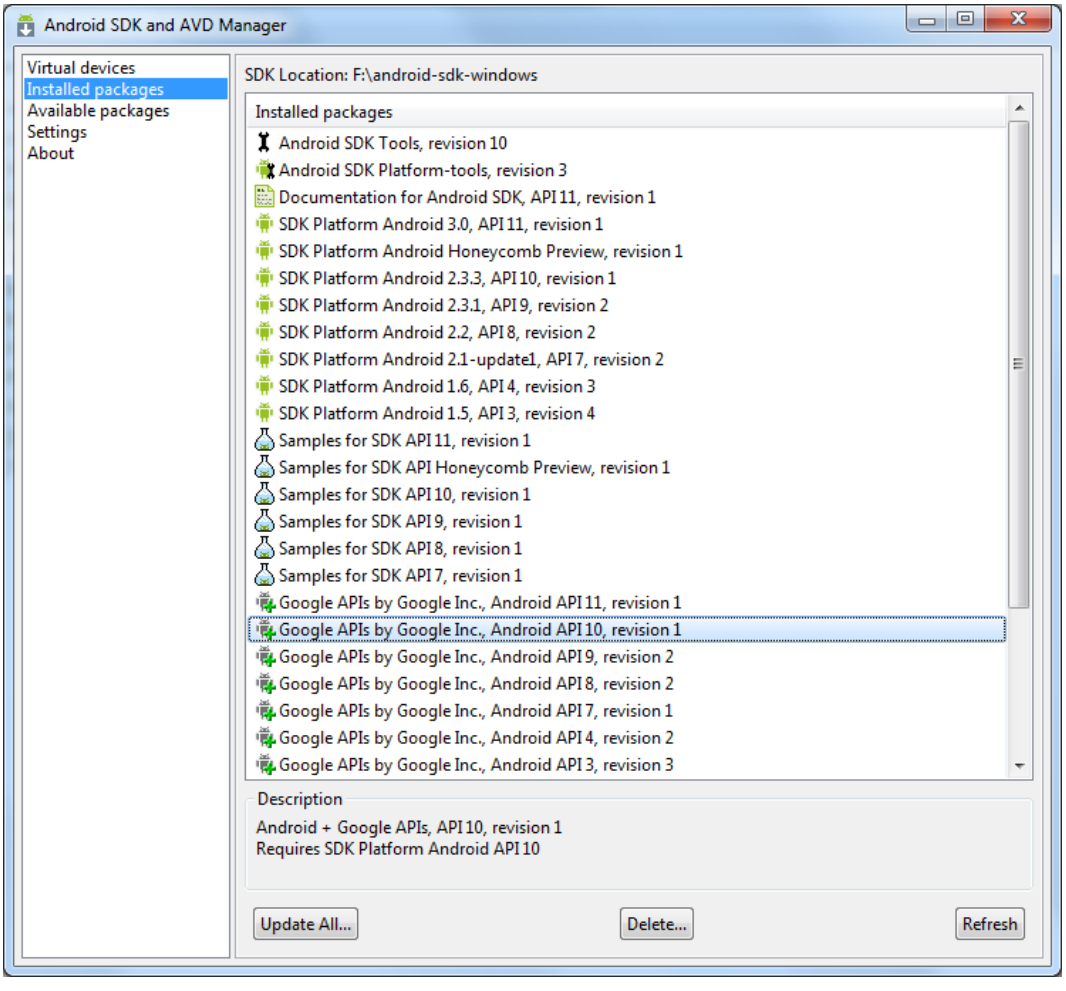


Рис. 1.1. Установленные пакеты Android SDK и Google API

Настройка мобильного телефона для отладки приложений

Эмулятор Android предоставляет широкие возможности для отладки и тестирования приложений. Те читатели, кто уже изучил мою предыдущую книгу [1], посвященную базовым принципам программирования на платформе Android, уже познакомились с функциональностью, предоставляемой эмулятором мобильного устройства Adnroid. Практически все приложения для платформы Android, создаваемые в книге [1], — службы, приложения с графическим интерфейсом, приложения для работы с базами данных, графикой и мультимедиа — позволяют использовать эмулятор Android для отладки и тестирования.

Однако при разработке коммерческого приложения для мобильного устройства Android необходимо проверить свое приложение на реальном телефоне, прежде чем передавать его пользователям. Кроме того, необходимо учитывать, что не все приложения можно отладить на эмуляторе Android. Безусловно, в эмуляторе Android имеется множество возможностей, однако некоторую функциональность требуется отлаживать и тестировать только на реальном устройстве. Например, запускать приложения, работающие по Wi-Fi, Bluetooth, использующие встроенные в мобильный телефон сенсоры для измерения температуры, ориентации телефона, уровня зарядки батареи, а также встроенную видеокамеру, необходимо только на реальном мобильном устройстве.

На реальном мобильном устройстве Android 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 выполнить 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 поставили.
- ❒ Проверить работоспособность соединения с помощью инструмента DDMS (Dalvik Debug Monitor Server).

Установка режима отладки через USB

На мобильном телефоне выберите **Settings** (обычно эта опция выводится на домашнем экране 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. 1.2.

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.

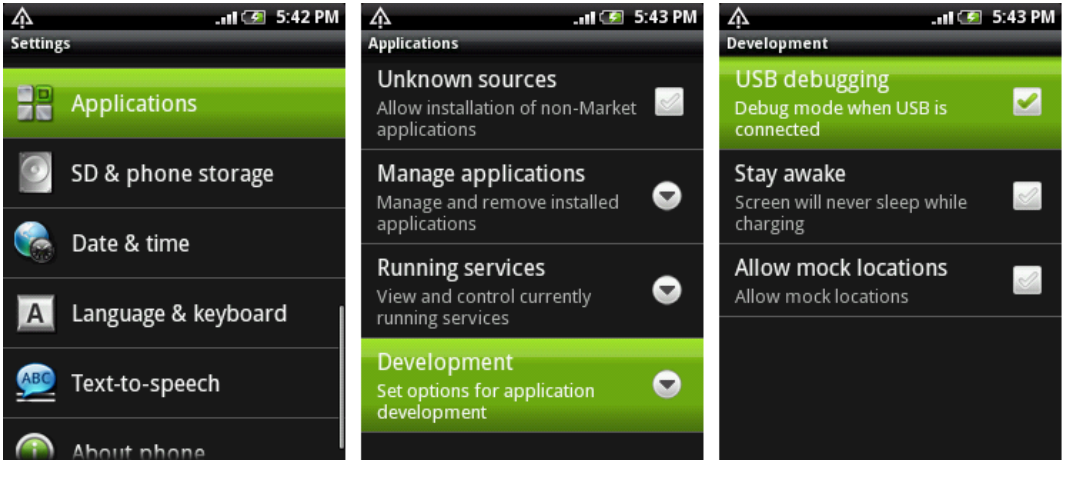


Рис. 1.2. Включение режима USB debugging на мобильном телефоне

Установка драйвера USB

Чтобы соединить мобильный телефон 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 вашего 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 мобильного телефона с DDMS

После подключения телефона и установки драйвера USB необходимо проверить взаимодействие мобильного устройства и инструментов для отладки Android-приложений. Для этого запустите инструмент Dalvik Debug Monitor Server из подкаталога tools в каталоге установки вашего Android SDK или напрямую из IDE Eclipse, выбрав представление **Devices** (пункт меню **Window | Show View | Other | Android | Devices**).

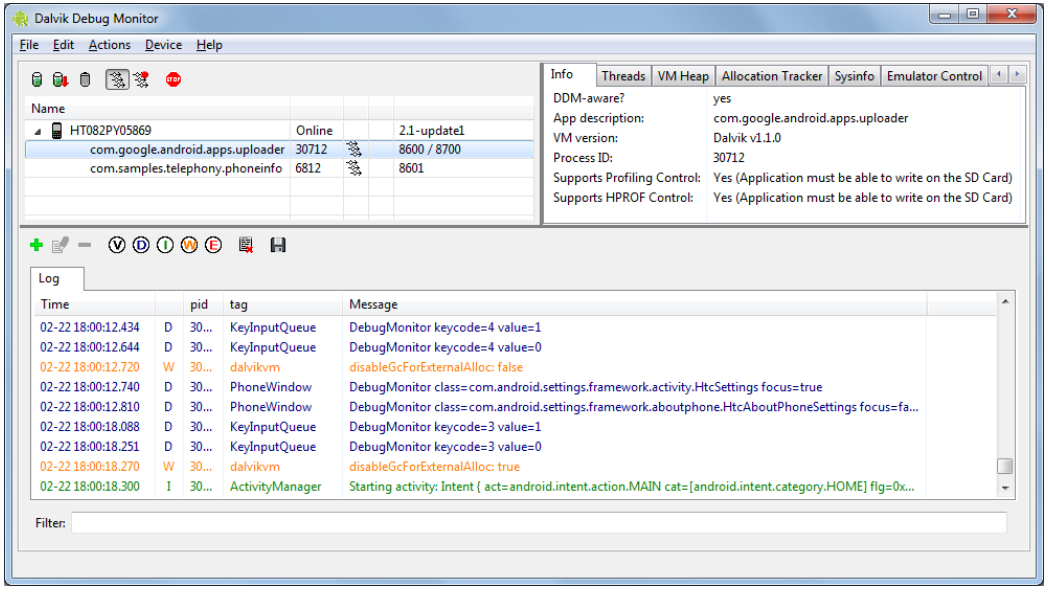


Рис. 1.3. Подключение мобильного устройства к DDMS

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. 1.3.

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

Запуск и инсталляция проекта на мобильный телефон из IDE Eclipse

Теперь мобильное устройство успешно подключено к компьютеру, в среде Eclipse можно выбирать место развертывания вашего приложения: эмулятор Android или мобильное устройство. При запуске проекта на выполнение появится диалоговое окно **Android Device Chooser**, которое позволяет выбрать место для развертывания приложения (рис. 1.4), при условии, что у вас запущен эмулятор Android. Если эмулятор не запущен, окно выбора устройства не появится и приложение развернется сразу на мобильном устройстве.

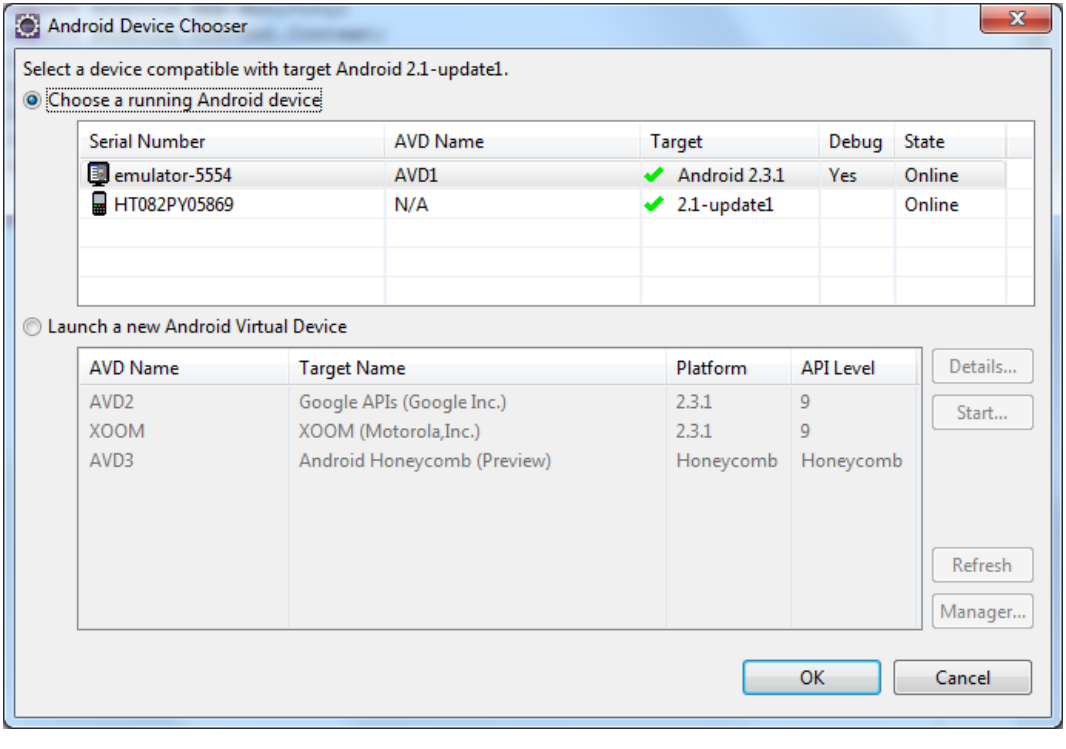


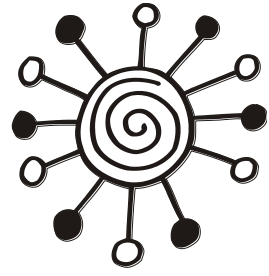
Рис. 1.4. Выбор целевого устройства для инсталляции и запуска проекта

Резюме

В этой главе мы рассмотрели подключение и настройку мобильного устройства на платформе Android для тестирования и отладки приложений, необходимые для работы с примерами, предоставленными в книге. Однако если у вас пока нет в наличии телефона Android, не расстраивайтесь. Многие примеры приложений в этой книге позволяют обойтись эмулятором Android.

Но необходимо учитывать, что для разработки приложений, использующих соединения Wi-Fi (*глава 7*), встроенную видеокамеру (*глава 12*), аппаратные датчики (*глава 13*), доступ к аккумуляторной батарее (*глава 15*) и управление энергопотреблением (*глава 16*), без мобильного телефона Android не обойтись.

В следующей главе будут рассмотрены базовые принципы вызова локальных и сетевых служб, системных компонентов и стандартных окон для настройки внутренних баз данных, а также установка требуемых разрешений для обеспечения приложениям доступа к системным компонентам Android и сетевым коммуникациям.



Глава 2

Доступ к системным компонентам и сетевым сервисам

Прежде чем мы перейдем непосредственно к созданию приложений с использованием функциональностей, предоставляемых системными компонентами Android, имеет смысл сделать небольшой обзор этих компонентов и обсудить некоторые базовые принципы работы с ними.

В этой главе описываются общие принципы организации системных компонентов Android и доступа к ним приложений.

Компоненты системы Android

Те из читателей, кто уже изучал самостоятельно архитектуру системы Android, должны быть знакомы с базовыми компонентами Android. Их всего четыре:

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

Из моей книги [1] вы уже узнали принципы работы с компонентами, научились строить приложения с графическим интерфейсом, создавать и вызывать собственные службы, работать с базами данных. 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, предоставляемые Google, например, Google Maps, Street View и др.

Системные службы

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 3.0 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. Эта 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, получению или потери фокуса виджетом;
- *Clipboard Service* — управляет буфером обмена. Буфер обмена в Android, в отличие от других систем, может использоваться только для текста и 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;

- *Dropbox 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 виброзвонком 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 управляем 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, например, 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.

Объекты *Intent*

Помимо служб, перечисленных в предыдущем разделе, в системе Android существует множество стандартных Activity, которые вы можете запускать из своего приложения, используя объекты `Intent`. Эти объекты в системе Android служат средством для позднего связывания во время выполнения между компонентами одного или нескольких приложений.

Все объекты `Intent` можно разделить на две категории:

- *Activity Action Intent* — используются для вызова Activity, находящегося в другом приложении. Для вызова абонента и ответа на входящий звонок можно вызвать соответствующий Activity. Если нам требуется включить сетевое соединение, можно сделать элемент управления (например, кнопку) для включения соединения в приложении или вызвать стандартный системный Activity, который предоставляет разнообразные опции для управления и конфигурирования сетевого соединения;
- *Broadcast Intent* — рассылаются всем компонентам, находящимся на мобильном устройстве для оповещения о каком-либо событии, произошедшем в системе. Например, при получении мобильным устройством входящего SMS-сообщения система Android посылает оповещение Broadcast Intent. Любое приложение может перехватить и обработать этот объект `Intent`, отреагировав на это событие соответствующим образом.

Intent-фильтры

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. Для этого существуют специальные фильтры объектов `Intent` (Intent Filter). Мы их также будем применять, в случае необходимости, при создании приложений в этой книге.

Системные контент-провайдеры

Доступ к данным и работа с данными в приложениях в системе Android осуществляется через следующие механизмы:

- база данных SQLite — реляционная база данных, которая используется на платформе Android;
- контент-провайдеры — компоненты системы, которые предоставляют приложениям интерфейс для работы с данными.

Контент-провайдеры сохраняют и возвращают данные для приложения, в котором они используются. По сути, они являются посредниками между приложением и базой данных.

Встроенные базы данных

Платформа Android предоставляет несколько встроенных баз данных, доступных для приложений через соответствующие контент-провайдеры.

Вы можете получить доступ к этим контент-провайдерам, используя функциональность, предоставляемую пакетом `android.provider`. Это стандартные контент-провайдеры, 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.

Каждому из перечисленных контент-провайдеров в системе Android соответствует одноименная база данных.

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 5*, при изучении способов программного перехвата входящих и исходящих SMS-сообщений.

Кроме перечисленных, на мобильном телефоне также имеются различные специализированные базы данных, предназначенные для обеспечения работы различных приложений. Например, веб-браузер, помимо базы данных Browser, использует дополнительную базу данных WebviewCache для кэширования, будильник использует собственную базу данных Alarms.

Установка требуемых разрешений в приложении

По умолчанию, 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.

При создании приложений в этой книге мы будем постоянно добавлять требуемые разрешения в файл манифеста приложения. Для тех читателей, кому еще не доводилось применять разрешения в своих приложениях, я кратко расскажу об их использовании.

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 **User Permissions**, как показано на рис. 2.1.

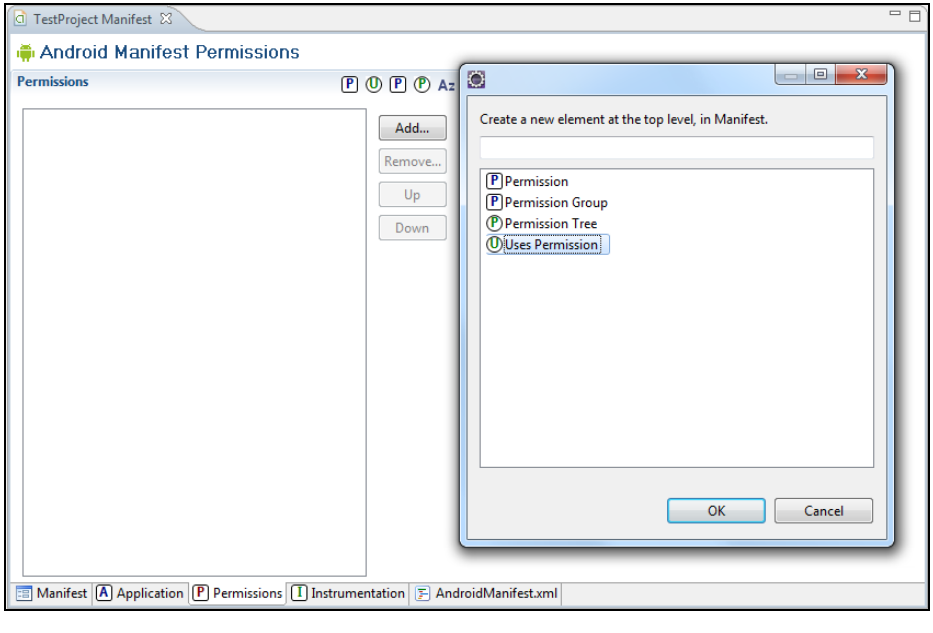


Рис. 2.1. Установка разрешений в файле AndroidManifest.xml

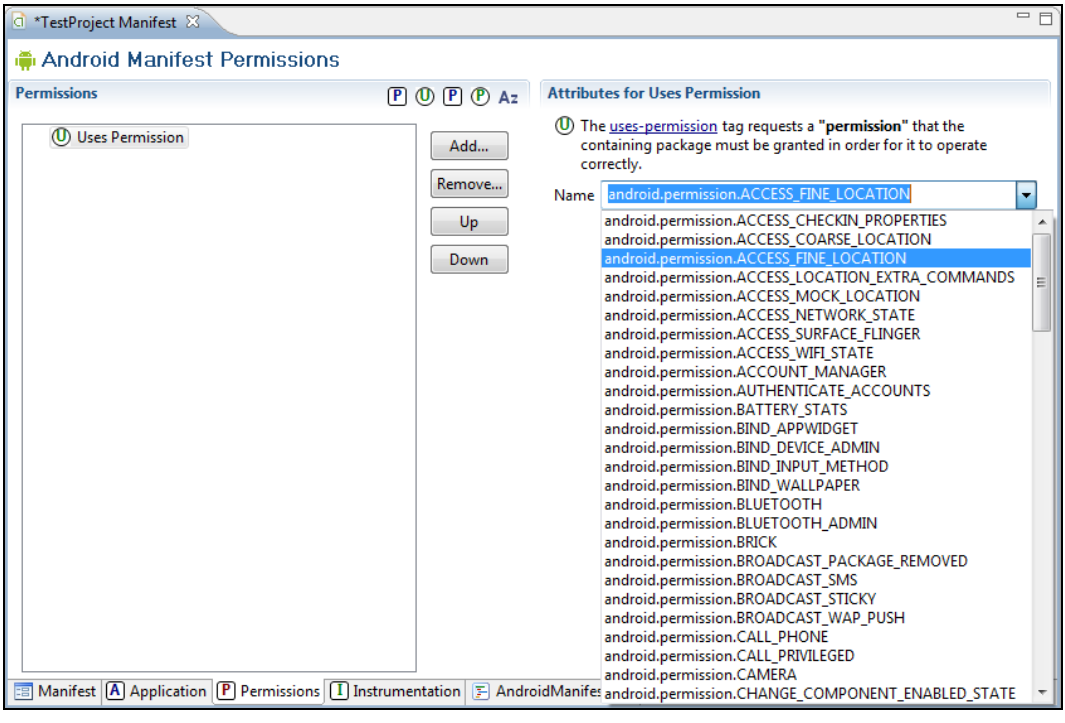


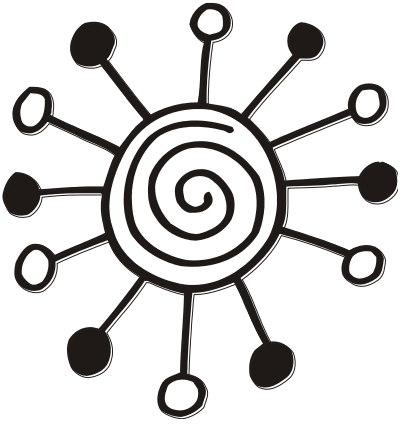
Рис. 2.2. Выбор необходимого разрешения для доступа к системным компонентам

После этого на вкладке **Permissions** в правой части появится панель **Attributes for Uses Permission**. На этой панели в выпадающем списке **Name** находится полный список всех доступных разрешений (рис. 2.2), которые вы можете использовать у себя в приложении. После выбора нужного разрешения редактор манифеста добавит его в файл AndroidManifest.xml.

При проектировании приложения программисты довольно часто забывают подключить разрешения. Впрочем, это сразу обнаруживается при запуске приложения — при отсутствии нужного разрешения генерируется исключение.

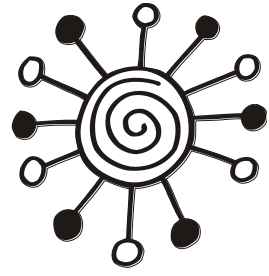
Резюме

В этой главе были рассмотрены компоненты системы Android и общая организация доступа к системным компонентам Android, внутренним базам данных и сетевым службам. Более детализированный доступ к системной функциональности из кода приложений и работа с компонентами, входящими в состав платформы Android, будут рассматриваться в следующих главах книги.



ЧАСТЬ II

Базовые функции телефона и сетей сотовой связи



Глава 3

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`. Это 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 фунциональности, 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. Стандарт GSM на сегодняшний день является наиболее распространенным стандартом связи;
- `PHONE_TYPE_CDMA` — 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` — 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()`, определяющий тип сотовой сети, в зоне действия которой в данный момент находится мобильный телефон, 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 должны работать в любой точке мира. Кстати, список констант в классе `TelephonyManager`, определяющий типы 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 2.3.1 (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* в приложении

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 на прилагаемом к книге компакт-диске находится в каталоге Ch03_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 3.1 (выделено полужирным шрифтом), иначе при запуске приложения у вас будет генерироваться исключение.

Листинг 3.1. 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 3.2.

Листинг 3.2. 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 3.3.

Листинг 3.3. 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 3.1 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. 3.2.

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. 3.1. Вывод информации о телефоне на реальном мобильном устройстве

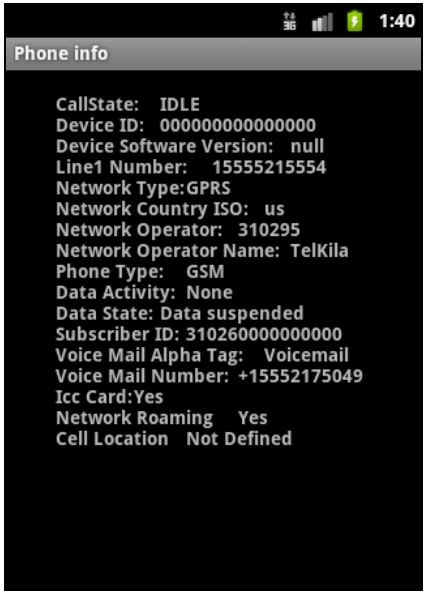


Рис. 3.2. Вывод информации о телефоне на эмуляторе Android

Доступ к SIM-карте

Используя класс `TelephonyManager`, можно получить полезную информацию о SIM-карте мобильного телефона. Для этих целей в классе `TelephonyManager` предусмотрено несколько методов:

- `getSimCountryIso()` — 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 идентификатором оператора мобильной связи;
- `getSimOperatorName()` — возвращает SPN (Service Provider Name, имя сервис-провайдера);
- `getSimSerialNumber()` — возвращает серийный номер SIM-карты при условии, что он доступен;
- `getSimState()` — возвращает целочисленный идентификатор, определяющий состояние SIM-карты.

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 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 на прилагаемом к книге компакт-диске находится в каталоге Ch03_SimInfo.

В файле манифеста приложения AndroidManifest.xml добавьте разрешение для чтения состояния телефона `READ_PHONE_STATE`, как в листинге 3.1. Для компоновки окна приложения используйте файл main.xml, представленный в листинге 3.2.

Класс главного окна приложения `SimInfoActivity` в целом аналогичен классу `PhoneInfoActivity` из предыдущего примера. Код класса `SimInfoActivity` представлен в листинге 3.4.

Листинг 3.4. 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. 3.3).

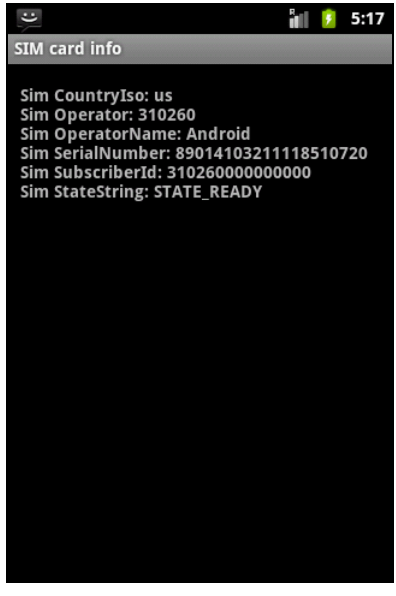


Рис. 3.3. Вывод информации о SIM-карте

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 (используется 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 стации 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 вкючении 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:** com.samples.telephony.phonechangestateinfo;
- **Package name:** com.samples.telephony.phonechangestateinfo;
- **Create Activity:** PhoneChangeStateInfoActivity.

> ***Примечание***
>
> Полный код приложения на прилагаемом к книге компакт-диске находится в каталоге Ch03_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 3.5.

Листинг 3.5. 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"
    android:layout_margin="5px">

    <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:layout_weight="1"/>
        <Button
            android:id="@+id/bStop"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent"
            android:text="Stop"
            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()`, `convertDataActivityToString()`, `convertDataConnStateToString()` и `convertPhoneTypeToString()`, которые аналогичны методам из листинге 3.3, представлен в листинге 3.6.

Листинг 3.6. 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);

        Button bStart = (Button)findViewById(R.id.bStart);
        Button bStop = (Button)findViewById(R.id.bStop);
        text = (TextView)findViewById(R.id.text);
        bStart.setOnClickListener(this);
        bStop.setOnClickListener(this);

        // 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 "Emergensy 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. 3.4.

Рис. 3.4. Начало работы приложения для прослушивания изменения состояний сети

При запуске приложение находится в состоянии покоя и не реагирует на изменения состояния сотовой сети. Для запуска прослушивания приложения надо нажать кнопку **Start**.

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. Эмулятор Android и Dalvik Debug Monitor Server предоставляют требуемую функциональность.

Запустить DDMS можно напрямую из каталога Android SDK (подкаталог tools\ddms.bat) или из среды Eclipse (конечно, у вас должен быть установлен плагин ADT). В среде Eclipse надо открыть меню **Window | Show View | Other** и выбрать опции **Devices** и **Emulator Control** (рис. 3.5).

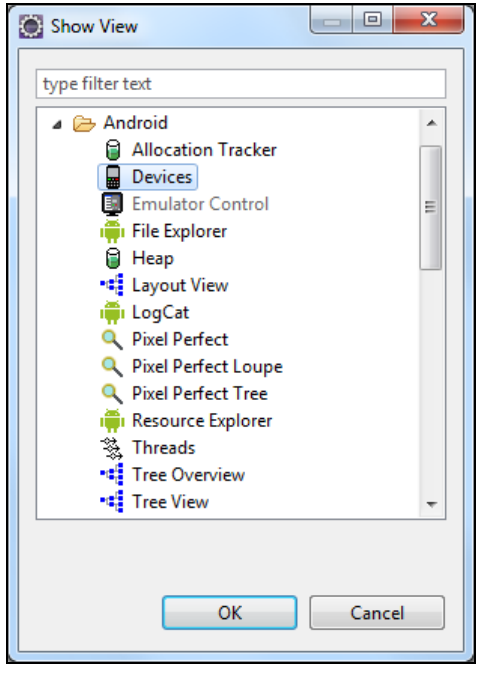


Рис. 3.5. Открытие представлений **Devices** и **Emulator Control** в IDE Eclipse

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. 3.6.

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. 3.7.

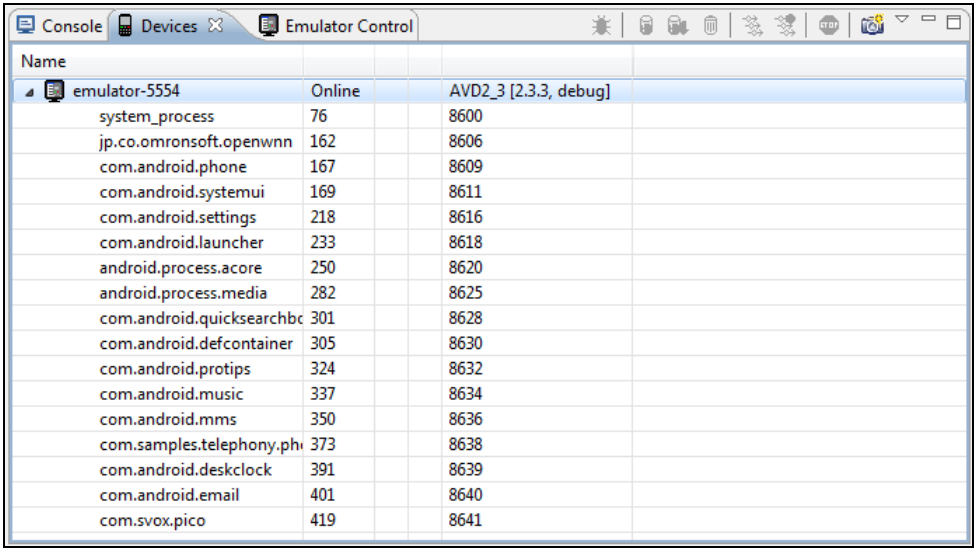


Рис. 3.6. Представление **Devices**

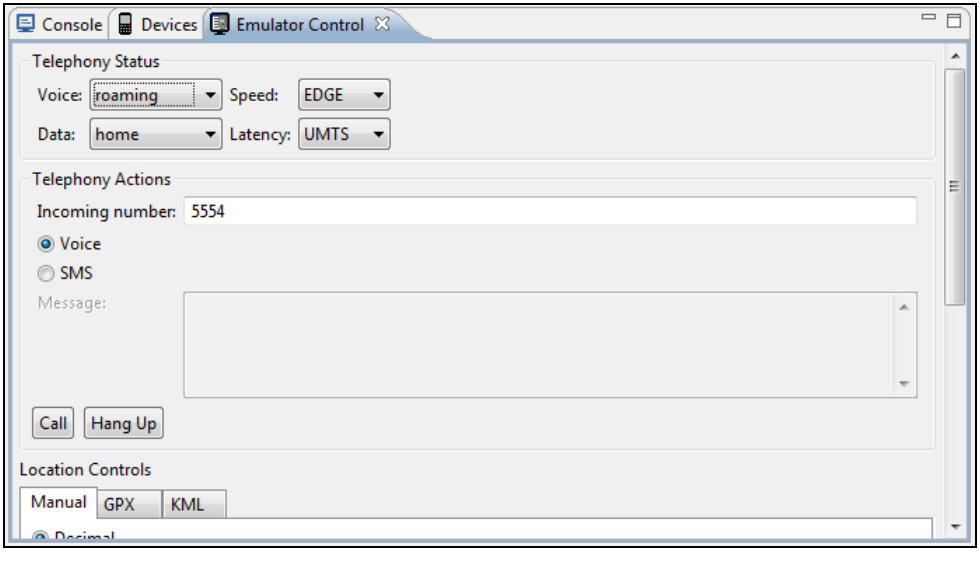


Рис. 3.7. Представление **Emulator Control**

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 *части IV* 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. 3.8.

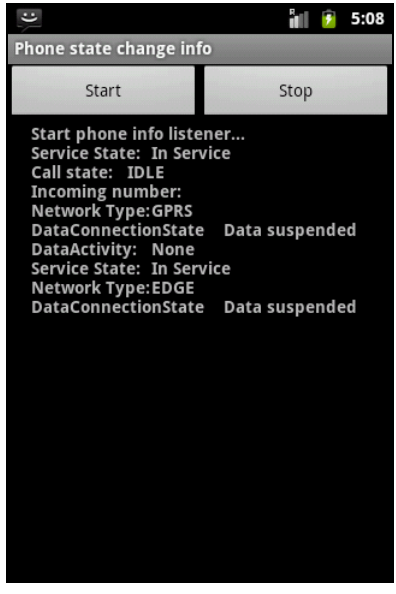


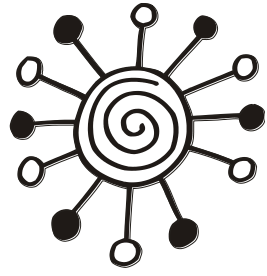
Рис. 3.8. Перехват приложением изменения статуса сети

Затем в представлении **Emulator Control** в панели **Telephony Status** попробуйте поменять значения в выпадающих списках **Voice**, **Data**, **Speed**, **Latency** (см. рис. 3.7). Приложение должно реагировать на изменение состояния сотовой сети, обновляя информацию на экране.

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 состояния сети сотовой связи и состояния телефона.

В следующей главе мы рассмотрим возможности отправки и получения SMS-сообщений напрямую из нашего приложения.



Глава 4

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 в входящими и исходящими телефонными вызовами совсем необязательно для отладки и тестирования программы использовать реальное мобильное устройство. Эмулятор Android предоставляет нам необходимую функциональность.

Для имитации входящих и исходящих звонков в эмуляторе мобильного устройства существуют два способа:

- имитация телефонного вызова с использованием инструмента DDMS;
- имитация телефонного вызова между двумя экземплярами эмуляторов Android.

Эти воможности эмулятора Android и DDMS пригодятся нам 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. 4.1).

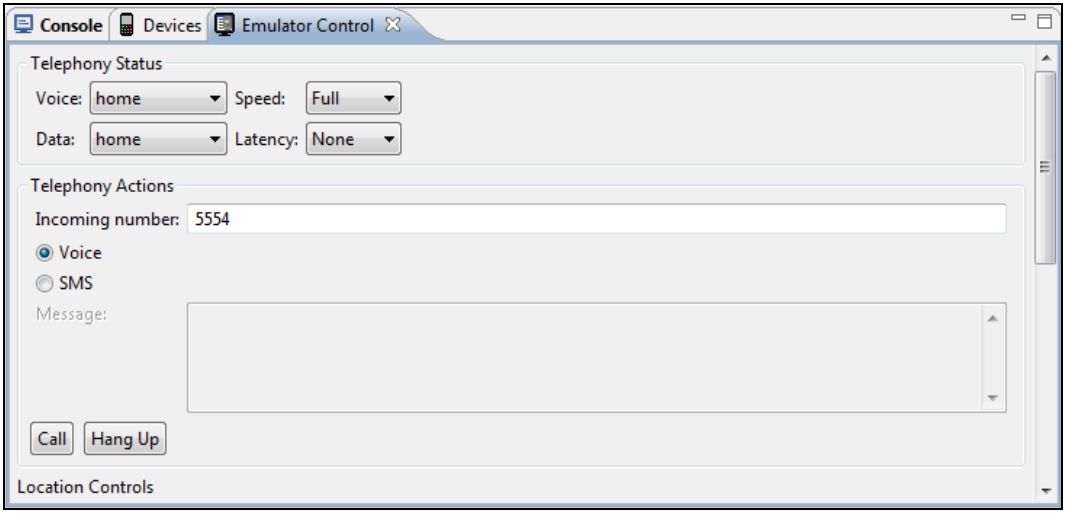


Рис. 4.1. Имитация телефонного звонка из Eclipse

В результате наш запущенный экземпляр эмулятора должен получить входящий вызов, как показано на рис. 4.2.

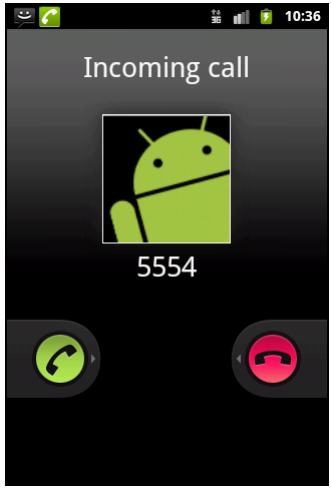


Рис. 4.2. Входящий звонок в эмуляторе Android

Имитация телефонного вызова между двумя эмуляторами Android

Для имитации телефонных вызовов между эмуляторами Android запустите еще один экземпляр эмулятора. Запущенные экземпляры эмуляторов будут видны в среде Eclipse в представлении **Devices**, как показано на рис. 4.3.

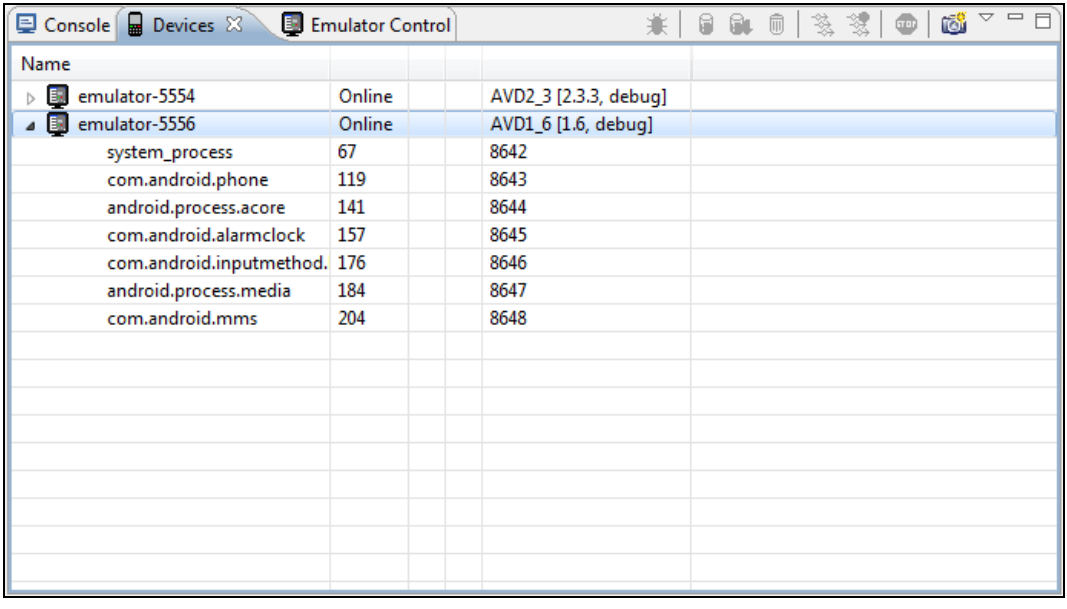


Рис. 4.3. Запущенные экземпляры эмуляторов Android в окне **Devices**

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.

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. 4.4.

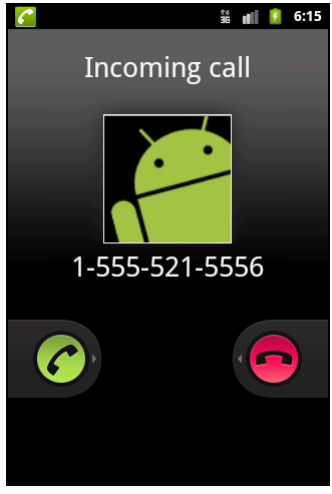


Рис. 4.4. Входящий звонок в эмуляторе 5554, полученный из эмулятора 5556

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. 4.5.

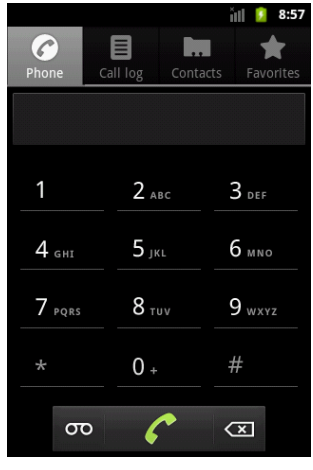


Рис. 4.5. Вызов стандартной панели набора номера

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.

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 на прилагаемом к книге компакт-диске в каталоге Ch04_PhoneCall.

В файл манифеста нужно обязательно добавить разрешение android.permission.CALL_PHONE для обеспечения возможности телефонного вызова абонента из приложения. Файл манифеста приложения AndroidManifest.xml представлен в листинге 4.1.

Листинг 4.1. 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 `EditView` для ввода телефонного номера и кнопку вызова с надписью **Call** и идентификатором `bCall`. Файл компоновки окна приложения представлен в листинге 4.2.

Листинг 4.2. 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"/>

</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 4.3.

Листинг 4.3. 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.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class PhoneCallActivity
        extends Activity implements OnClickListener {

    private EditText textNumber;
    private Button bCall;

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

        textNumber=(EditText)findViewById(R.id.textNumber);
        bCall = (Button)findViewById(R.id.bCall);

        bCall.setOnClickListener(this);
    }

    @Override
    public void onClick(View arg0) {
        try {
            // Создаем объект URI
            Uri uri = Uri.parse("tel:" + textNumber.getText().toString());

            // создаем объект Intent и вызываем Activity — наборную панель
            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. 4.6.

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. 4.7.

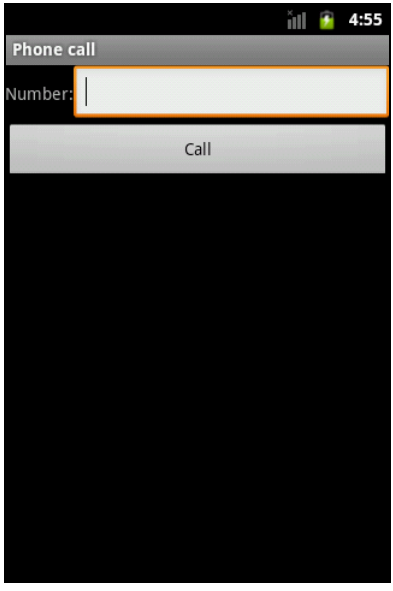


Рис. 4.6. Приложение для создания исходящего вызова

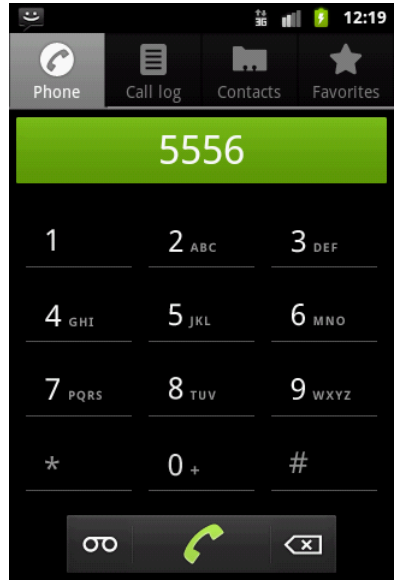


Рис. 4.7. Загрузка вызова абонента в окне наборной панели

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. 4.4.

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 4.4.

Листинг 4.4. 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. 4.8).

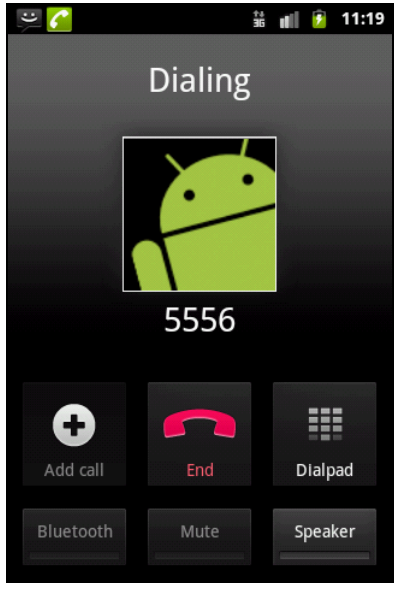


Рис. 4.8. Прямой телефонный вызов абонента 5556

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 на прилагаемом к книге компакт-диске в каталоге Ch04_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 4.5.

Листинг 4.5. 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 4.6.

Листинг 4.6. 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. 4.9.

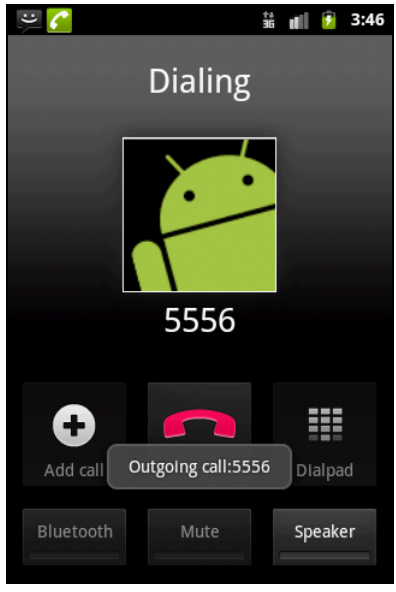
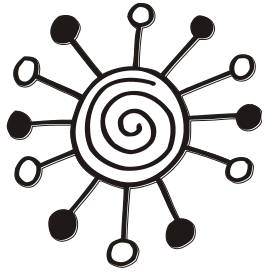


Рис. 4.9. Сообщение о перехвате входящего вызова приложением

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 возвожности 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 рассматривать возможности, предоставляемые платформой Android, по работе с SMS-сообщениями и рассмотрим способы отправки и получения SMS-сообщений напрямую из вашего приложения.



Глава 5

Отправка и получение SMS приложением

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. 5.1.

В эмуляторе мобильного устройства должно появиться уведомление о приходе SMS. Если открыть это уведомление, можно прочитать наше SMS, отправленное из представления **Emulator Control** (рис. 5.2).

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. 5.3).

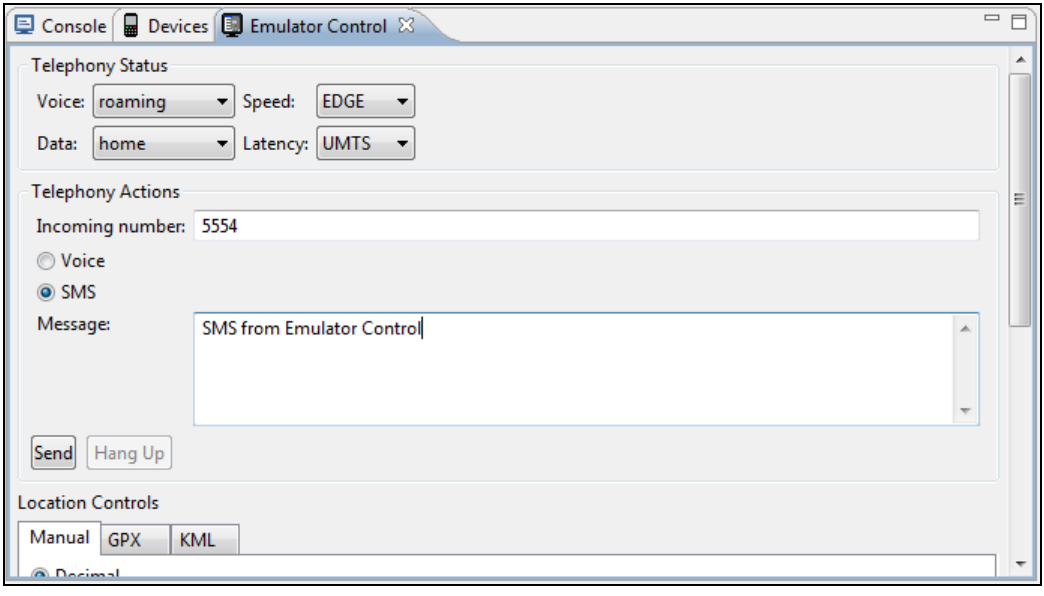


Рис. 5.1. Имитация отправки SMS из представления **Emulator Control**

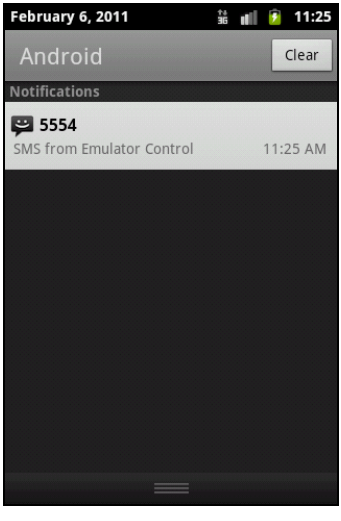


Рис. 5.2. SMS-сообщение, полученное эмулятором Android

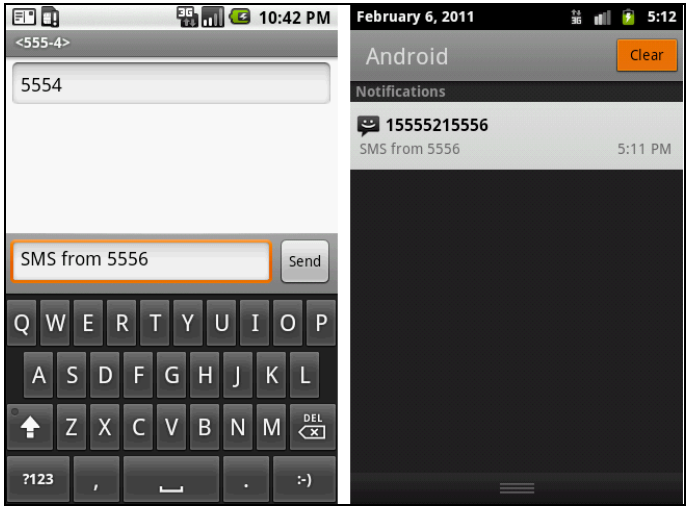


Рис. 5.3. Отправка SMS-сообщения между двумя эмуляторами

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 получать 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

Длина SMS на 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

Теперь рассмотрим 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 на прилагаемом к книге компакт-диске в каталоге Ch05_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 5.1.

Листинг 5.1. 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 5.2.

Листинг 5.2. 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 5.3.

Листинг 5.3. 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 icicle) {

        super.onCreate(icicle);
        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. 5.4.

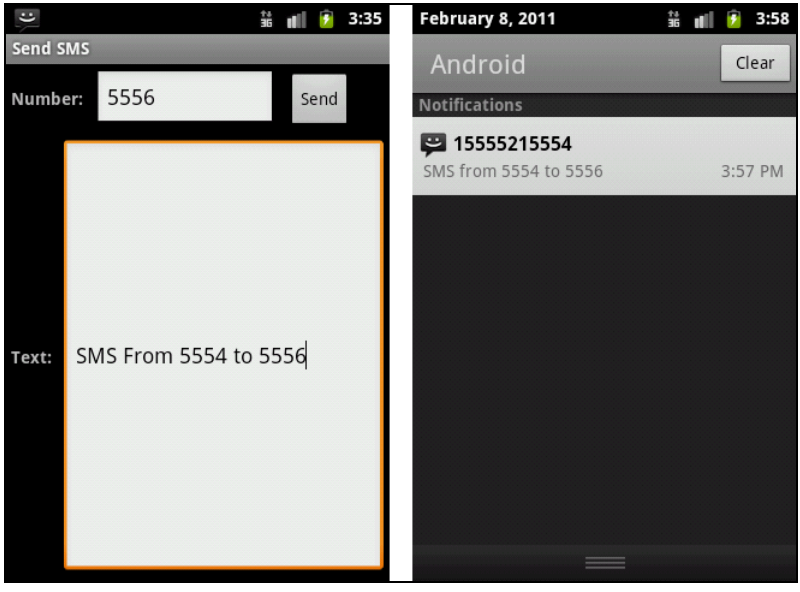


Рис. 5.4. Отправка SMS из приложения

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-собщения 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 получено, но не прочитано;
  - `STATUS_ON_ICC_UNREAD` — сообщение получено, но не прочитано;
  - `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

Для перехвата входящего SMS-сообщения используется объект `BroadcastReceiver`. 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 на прилагаемом к книге компакт-диске в каталоге Ch05_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 намерений с атрибутом `android:name="android.provider.Telephony.SMS_RECEIVED"`.

Файл манифеста приложения представлен в листинге 5.4.

Листинг 5.4. 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 5.5.

Листинг 5.5. 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[] pdus = (Object[]) bundle.get("pdus");
                for (Object pdu : pdus) {
                    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 содержимом этого сообщения. Внешний вид приложения с уведомлением представлен на рис. 5.5.

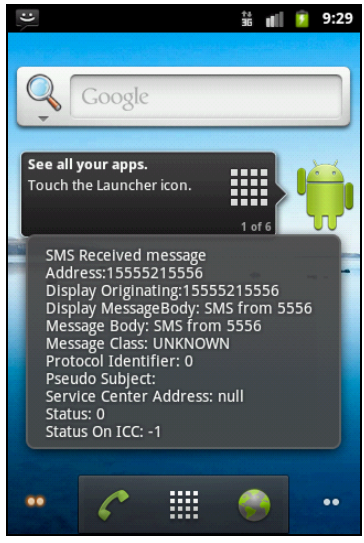


Рис. 5.5. Перехват входящего SMS

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 на прилагаемом к книге компакт-диске в каталоге Ch05_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 5.6.

Листинг 5.6. 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 5.7.

Листинг 5.7. 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.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 onListItemClick(ListView 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 5.8.

Листинг 5.8. 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 5.9.

Листинг 5.9. 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. 5.6.

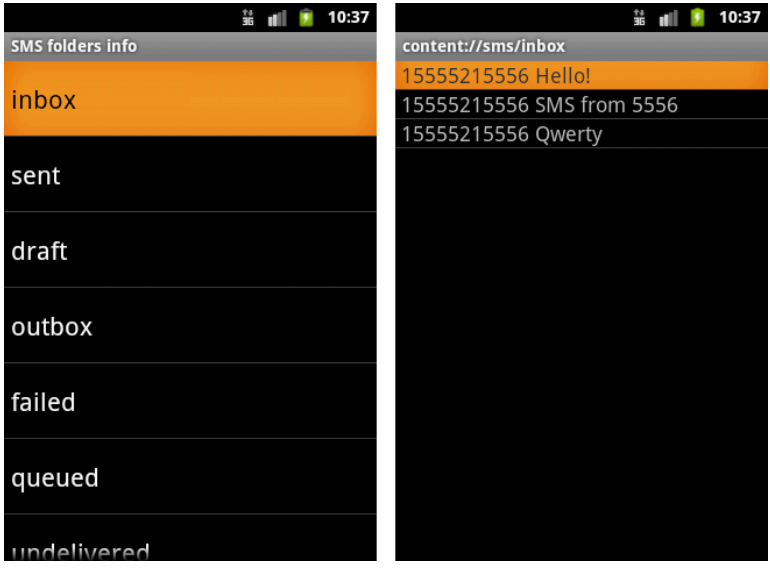


Рис. 5.6. Список SMS из папки **inbox**

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 блокированного сообщения;
- `error_code` — код ошибки, 0 в случае, если ошибки нет.

Названия этих полей можно получить в приложении. 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 содержимое выбранного SMS-сообщения.

Примечание

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch05_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 5.10.

Листинг 5.10. 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. 5.7.

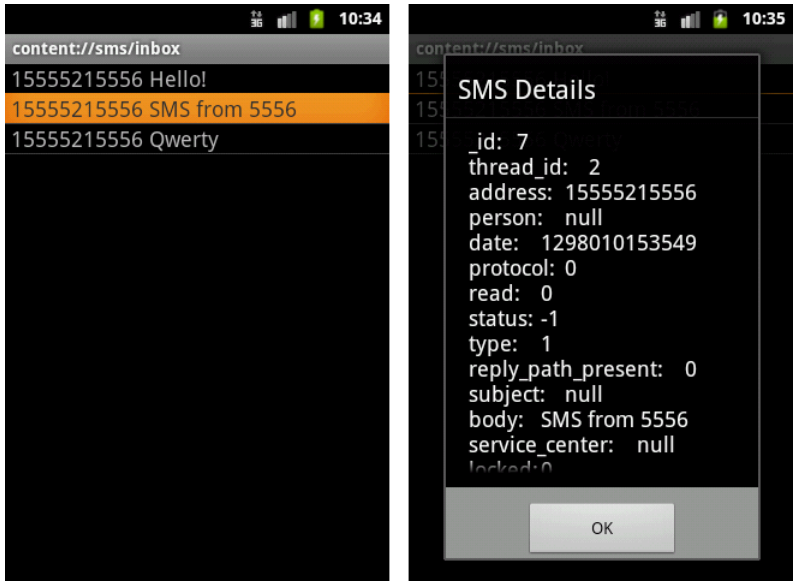


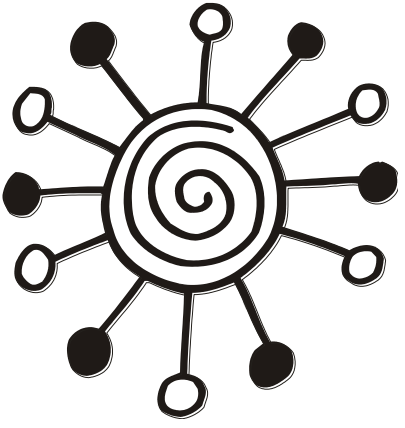
Рис. 5.7. Отображение детальной информации выбранного SMS

Резюме

Система Android предоставляет отличную функциональность для отправки и приема SMS-сообщений, что позволяет создать приложения, которые могут обмениваются информацией с другими мобильными устройствами и их пользователями.

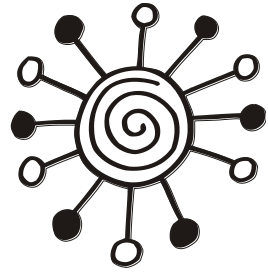
Здесь мы рассмотрели способы отправки и получения SMS-сообщений с использованием классов `SmsManager` и `SmsMessage` из пакета `android.telephony`. Также была рассмотрена структура каталогов SMS-сообщений и программный доступ к базе данных SMS, хранящейся на мобильном устройстве.

На этом мы заканчиваем рассмотрение базовых функций мобильного телефона и переходим к следующей части, в которой будем изучать сетевые коммуникации. Мы рассмотрим работу телефона с мобильным Интернетом, а также доступ и управление сетевым соединением через Wi-Fi.



ЧАСТЬ III

Сетевые коммуникации



Глава 6

Мобильный Интернет

Возможности мобильного устройства 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 используется 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 просмотреть 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 на прилагаемом к книге компакт-диске в каталоге Ch06_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 6.1.

Листинг 6.1. 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 6.2.

Листинг 6.2. 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. 6.1.

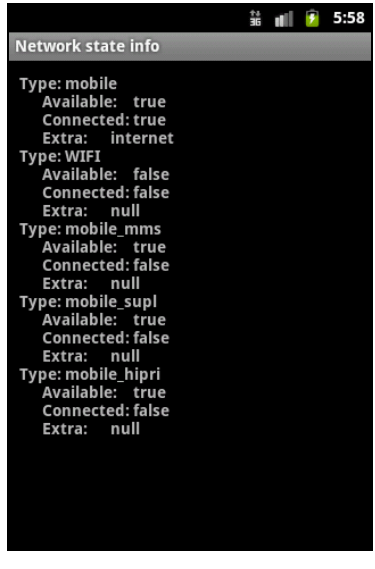


Рис. 6.1. Отображение информации о состоянии сети Интернет

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.

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 через 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 на прилагаемом к книге компакт-диске в каталоге Ch06_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 (также как и в предыдущем приложении, см. листинг 6.1).

В коде класса `TrafficStatsActivity` мы воспользуемся методами класса `TrafficStats`. Код класса `TrafficStatsActivity` главного окна приложения представлен в листинге 6.3.

Листинг 6.3. 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. 6.2.

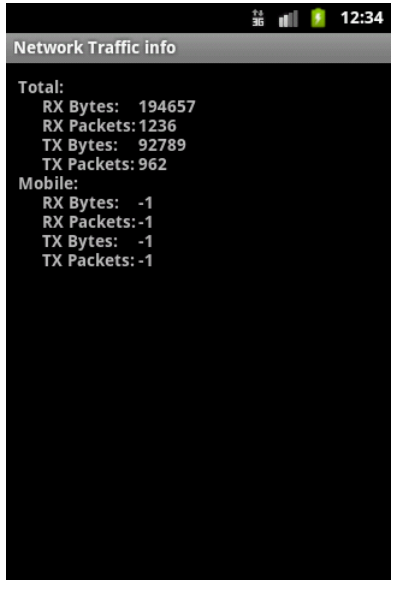


Рис. 6.2. Мониторинг сетевого трафика в приложении

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.

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, предъявляемым 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. 6.3 представлен пример отображения веб-страницы в браузере мобильного устройства.

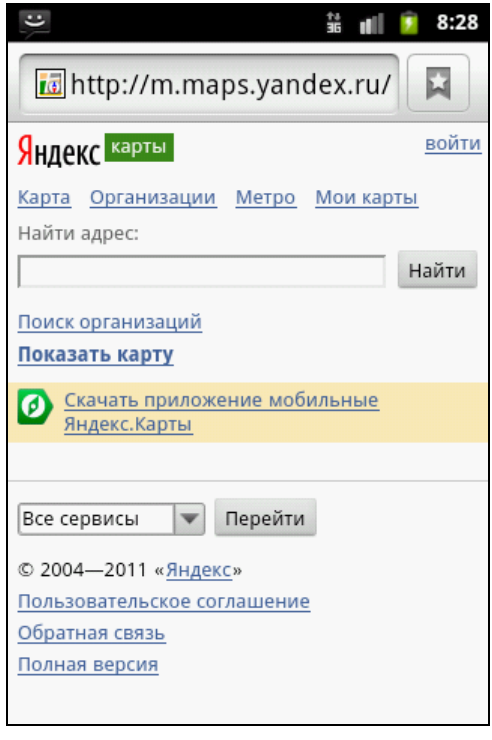


Рис. 6.3. Встроенный браузер

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.

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 на прилагаемом к книге компакт-диске в каталоге Ch06_WebKit.

В файле манифеста приложения AndroidManifest.xml для доступа к Интернету потребуется разрешение `android.permission.INTERNET`. Код файла манифеста приложения представлен в листинге 6.4.

Листинг 6.4. 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 6.5.

Листинг 6.5. 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 6.6.

Листинг 6.6. 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. 6.4.

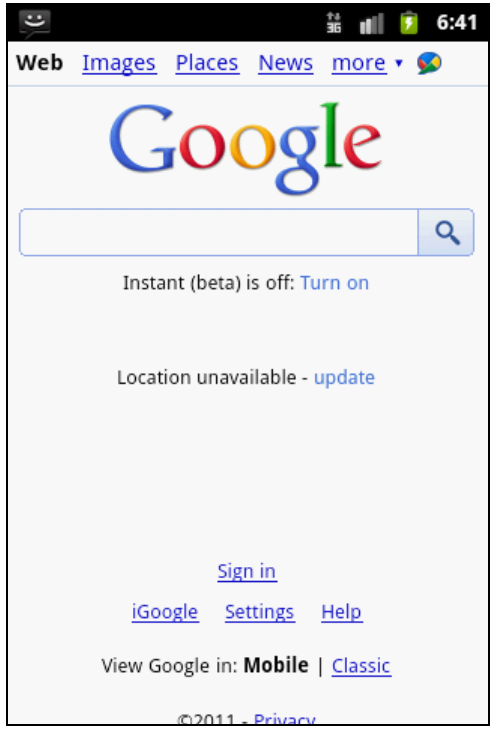


Рис. 6.4. Виджет `WebView` в приложении

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 6.7.

> ***Примечание***
>
> Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch06_WebKit_v2.

Листинг 6.7. 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. 6.5.

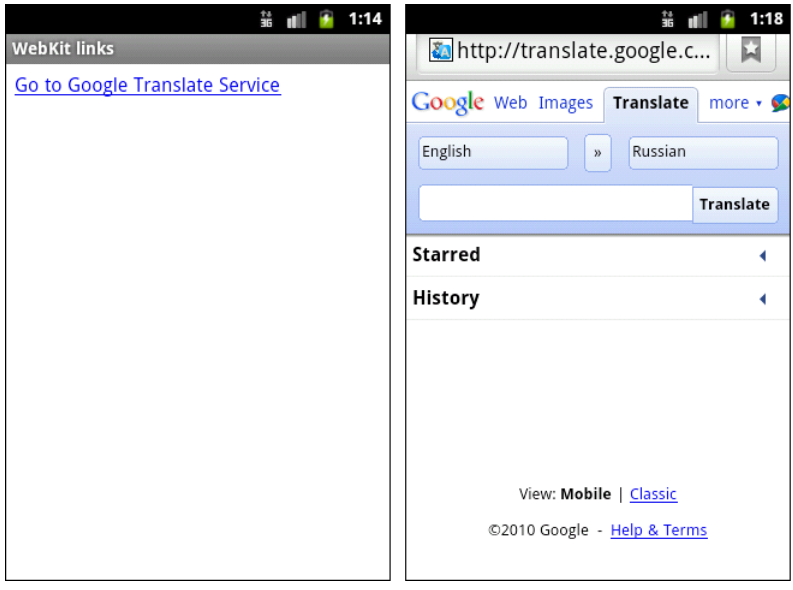


Рис. 6.5. Использование виджета `WebView` для загрузки данных

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()`;
- методы для установки масштаба: `setDefaultZoom()`, `setDisplayZoomControls()`, `setBuiltInZoomControls()`, `setSupportZoom()`;
- для установки параметров кэша: `setCacheMode()`, `setAppCachEnabled()` и т. д.

Пользовательские настройки браузера можно сохранять в системе, используя функциональность класса `Preference`, также 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 на прилагаемом к книге компакт-диске в каталоге Ch06_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 станицу:

- 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 6.8.

Листинг 6.8. 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 6.9.

Листинг 6.9. 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 6.10.

Листинг 6.10. 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">&amp;lt;&amp;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 6.11.

Листинг 6.11. 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 allowJScript =
            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(allowJScript);
        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 6.12.

Листинг 6.12. 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 6.13.

Листинг 6.13. 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);
    }
}
```

Скомпилируйте проект и запустите его на эмуляторе Android. Внешний вид нашего приложения с открытым меню представлен на рис. 6.6.

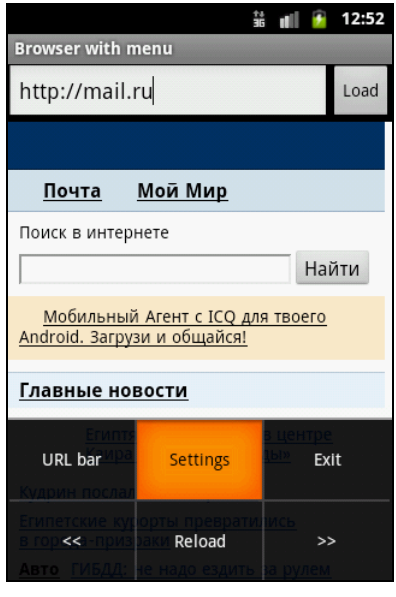


Рис. 6.6. Браузер на основе виджета `WebView` с меню

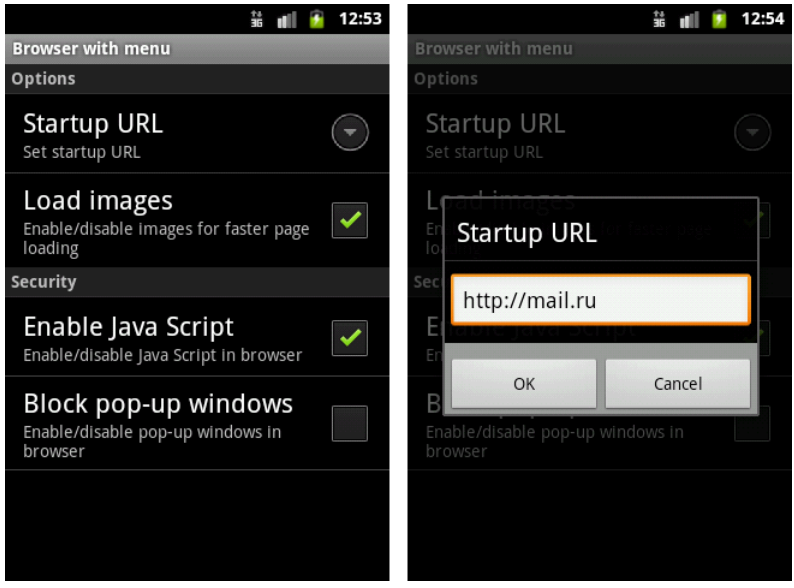


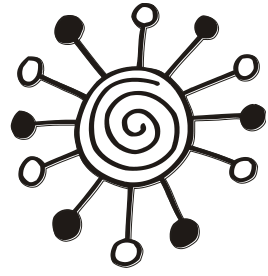
Рис. 6.7. Окно пользовательских настроек браузера

При выборе опции **Settings** откроется окно пользовательских настроек, представленное на рис. 6.7.

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 7

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 булевое 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 соединениия 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`

обеспечивает 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 соединение 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 с 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 на прилагаемом к книге компакт-диске в каталоге Ch07_ManageWiFiConnection.

В файле манифеста приложения AndroidManifest.xml нам потребуется объявить разрешение `android.permission.CHANGE_WIFI_STATE`. Код файла манифеста представлен в листинге 7.1.

Листинг 7.1. 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 7.2.

Листинг 7.2. 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 обработчике события `onClick()` кнопок будет производиться включение и выключение сетевого соединения.

Код класса `WiFiChangeActivity` представлен в листинге 7.3.

Листинг 7.3. 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. 7.1.

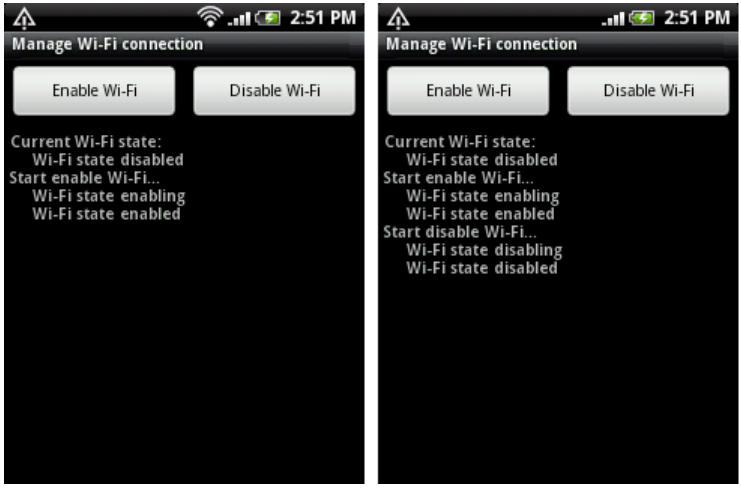


Рис. 7.1. Приложение, управляющее Wi-Fi-соединением на мобильном устройстве

Хотя эмулятор Android не обеспечивает симулирование Wi-Fi-соединения, но 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. 7.2).

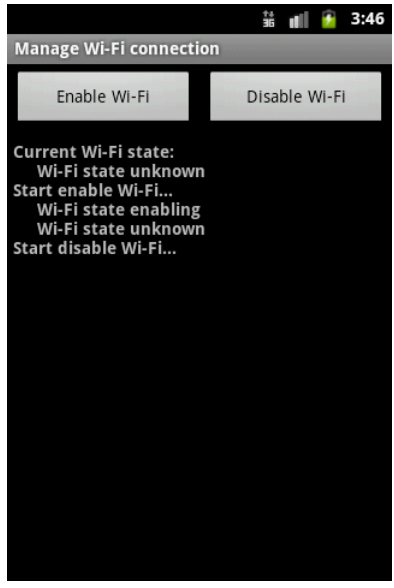


Рис. 7.2. Запуск приложения для управления Wi-Fi соединением в эмуляторе Android

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.

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:** WiFiChangeActivity.

> ***Примечание***
>
> Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch07_WiFiSettings.

В файл манифеста приложения AndroidManifest.xml добавьте разрешение `android.permission.CHANGE_WIFI_STATE`, как в листинге 7.1 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 7.4.

Листинг 7.4. 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.ArrayAdapter;
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 onListItemClick(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();
        }
    }
}
```

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. 7.3.

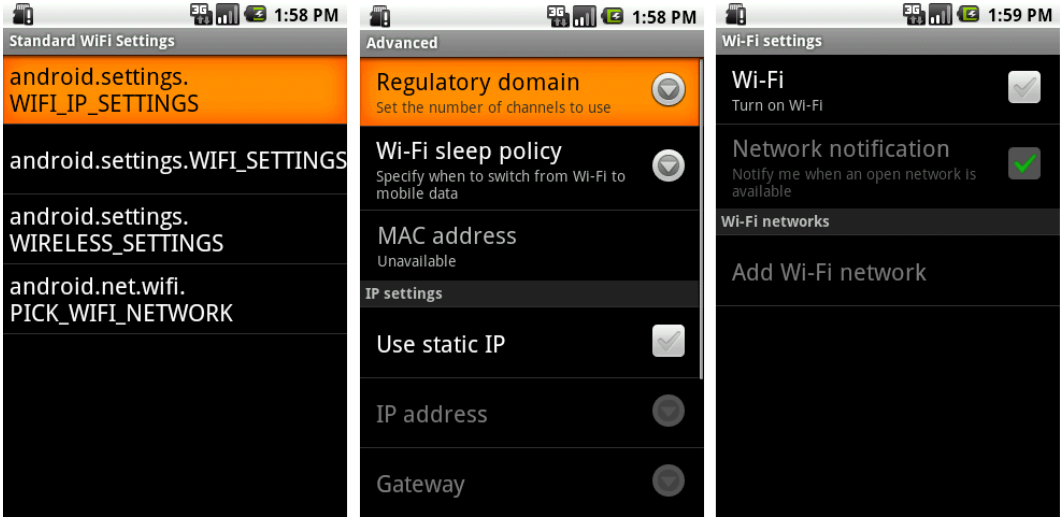


Рис. 7.3. Управление настройками Wi-Fi-соединения

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 3, раздел "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 необходимо 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 на прилагаемом к книге компакт-диске в каталоге Ch07_WiFiInfo.

В файле манифеста приложения AndroidManifest.xml объявите разрешения `ACCESS_NETWORK_STATE`, `CHANGE_WIFI_STATE` и `ACCESS_WIFI_STATE`, как показано в листинге 7.5.

Листинг 7.5. 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" andro-
id: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 7.6.

Листинг 7.6. 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 7.7.

Листинг 7.7. 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. 7.4.

Рис. 7.4. 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 использовании 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 на прилагаемом к книге компакт-диске в каталоге Ch07_WiFiConfiguration.

В файле манифеста приложения AndroidManifest.xml поставьте необходимые разрешения, как в листинге 7.5. Файл компоновки главного окна приложения main.xml будет аналогичен файлу компоновки предыдущего примера, приведенного в листинге 7.6.

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 7.8).

Листинг 7.8. 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("\nAuthentification 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. 7.5.

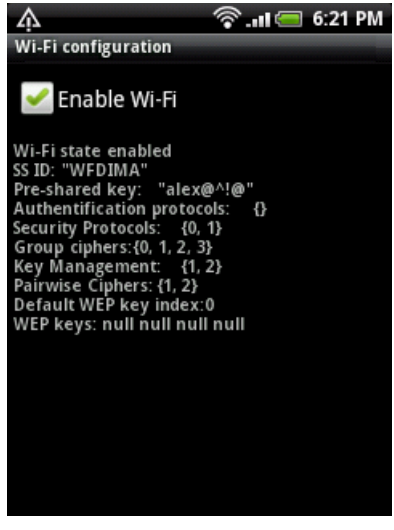


Рис. 7.5. Вывод конфигурации сети Wi-Fi

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 бóльшим уровнем сигнала.

Для сканирования точек доступа используется метод `startScan()`. Этот метод возвращает список объектов типа `ScanResult`. Этот тип определяет набор параметров, характеризующих каждую отсканированнную 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 она может выглядеть следующим образом: `"[WPA-PSK-CCMP]"`;
- `frequency` — рабочая частота канала в MHz;
- `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 на прилагаемом к книге компакт-диске в каталоге Ch07_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 7.9.

Листинг 7.9. 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 будет 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 7.10.

Листинг 7.10. 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. 7.6.

Рис. 7.6. 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 пользователем 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 7.7, 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 на прилагаемом к книге компакт-диске в каталоге Ch07_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 7.8 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 7.11.

Листинг 7.11. 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. 7.7.

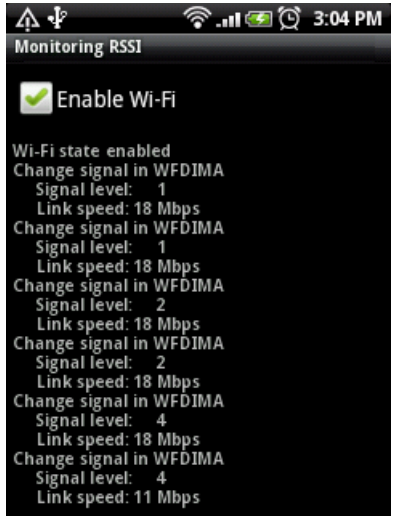
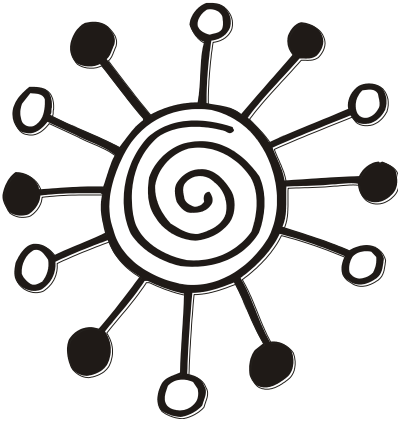


Рис. 7.7. Отслеживание изменения мощности сигнала Wi-Fi-соединения

Резюме

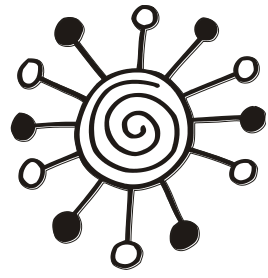
В этой главе мы рассмотрели подключение и конфигурирование соединения Wi-Fi в коде приложения. Мы также научились создавать приложения, которые могут осуществлять сканирование точек доступа сетевого соединения Wi-Fi и подключение к ним мобильного устройства, читать характеристики сети, отслеживать изменение уровня сигнала и управлять конфигурацией сети Wi-Fi.

Далее мы переходим к новой большой теме: определению местоположения и использование сетевых сервисов Google Maps и Geocoding.



ЧАСТЬ IV

Местоположение и навигация



Глава 8

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. 8.1.

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.

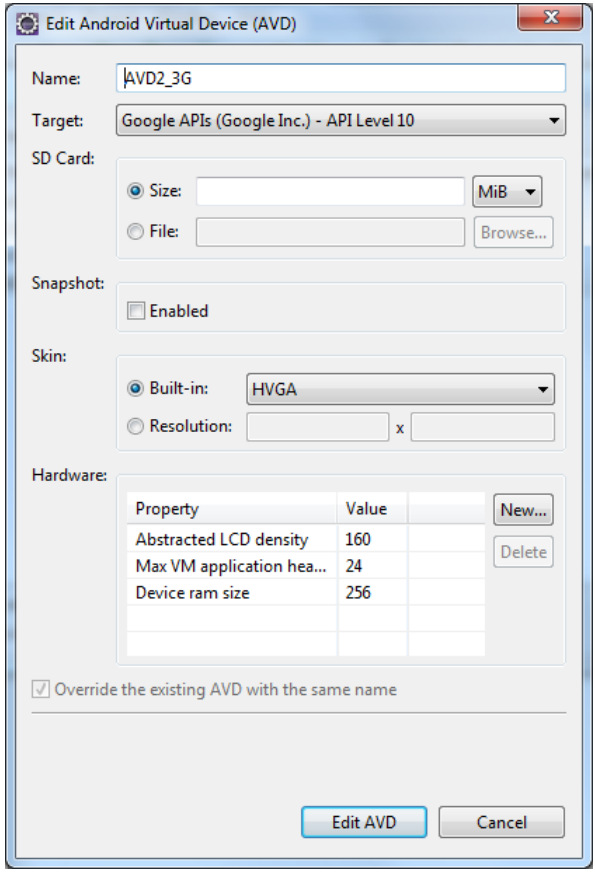


Рис. 8.1. Добавление в эмулятор поддержки Google API

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 координ-

нат, поддержка 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 на прилагаемом к книге компакт-диске в каталоге Ch08_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 8.1.

Листинг 8.1. 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 8.2.

Листинг 8.2. 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. 8.2.

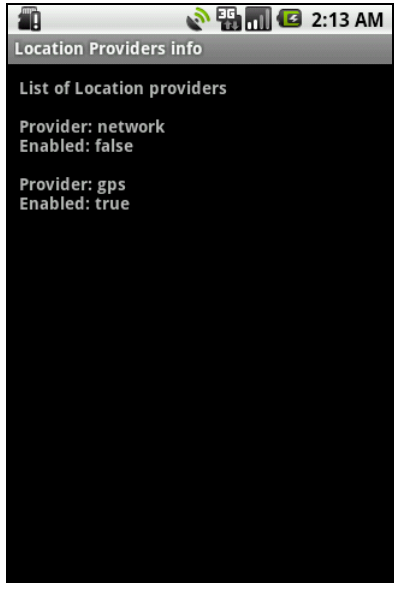


Рис. 8.2. Отображение провайдеров местоположения

Как вы видите, для эмулятора доступен провайдер GPS, и, в принципе, можно тестировать приложения для работы с сервисами местоположения на эмуляторе мобильного устройства Android.

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_LOW`;
- `ACCURACY_MEDIUM`;
- `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 на прилагаемом к книге компакт-диске в каталоге Ch08_Location_BestProviders.

В файл манифеста приложения AndroidManifest.xml добавьте разрешения `android.permission.ACCESS_FINE_LOCATION` и `android.permission.INTERNET`, как показано в листинге 8.3.

Листинг 8.3. 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 8.4.

Листинг 8.4. 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. 8.3.

Правда, в данном случае выбирать было не из чего: доступен всего один провайдер — GPS, но, конечно, существуют места, где возможно использование нескольких провайдеров местоположения.

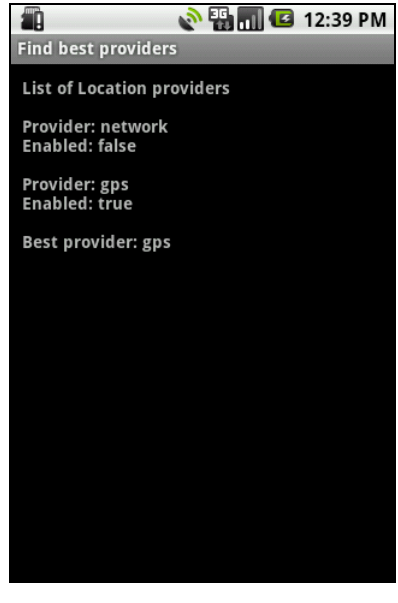


Рис. 8.3. Приложение для определения лучшего провайдера местоположения

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. 8.4).

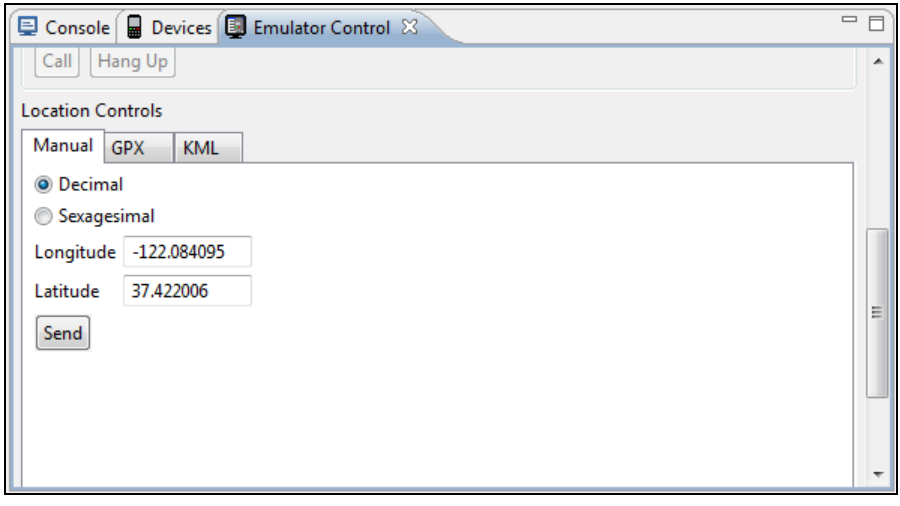


Рис. 8.4. Задание координат в представлении **Emulator Control**

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. 8.5.

Рис. 8.5. Отображение местоположения на карте

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.

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 является актуальным 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 описанный ранее способ использования интерфейса `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 на прилагаемом к книге компакт-диске в каталоге Ch08_Location_GetCoordinates.

В файл манифеста приложения AndroidManifest.xml добавьте разрешения, как вы уже делали ранее для приложений в этой главе (см. листинг 8.3). 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 обновлять вывод координат на экран мобильного устройства.

Файл класса окна приложения `GetCoordinatesActivity` представлен в листинге 8.5.

Листинг 8.5. 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("Longtitude:\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. 8.6.

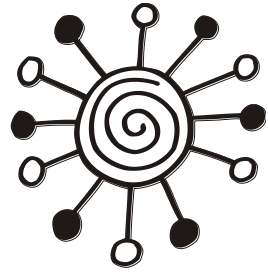
Рис. 8.6. Приложение для отображения координат

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.

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 следующих главах мы займемся более интересными вещами — сетевыми сервисами, предоставляемыми Google.



Глава 9

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:** Get geocoding;
- **Package name:** com.samples.location.reversegeocoding;
- **Create Activity:** GeocoderActivity.

> ***Примечание***
>
> Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch09_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 9.1.

Листинг 9.1. 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 представлено на рис. 9.1.

Рис. 9.1. 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 на прилагаемом к книге компакт-диске в каталоге Ch09_ReverseGeocodingWithMap.

В файле компоновки окна, кроме виджета `TextView` с идентификатором `text`, будет кнопка в верхней части окна для вызова стандартного Activity, содержащего карту. Назначьте для кнопки идентификатор `bMap` и надпись **Show Map**. Код для файла компоновки main.xml представлен в листинге 9.2.

Листинг 9.2. 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 9.3.

Листинг 9.3. 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 на кнопку **Show map** приложение отобразит это местоположение на карте, как показано на рис. 9.2.

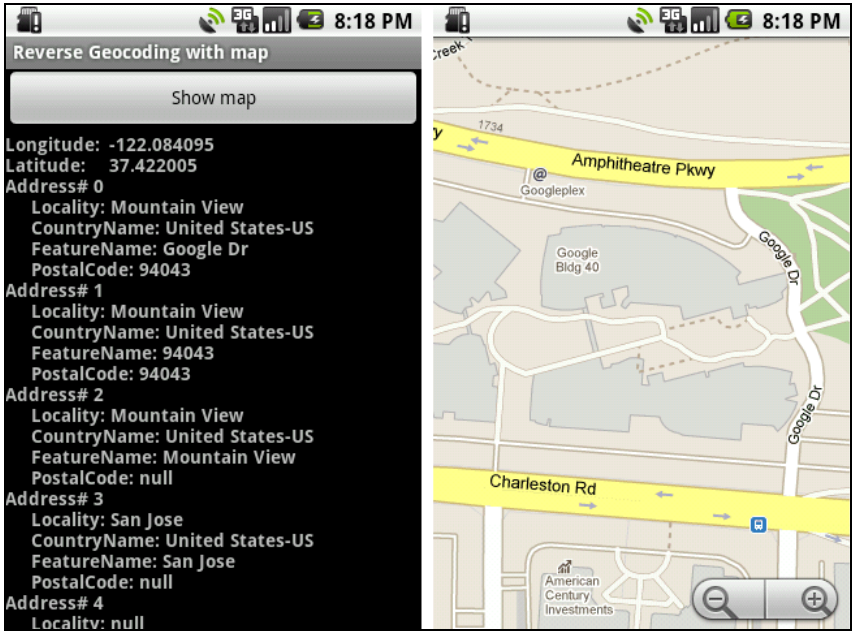


Рис. 9.2. Reverse Geocoding с отображением места на карте

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 на прилагаемом к книге компакт-диске в каталоге Ch09_ForwardGeocodingWithMap.

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 9.4.

Листинг 9.4. 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 `onListItemClick()`, в котором мы будем вызывать стандартный Activity для отображения найденного местоположения на карте.

Код класса `GeocoderActivity` представлен в листинге 9.5.

Листинг 9.5. 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.getFromLocationName(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 onListItemClick(
            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 этого названия 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. 9.3.

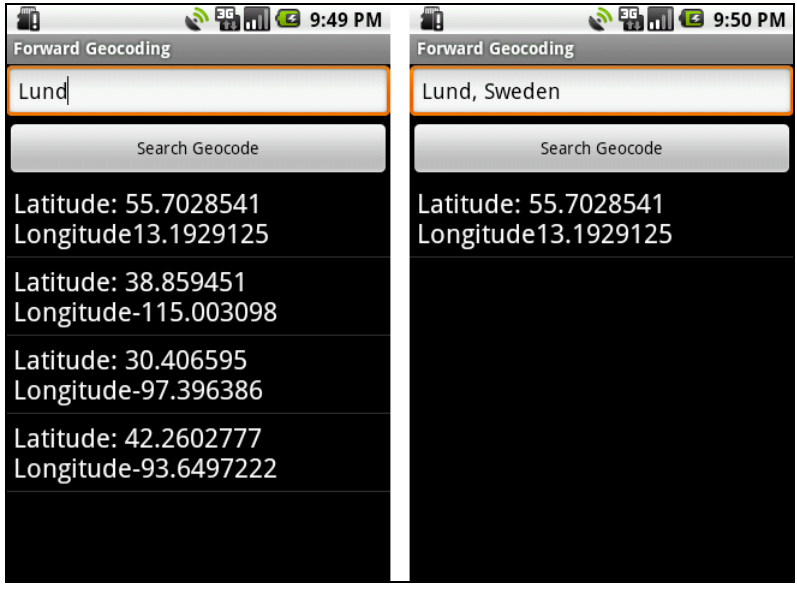


Рис. 9.3. Forward Geocoding — отображение списка возможных координат

Если выбрать первый элемент в списке, отобразится карта с городом Lund, расположенным в Швеции, как показано на рис. 9.4 (для тех, кто не в курсе, в этом городе расположено производство Sony Ericsson).

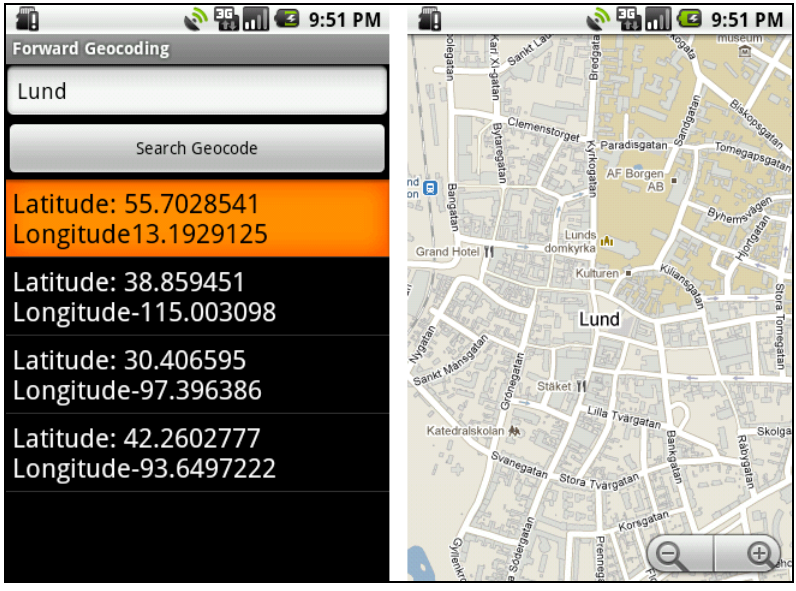
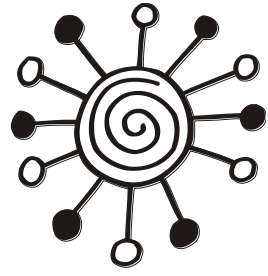


Рис. 9.4. Forward Geocoding с отображением выбранных координат на карте

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 10

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:

http://code.google.com/android/add-ons/google-apis/reference/index.html

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, 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. 10.1.

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. 10.2).

Теперь, имея сертификат, можно получить ключ Maps API Key. Для этого зайдите на страницу:

```
http://code.google.com/android/maps-api-signup.html
```

На этой странице находятся инструкции по получению Maps API Key, как представлено на рис. 10.3.

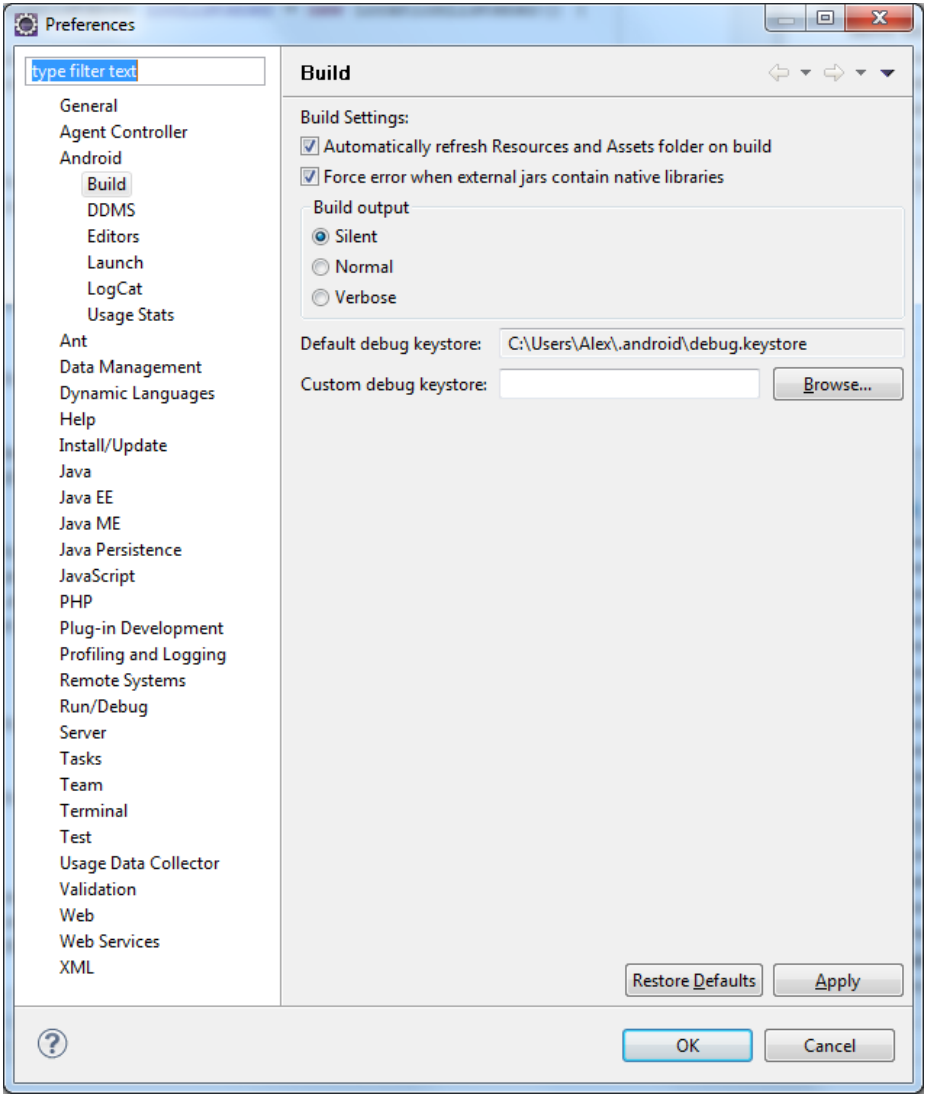


Рис. 10.1. Путь к хранилищу ключей

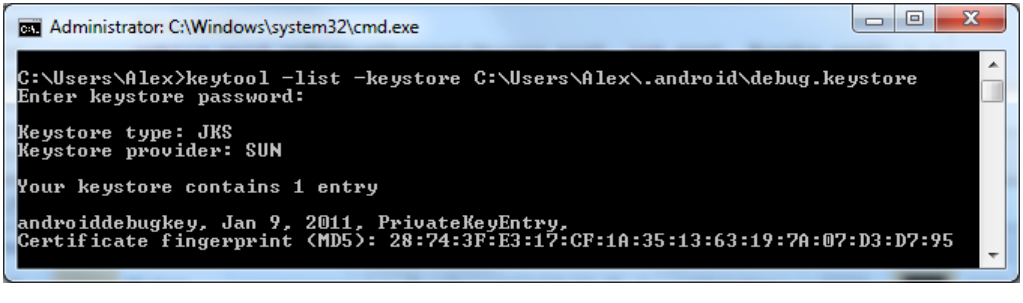


Рис. 10.2. Получение сертификата ключа

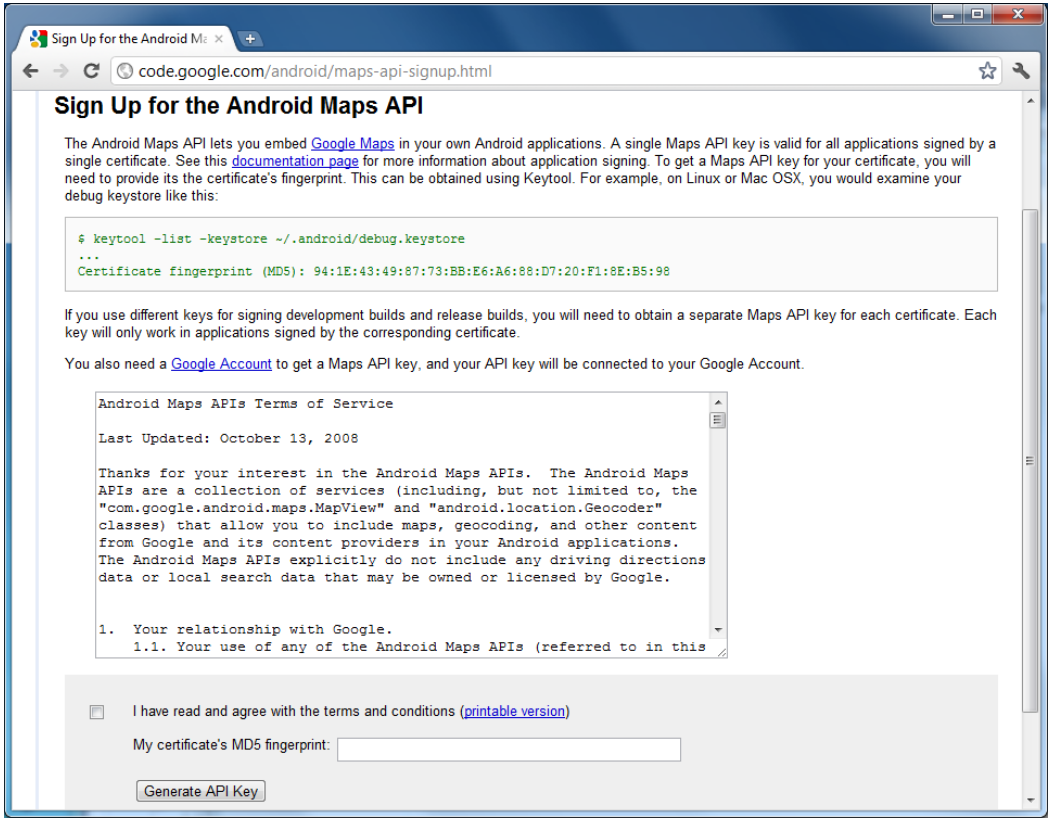


Рис. 10.3. Страница для генерации ключа

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.

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 с 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 широта-долгота.

При разработке приложений с 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 "спутник" 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 *MapActivity*

Виджет `MapView` может быть размещен только в Activity, которая наследуется от класса `MapActivity`.

Для использования в приложении создается производный класс, наследуемый от `MapActivity` (ранее, например, в *главе 5*, мы уже использовали похожие специализированные Activity, такие как `ListActivity` для создания списка):

```
public class MapViewActivity extends MapActivity {
    ...
}
```

В производном классе надо создать экземпляр `MapView` в теле метода `OnCreate()`, также как и любоой другой виджет.

> ***Примечание***
>
> В одном процессе желательно использовать только один экземпляр `MapActivity`, т. е. этот класс не является потокобезопасным.

Класс *MapController*

Для управления масштабированием и общим видом объекта `MapView` используется класс `MapController`. 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 с помощью 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 `MapActivity`. 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 на прилагаемом к книге компакт-диске в каталоге Ch10_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 10.1.

Листинг 10.1. 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 10.2.

Листинг 10.2. 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 10.3.

Листинг 10.3. 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. 10.4.

Рис. 10.4. 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 этого элемента в этом классе существует еще один метод `displayZoomControls()`.

Давайте используем элемент управления масштабированием в нашем приложении. Для этого в код класса `MapViewActivity` из листинга 10.3 добавим вызовы этих методов. Фрагмент метода `onCreate()` с внесенными добавлениями для управления масштабом карты представлен в листинге 10.4.

Листинг 10.4. 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. 10.5.

Рис. 10.5. Встроенный элемент управления масштабированием на карте

Добавление маркера

Когда вы работаете с сервисом Google Maps в интернет-браузере, вы видите, что сервис помечает 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 использовать 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 на прилагаемом к книге компакт-диске в каталоге Ch10_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 10.3, чтобы устанавливать маркер на заданной точке, представлены в листинге 10.5.

Листинг 10.5. 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));
        ...
}
```

Рис. 10.6. Маркер на карте, подгружаемый из внешнего ресурса

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. 10.6.

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 на прилагаемом к книге компакт-диске в каталоге Ch10_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 10.6.

Листинг 10.6. 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, внеся 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(SeekBar 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 10.7.

Листинг 10.7. 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. 10.7.

Рис. 10.7. Использование виджета `SeekBar` для плавного регулирования масштаба карты

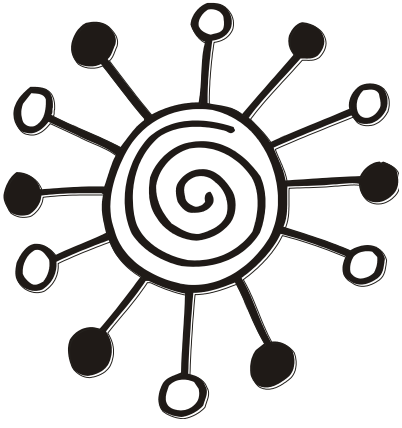
Используя вместо встроенного элемента управления масштабированием виджет `SeekBar`, мы можем плавно изменять масштаб карты, однако ползунок занимает довольно большую площадь на экране мобильного устройства, поэтому, как правило, для приложений с использованием карт применяют встроенный элемент масштабирования.

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 12*, когда будем изучать работу со встроенной видеокамерой мобильного устройства, мы рассмотрим создание собственных оверлеев в окне приложения.

Резюме

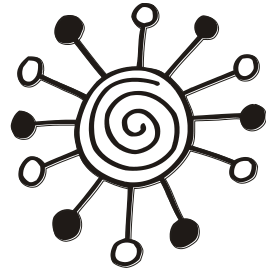
Использование встроенных карт Google Maps в приложениях открывает широкие возможности при создании различных приложений для мобильных устройств, используемых в области навигации. Как вы могли убедиться, такие приложения создавать очень легко, т. к. библиотеки Android SDK и Google API предлагают очень удобные компоненты, которые предоставляют хорошую базовую функциональность для использования в ваших приложениях.

В следующей части книги мы переходим к еще одной интересной теме — работе с "железом" мобильного телефона Android.



ЧАСТЬ V

Работа с оборудованием мобильного устройства



Глава 11

Карта памяти и файловая система

В этой части мы будем рассматривать важную тему — работу с оборудованием мобильного устройства: доступ и управление оборудованием из кода приложений.

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 откройте инструмент Android SDK and AVD Manager, создайте новый или измените существующий экземпляр Android Virtual Device. В окне **Edit Android Virtual Device (AVD)** в панели **SD Card** введите размер карты памяти в мегабайтах или килобайтах, как показано на рис. 11.1.

Затем нажмите кнопку **Edit AVD**. Теперь у нашего AVD есть подключенная карта памяти с объемом 1024 Мбайт, которую мы будем использовать для тестирования примеров приложений из этой главы.

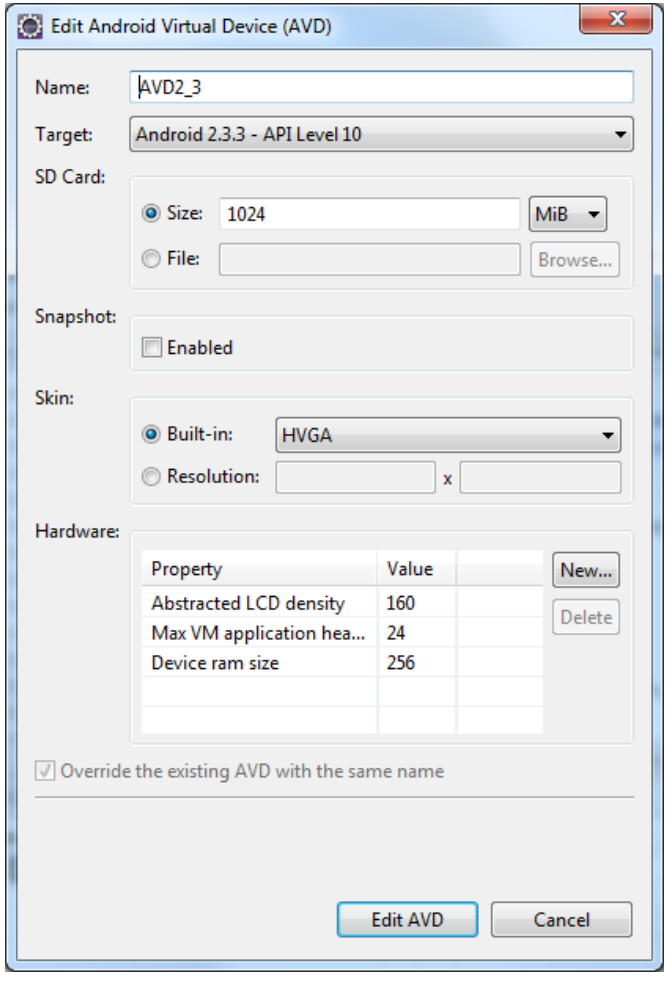


Рис. 11.1. Создание карты памяти в эмуляторе

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.

Для просмотра файловой системы Android вы можете воспользоваться инструментом 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. 11.2.

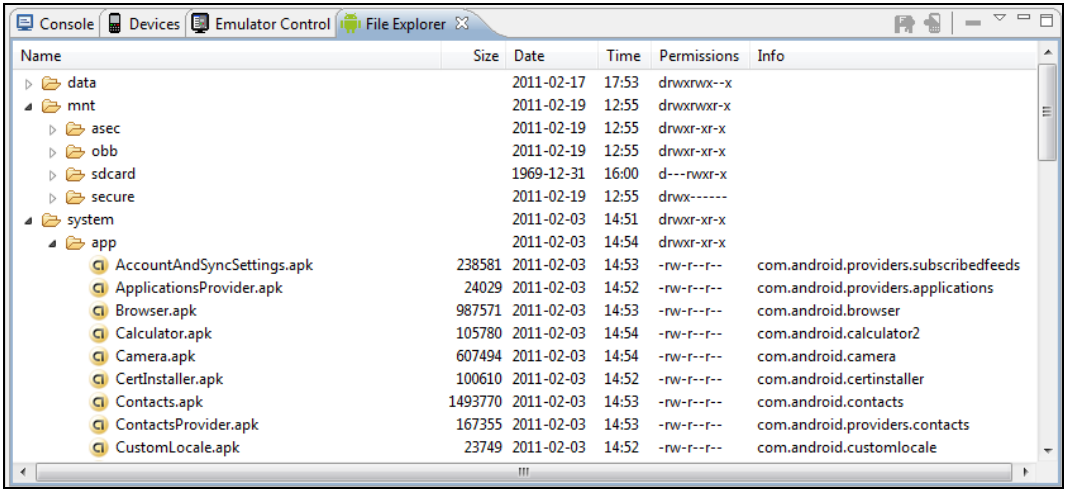


Рис. 11.2. Файловая система устройства Android

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 — 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 на прилагаемом к книге компакт-диске в каталоге Ch11_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 11.1.

Листинг 11.1. 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. 11.3.

Рис. 11.3. 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.

Сохранение и чтение файлов с SD-карты

Для изучения функциональности, предоставляемой системой Android для работы с файлами, создадим новое приложение. Это будет текстовый редактор, в котором пользователь может написать текстовый 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 в диалоговом окне **New Android Project**:

- **Project name:** SDCard_ReadWriteFiles;
- **Application name:** Wordpad;
- **Package name:** com.samples.sdcard.readwritefiles;
- **Create Activity:** EditorActivity.

> ***Примечание***
>
> Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch11_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 11.2.

Листинг 11.2. 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 у нас в приложении будут меню и диалоги, которые будут содержать достаточно большое количество надписей, все строковые ресурсы для удобства вынесем в файл strings.xml, код которого показан в листинге 11.3.

Листинг 11.3. 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 11.4.

Листинг 11.4. 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">

    <EditText
        android:id="@+id/edit"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:singleLine="false"/>

</LinearLayout>
```

Диалоговое окно для сохранения файла будет нестандартное, т. к. 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.

Файл компоновки окна диалога представлен в листинге 11.5.

Листинг 11.5. 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 у нас будет текстовый редактор `EditText` и меню из четырех опций:

- **New**;
- **Save**;
- **Open**;
- **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 11.6.

Листинг 11.6. 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 icicle) {
        super.onCreate(icicle);
        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 на 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. 11.4.

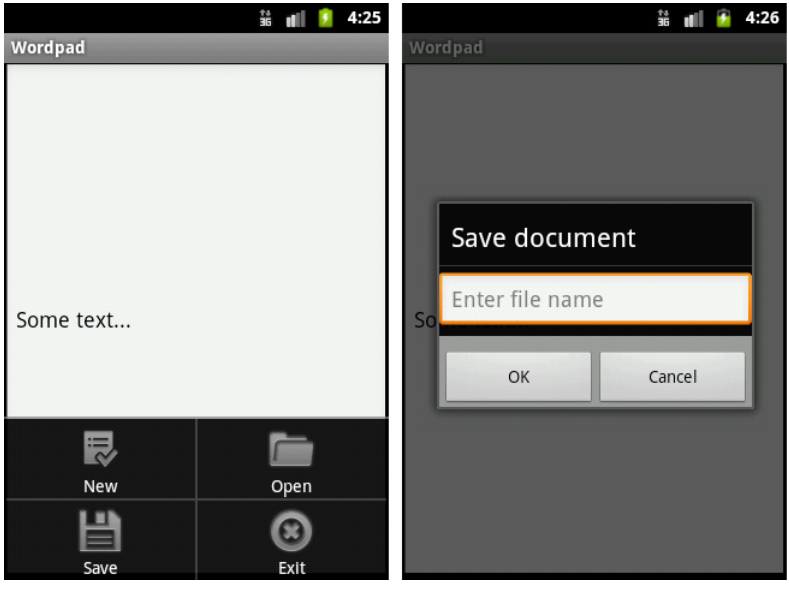


Рис. 11.4. Создание и сохранение документа

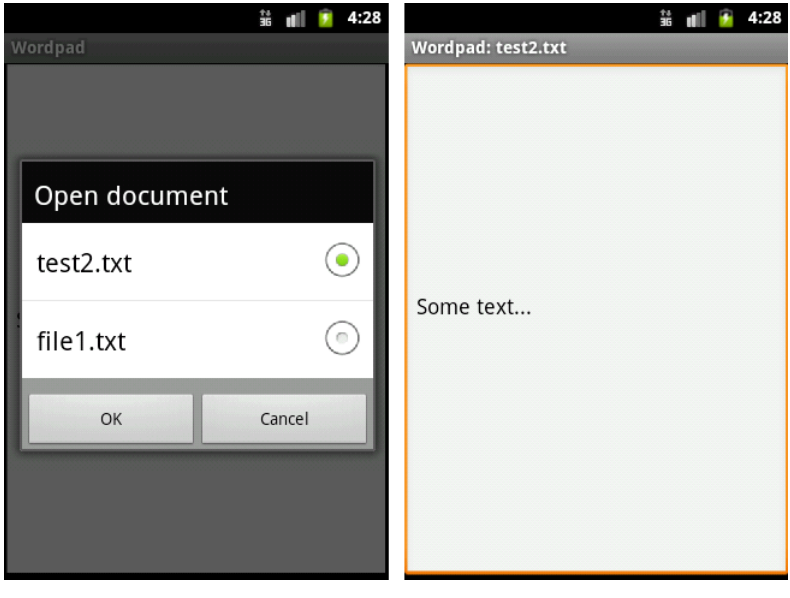


Рис. 11.5. Открытие документа

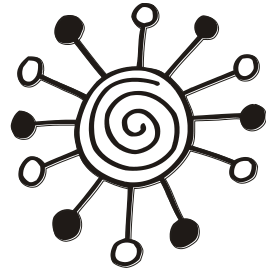
При открытии приложения отображается диалоговое окно открытия файла **Open document**, который выводит список текстовых файлов, находящихся в этой директории. С помощью этого диалога пользователь может выбрать и открыть в редакторе текстовый файл из списка (рис. 11.5).

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.

Резюме

В этой главе мы рассмотрели структуру файловой системы, доступ к файлам и директориям на мобильном устройстве. Мы рассмотрели подключение и использование внешней карты памяти, сохранение и открытие файлов из приложения.

В следующей главе мы рассмотрим еще одну интересную тему — подключение и управление из приложения встроенной видеокамерой мобильного телефона.



Глава 12

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, являются 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 следующий код:

```
// 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 видеокамеры, 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 видеокамеры, 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 притяжении 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 на прилагаемом к книге компакт-диске в каталоге Ch12_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 12.1.

Листинг 12.1. 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 12.2.

Листинг 12.2. 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. 12.1.

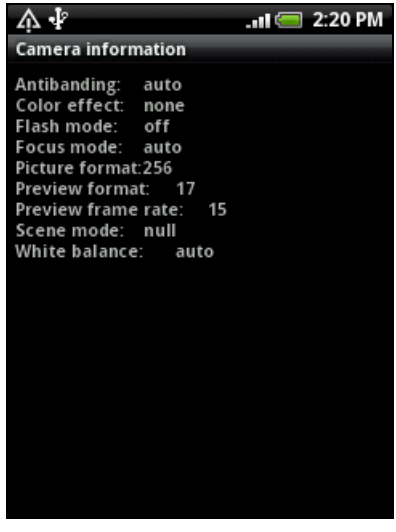


Рис. 12.1. Вывод информации о камере

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, имею-

щих большой разброс характеристик. В случае предъявления каких-либо специфических требований к встроенной камере приложение должно проводить проверку камеры 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.

Важной характеристикой 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 на прилагаемом к книге компакт-диске в каталоге Ch12_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 12.3.

Листинг 12.3. 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. 12.2.

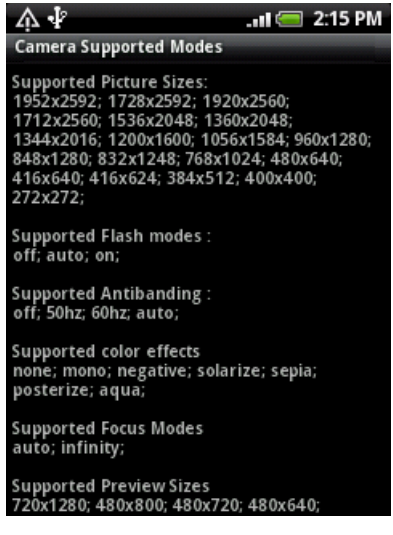


Рис. 12.2. Получение информации о режимах, поддерживаемых камерой

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.

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 камеры 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 Actions;
- **Package name:** com.samples.camera.callactivities;
- **Create Activity:** CallCameraActivity.

> ***Примечание***
>
> Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch12_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 12.4.

Листинг 12.4. 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 видеокамеры. Код класса `CallCameraActivity` представлен в листинге 12.5.

Листинг 12.5. 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;
        }

    }
}
```

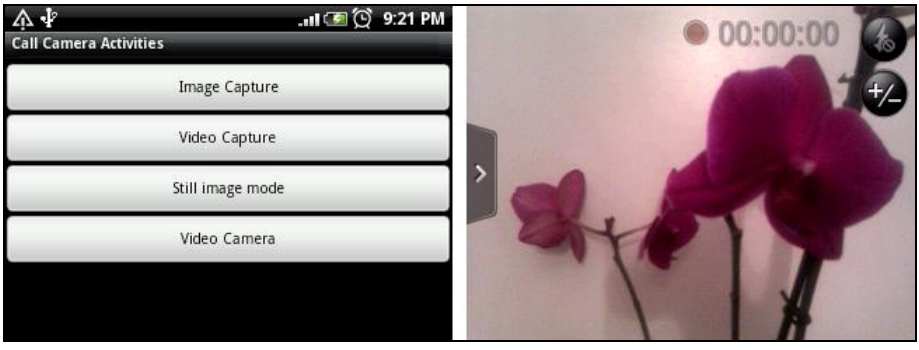


Рис. 12.3. Запуск стандартного Activity с камерой в режиме `ACTION_VIDEO_CAPTURE`

Скомпилируйте проект и запустите его на своем мобильном устройстве. Внешний вид приложения представлен на рис. 12.3.

Встраивание камеры в приложения

В предыдущем разделе мы использовали стандартный Activity для открытия и 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 видеокамеры.

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 видоискателя 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(surHolder);

    // 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 на прилагаемом к книге компакт-диске в каталоге Ch12_CameraPreview.

В файле компоновки главного окна приложения main.xml будет находиться виджет `SurfaceView` с идентификатором `surfaceview`, как показано в листинге 12.6.

Листинг 12.6. 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 12.7.

Листинг 12.7. Файл класса окна приложения CameraPreviewActivity.java

```
package com.samlpes.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 камера остановится и освободит ресурсы для других приложений. Внешний вид приложения представлен на рис. 12.4.

Рис. 12.4. 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 with manage preview;
- **Package name:** com.samples.camera.managepreview;
- **Create Activity:** CameraPreviewActivity.

Примечание

Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch12_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 12.8.

Листинг 12.8. 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 останова 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 12.9.

Листинг 12.9. 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 видеокамера запускается только при нажатии кнопки **Start**. После запуска видеокамеры кнопка **Start** блокируется во избежание повторного нажатия и одновременно становится доступной кнопка **Stop** для возможности останова видеокамеры. Внешний вид приложения представлен на рис. 12.5.

Рис. 12.5. 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 видоискателя, 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. 12.5), 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 на прилагаемом к книге компакт-диске в каталоге Ch12_CameraOverlay.

Код файла main.xml остается таким же, как в листинге 12.6. 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 Ch12_CameraOverlay/res/drawable.

Код файла overlay.xml представлен в листинге 12.10.

Листинг 12.10. 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 12.11.

Листинг 12.11. 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;
        }
    }
}
```

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. 12.6.

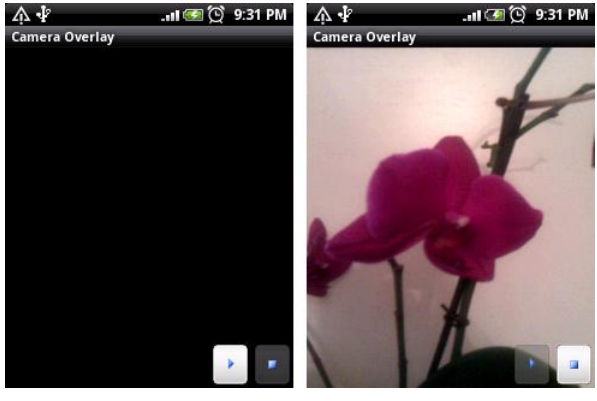


Рис. 12.6. Приложение с оверлеями для управления камерой

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 вызывается 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
        // в 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 на прилагаемом к книге компакт-диске в каталоге Ch12_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 на которую будет происходить захват изображения. Файл компоновки окна представлен в листинге 12.12.

Листинг 12.12. 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 12.13.

Листинг 12.13. 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 видеокамеры. Внешний вид приложения представлен на рис. 12.7.

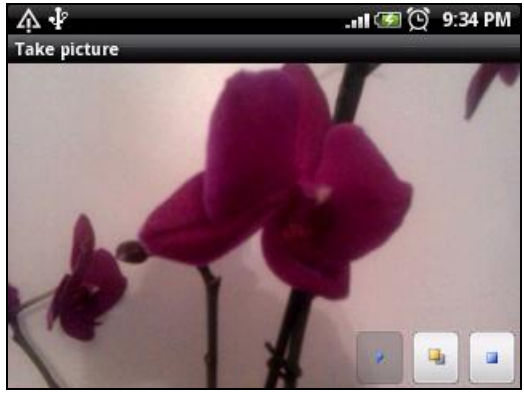


Рис. 12.7. Приложение с захватом изображения камерой

Использование автофокуса

Если встроенная камера поддерживает режим автофокуса, вы можете использовать этом режим в своем приложении, например, производить захват изображения только в том случае, если фотографируемый объект находится в фокусе. Дело в том, что для настойки фокуса камере необходимо некоторое время, особенно при резких изменениях расстояния до объекта.

Для отслеживания настройки фокуса необходимо 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:** com.samples.camera.autofocus;
- **Package name:** Set Autofocus;
- **Create Activity:** CameraPreviewActivity.

> ***Примечание***
>
> Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch12_CameraAutoFocus.

В коде класса `CameraPreviewActivity` добавим функциональность для отслеживания автофокуса. Код класса `CameraPreviewActivity` приложения представлен в листинге 12.14.

Листинг 12.14. 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);

        // устанавливаем блокирование кнопок
        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;
        }
    }
}
```

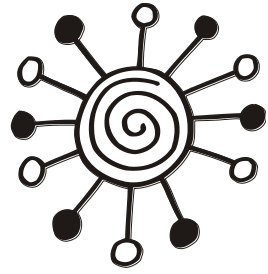
Скомпилируйте проект и запустите его на мобильном устройстве. Теперь приложение сможет производить захват изображения с камеры только в том случае, если изображение находится в фокусе. Если изображение расфокусировано, кнопка захвата изображения будет недоступна. При установке фокуса кнопка захвата изображения будет разблокирована, и можно будет произвести захват изображения. Внешний вид приложения представлен на рис. 12.8.

Рис. 12.8. Приложение с отслеживанием автофокуса

Резюме

В этой главе мы научились работать с 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 13

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 у 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.

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()`:

```
SensorManager sensors =
    (SensorManager)getSystemService(Context.SENSOR_SERVICE);
```

Класс `SensorManager` содержит набор констант, которые определяют характеристики датчиков мобильного 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 2.3.3 в классе `Sensor` содержится следующий набор констант, определяющих типы датчиков:

- `TYPE_ACCELEROMETER` — акселерометр, который возвращает текущее ускорение для трех осей измерения;
- `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_ROTATION_VECTOR` — датчик ротации. Этот датчик представляет ориентацию мобильного устройства в виде комбинации угла поворота и оси;
- `TYPE_TEMPERATURE` — датчик температуры;
- `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 данного 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 этим 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 используйте этот 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 ситуации, 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 устройтве, 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 на прилагаемом к книге компакт-диске в каталоге Ch13_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 13.1.

Листинг 13.1. 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 и их типы, представленный на рис. 13.1.

Конечно, для другого мобильного устройства набор датчиков будет отличаться от представленного на рис. 13.1.

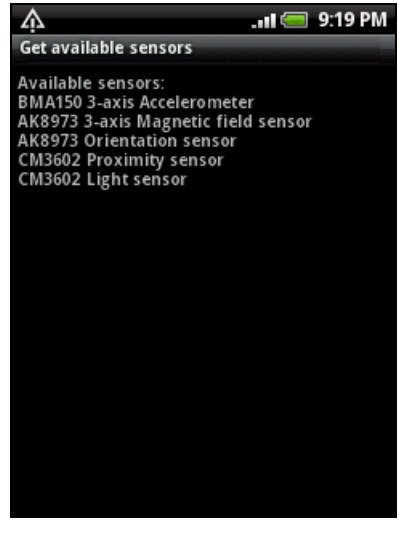


Рис. 13.1. Список доступных сенсоров на мобильном устройстве

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. Входными параметрами служат объект класса `Sensor`, представляющий датчик, второй представляет целое число, которое соответствует новому значению точности этого датчика. Это 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;
- целое число, устанавливающее 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 используется 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 проч.).

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 на прилагаемом к книге компакт-диске в каталоге Ch13_LightSensor.

Это 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 13.2.

Листинг 13.2. 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 13.3.

Листинг 13.3. 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 13.4.

Листинг 13.4. 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. 13.2.

Как уже говорилось ранее в этой главе, для приложений, работающих с оборудованием мобильного телефона, обязательно нужно предусматривать корректную обработку ситуации, когда датчик данного типа отсутствует на мобильном устройстве. В коде класса окна приложения из листинга 13.4 в случае отсутствия датчика предусмотрено отображение диалогового окна с соответствующим сообщением, и после нажатия пользователем на кнопку **OK** 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. 13.3.

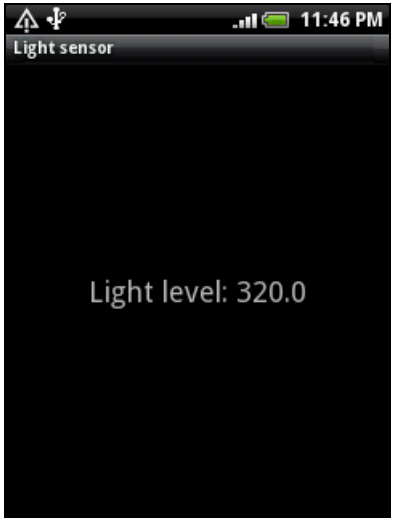


Рис. 13.2. Измерение уровня освещенности

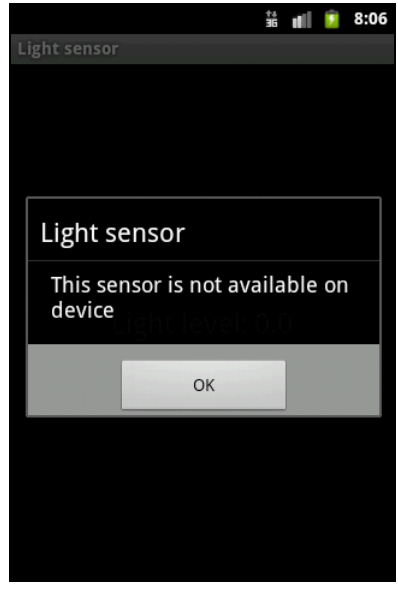


Рис. 13.3. Диалоговое окно при отсутствии датчика данного типа на устройстве

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 на прилагаемом к книге компакт-диске в каталоге Ch13_ProximitySensor.

Для работы приложения с датчиком расстояния измените одну строку кода в методе `onCreate()` класса `SensorActivity` (см. листинг 13.4), как показано в листинге 13.5.

Листинг 13.5. 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 13.6.

Листинг 13.6. 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. 13.4.

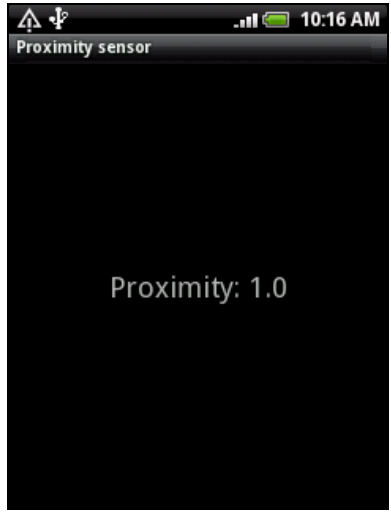


Рис. 13.4. Измерение расстояния до объекта

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 на прилагаемом к книге компакт-диске в каталоге Ch13_OrientationSensor_Azimuth.

Совсем необязательно для работы с приложением использовать все три параметра. Например, этот датчик можно использовать 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 13.4. нам изменять не надо, достаточно лишь изменить доступ к датчику ориентации в классе `SensorActivity`, как показано в листинге 13.7.

Листинг 13.7. 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 13.8.

Листинг 13.8. 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>
```

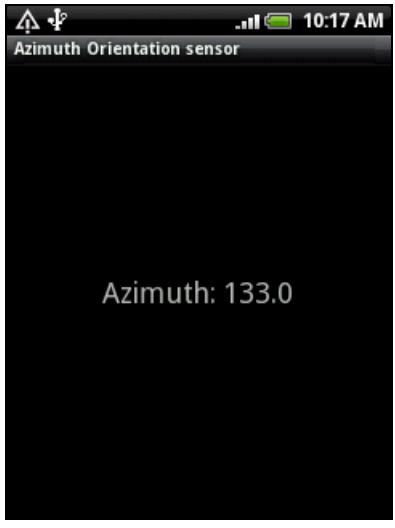


Рис. 13.5. Изменение ориентации в пространстве

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. 13.5.

Используя датчик ориентации, из мобильного устройства можно сделать измеритель уровня наклона поверхности. Для этого нам потребуется небольшая модификация нашего приложения.

> ***Примечание***
>
> Полный код приложения находится на прилагаемом к книге компакт-диске в каталоге Ch13_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 13.9.

Листинг 13.9. 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 13.10.

Листинг 13.10. 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 13.11.

Листинг 13.11. 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 поверхности. Внешний вид приложения для работы с датчиком ориентации для измерения уровня наклона поверхности представлен на рис. 13.6.

Рис. 13.6. 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 устойства;
- Lateral — ускорение, направленное поперек корпуса мобильного устойства;
- 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` определена 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 на прилагаемом к книге компакт-диске в каталоге Ch13_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 13.12.

Листинг 13.12. 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 13.13.

Листинг 13.13. 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 13.14.

Листинг 13.14. 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. 13.7.

Рис. 13.7. Измерение ускорения по трем осям

Как видно из рисунка, датчик акселерометра показывает значение ускорения свободного падения по оси Z, с учетом погрешности измерения примерно равное 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.

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` и `MAGNETIC_FIELD_EARTH_MIN`, которые 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 на прилагаемом к книге компакт-диске в каталоге Ch13_MagneticField3DSensor.

Файл строковых ресурсов приложения для текстовых полей и текста диалогового окна представлен в листинге 13.15.

Листинг 13.15. 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 13.16.

Листинг 13.16. 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. 13.8.

Рис. 13.8. 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 этой главе, и при созданнии 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. 13.9.

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. 13.10. Значения IP-адреса и номера порта надо взять из окна **Sensor Simulator** в левой панели (см. рис. 13.9).

Затем перейдите на вкладку **Testing** приложения Sensor Simulator на эмуляторе Android и протестируйте соединение, нажав кнопку **Connect** (рис. 13.11). 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 этот инструмент, при желании могут познакомиться с ним поближе и использовать его при тестировании своих приложений.

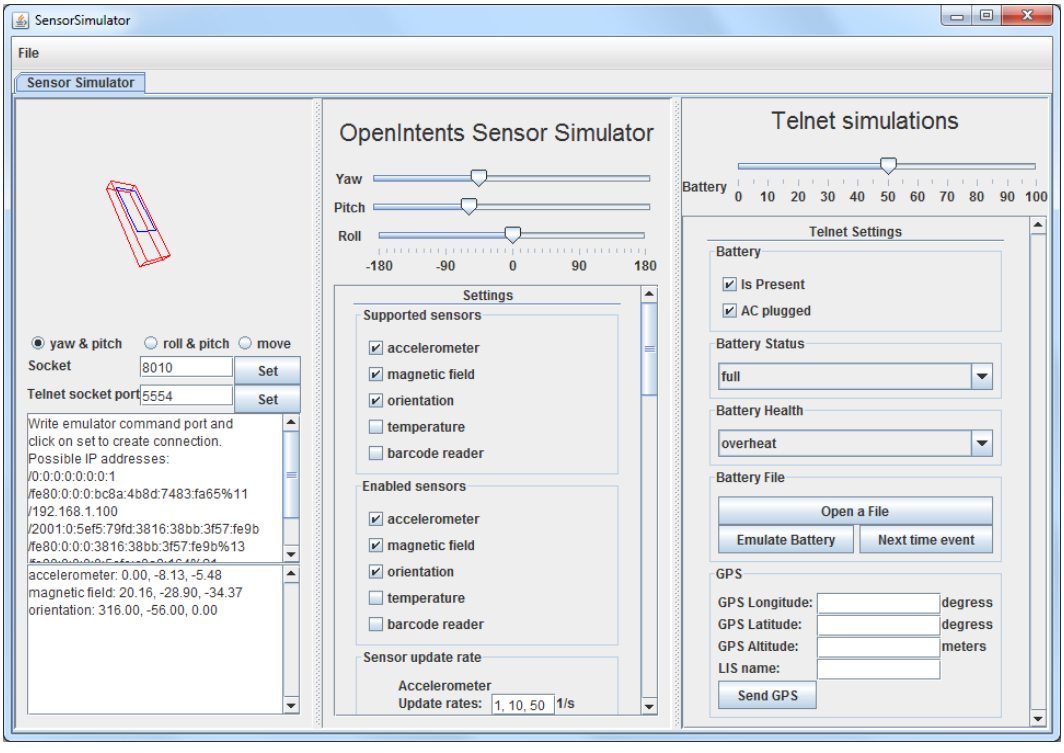


Рис. 13.9. Инструмент Sensor Simulator от OpenIntents

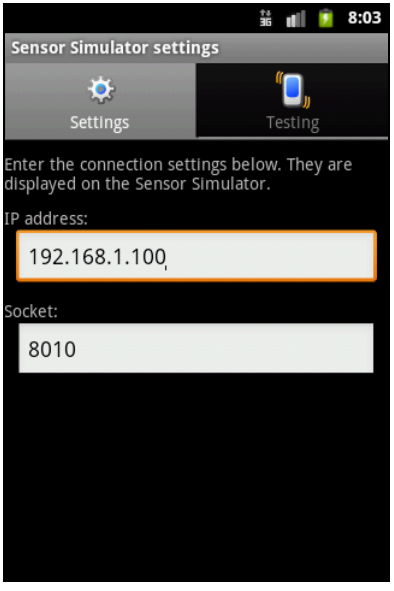


Рис. 13.10. Настройка IP-адреса и номера порта имитатора

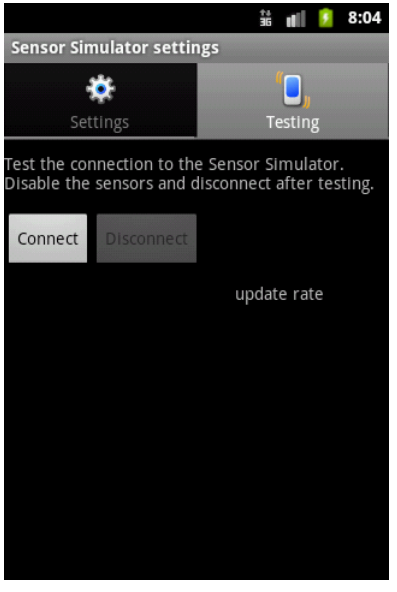
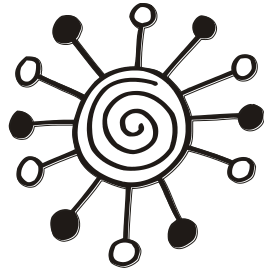


Рис. 13.11. Тестирование соединения с имитатором

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 14

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.

Доступ к дисплею мобильного устройства

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 пикселов.

Класс `DisplayMetrics` содержит набор открытых полей, содержащих метрики 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 пикселов на дюйм экрана по оси X (по ширине экрана);
- `ydpi` — точное физическое количество пикселов на дюйм экрана по оси Y (по высоте экрана);
- `densityDpi` — плотность пикселов на дисплее в dpi (dots-per-inch, количество пикселов на один дюйм).

Поле `densityDpi`, в отличие от полей `xdpi` и `ydpi`, возвращает не точную физическую плотность пикселов, а обобщенную величину плотности пикселов на экране, которая используется для качественной оценки свойств дисплея. Эта величина характеризуется 4-мя константами, представляющими типы дисплеев, которые устанавливаются на мобильных устройствах:

- `DENSITY_LOW` — низкая плотность пикселов;
- `DENSITY_MEDIUM` — средняя плотность пикселов;
- `DENSITY_HIGH` — высокая плотность пикселов;
- `DENSITY_XHIGH` — сверхвысокая плотность пикселов.

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 требованием, 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 для получения характеристик дисплея в коде приложения, создав 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 на прилагаемом к книге компакт-диске в каталоге Ch14_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 прочитать метрики дисплея мобильного устройства. Код класса главного окна приложения представлен в листинге 14.1.

Листинг 14.1. 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. 14.1.

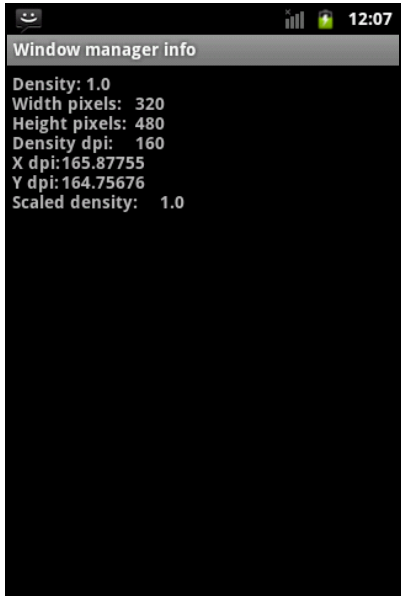


Рис. 14.1. Вывод информации о дисплее

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 на прилагаемом к книге компакт-диске в каталоге Ch14_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 14.2.

Листинг 14.2. 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 идентификатом `seek` для регулировки яркости экрана;
- текстовое поле `TexView` с идентификатом `text`;
- кнопку `Button` с идентификатом `bSetBright`.

Файл компоновки окна представлен в листинге 14.3.

Листинг 14.3. 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"/>

</LinearLayout>
```

В коде класса `ScreenBrightessActivity` 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 `ScreenBrightessActivity` представлен в листинге 14.4.

Листинг 14.4. Файл класса окна приложения ScreenBrightessActivity.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.View.OnClickListener;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
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);
        bSetBright = (Button)findViewById(R.id.bSetBright);

        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 + "%");

        bSetBright.setOnClickListener(this);
        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. 14.2.

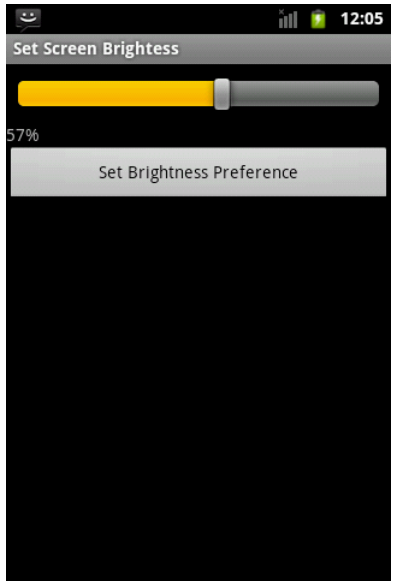


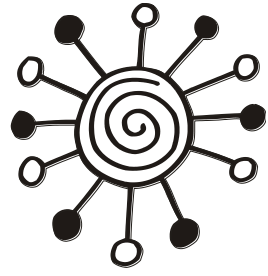
Рис. 14.2. Приложение, управляющее яркостью дисплея

Резюме

В этой главе мы изучили возможности использования службы Window Service в своих приложениях. Эту службу можно использовать для получения сведений о дисплее, используемом на мобильном телефоне, на который инсталлировано 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 15

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, определяющими 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 на прилагаемом к книге компакт-диске в каталоге Ch15_BatteryManagerInfo.

В файл манифеста приложения AndroidManifest.xml не забудьте добавить разрешение `android.permission.BATTERY_STATS`. Код файла AndroidManifest.xml представлен в листинге 15.1.

Листинг 15.1. 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 и текстовое поле `TextView` с идентификатором `text` для вывода результатов.

Файл компоновки окна представлен в листинге 15.2.

Листинг 15.2. 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:text="Start"
         android:layout_width="fill_parent"
         android:layout_weight="1"/>
      <Button
         android:id="@+id/bStop"
         android:layout_height="wrap_content"
         android:layout_width="fill_parent"
         android:text="Stop"
         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"
        android:layout_margin="10px"/>
</LinearLayout>
```

В коде класса главного окна приложения для кнопок `dStart` 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 15.3.

Листинг 15.3. 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.Button;
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);

        bStart = (Button)findViewById(R.id.bStart);
        bStop = (Button)findViewById(R.id.bStop);
```

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

        bStart.setOnClickListener(this);
        bStop.setOnClickListener(this);
        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();
    }
}
```

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. Внешний вид приложения представлен на рис. 15.1.

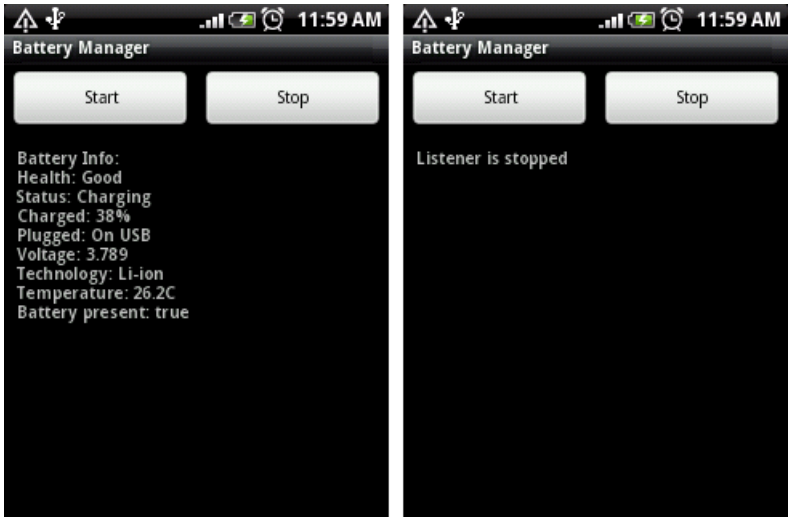


Рис. 15.1. Получение информации о состоянии батареи

Отображение статистики использования батареи

Если вам требуется в приложении вывести суммарную статистику использования батареи на мобильном 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 на прилагаемом к книге компакт-диске в каталоге Ch15_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 15.4.

Листинг 15.4. 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: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 15.5.

Листинг 15.5. 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;
import android.view.View.OnClickListener;
import android.widget.Button;

public class BatteryUsageActivity extends Activity
    implements OnClickListener {

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

        final Button bPowerUsage = (Button)findViewById(R.id.bBatteryUsage);
        bPowerUsage.setOnClickListener(this);
    }

    @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. 15.2). Конечно, это окно зависит от модели телефона и может отличаться от представленного на рисунке.

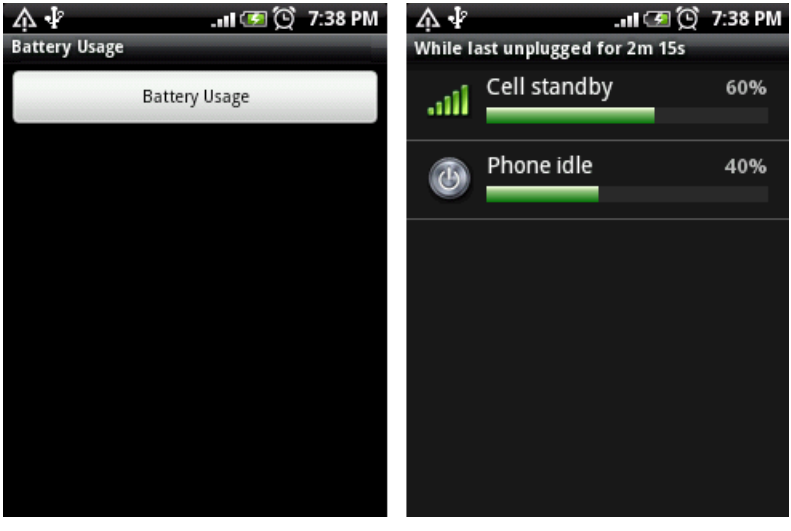
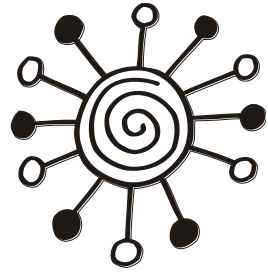


Рис. 15.2. Вызов стандартного Activity, отображающего использование батареи

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 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 16

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 режимом 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 из них:

- `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) и 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 отрабатывает 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 `aquire()` и `release()`. Каждый вызов метода `aquire()` 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 на прилагаемом к книге компакт-диске в каталоге Ch16_PowerManager.

В файл манифеста приложения AndroidManifest.xml добавьте разрешение `android.permission.WAKE_LOCK`, как показано в листинге 16.1.

Листинг 16.1. 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 16.2.

Листинг 16.2. 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 16.3.

Листинг 16.3. 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: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: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: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: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 16.4.

Листинг 16.4. 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.view.View.OnClickListener;
import android.widget.Button;
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);

        final Button bPartialWL = (Button)findViewById(R.id.bPartialWL);
        final Button bScreenBrightWL = (Button)findViewById(
                R.id.bScreenBrightWL);
        final Button bScreenDimWL = (Button)findViewById(R.id.bScreenDimWL);
        final Button bFullWL = (Button)findViewById(R.id.bFullWL);

        bPartialWL.setOnClickListener(this);
        bScreenBrightWL.setOnClickListener(this);
        bScreenDimWL.setOnClickListener(this);
        bFullWL.setOnClickListener(this);

        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. 16.1.

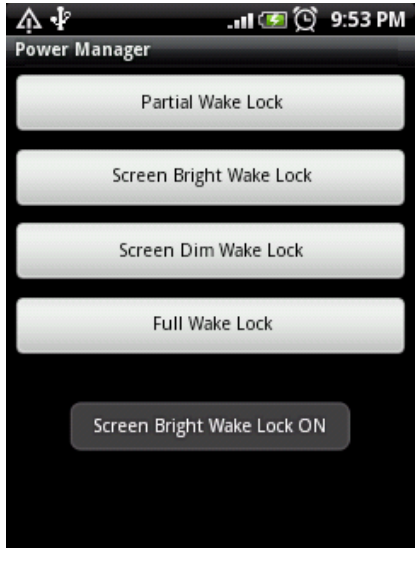


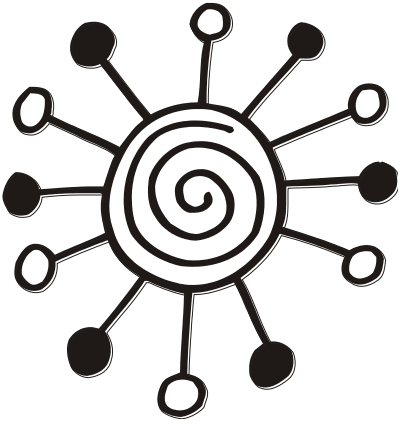
Рис. 16.1. Приложение для управления блокировками

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.

Резюме

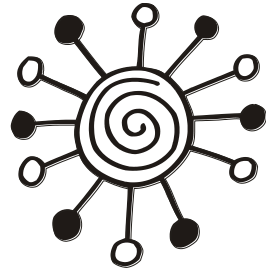
В этой главе мы рассмотрели использование менеджера энергопотребления для управления процессором, экраном и клавиатурой мобильного устройства. С помощью менеджера энергопотребления можно управлять мобильным телефоном с целью уменьшения потребления им энергии или, если для приложения требуются специфические варианты управления экраном и клавиатурой, создавать соответствующие блокировки.

В следующей, заключительной части мы переходим к изучению различных системных сервисов для управления и мониторинга системы Android и интерактивного взаимодействия с пользователем.



ЧАСТЬ VI

Системные сервисы



Глава 17

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 этого 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 этого менеджера. Это 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 на прилагаемом к книге компакт-диске в каталоге Ch17_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 17.1.

Листинг 17.1. 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 17.2.

Листинг 17.2. 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 17.3.

Листинг 17.3. 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.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 onListItemClick(
        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 17.4.

Листинг 17.4. 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, будет структура 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 17.5.

Листинг 17.5. 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. 17.1.

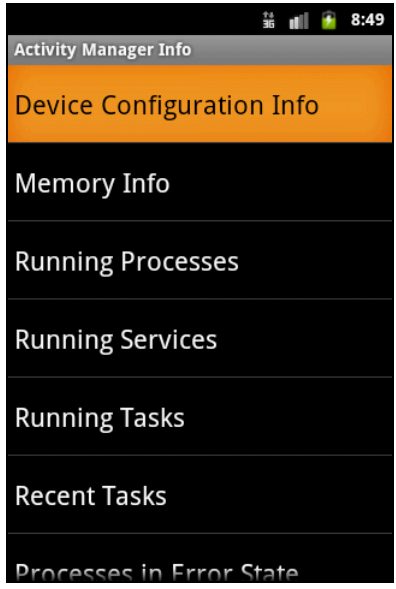


Рис. 17.1. Окно приложения со списком опций

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.

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 смартофонов почти во все 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 токосъемных 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 17.6 (это дополнение для класса `SystemInfoItemActivity` из листинга 17.5).

Листинг 17.6. 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;
        }
}
```

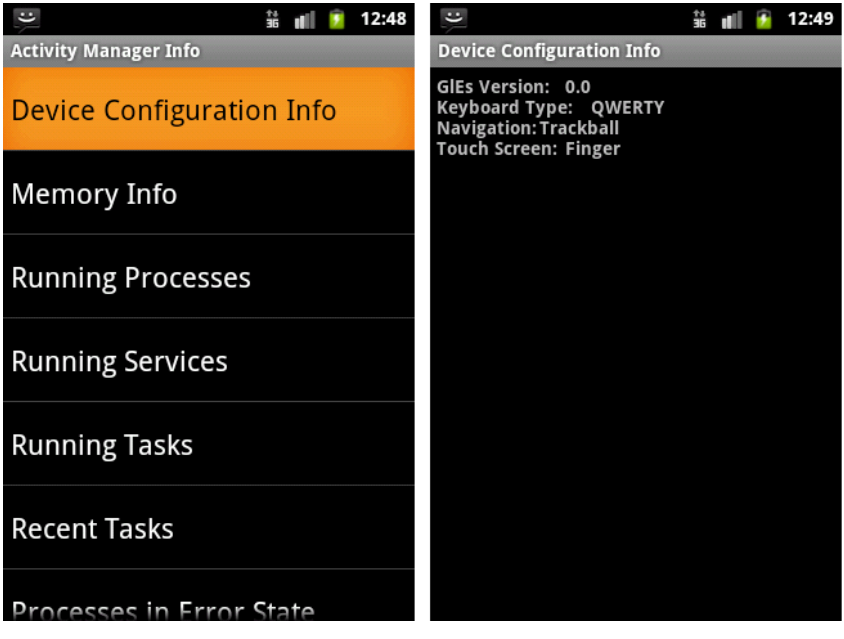


Рис. 17.2. Вывод информации о конфигурации устройства

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. 17.2.

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 текущей 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 прочитать данные о распределении памяти.

Пример использования объекта `ActivityManager.MemoryInfo` в нашем приложении в коде класса `SystemInfoItemActivity` представлен в листинге 17.7.

Листинг 17.7. 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. 17.3.

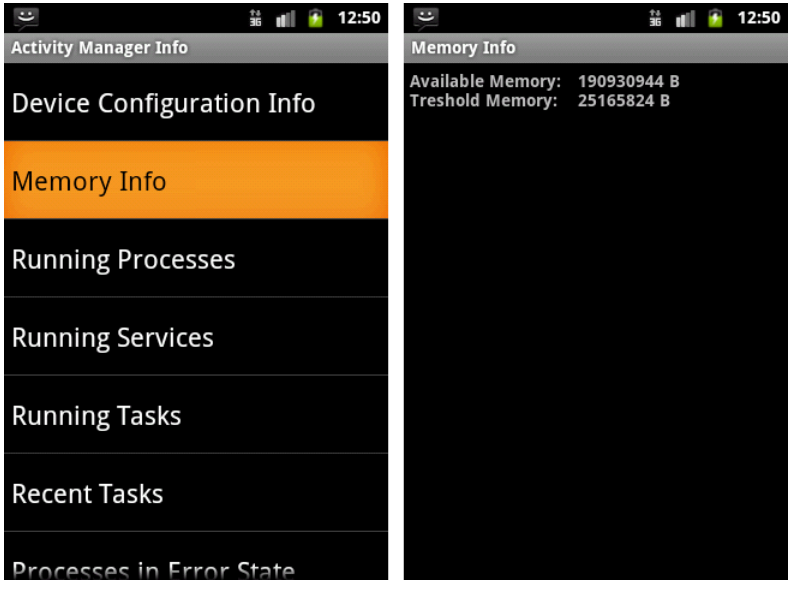


Рис. 17.3. Вывод информации о доступной памяти

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.

> ***Примечание***
>
> Условия выполнения процессов в системе Android и их приоритеты подробно были рассмотрены в [1].

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 17.8.

Листинг 17.8. 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);
    }
}
```

Выполните компиляцию приложения, запустите его на эмуляторе и откройте опцию **Running Processes**. На рис. 17.4. 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.

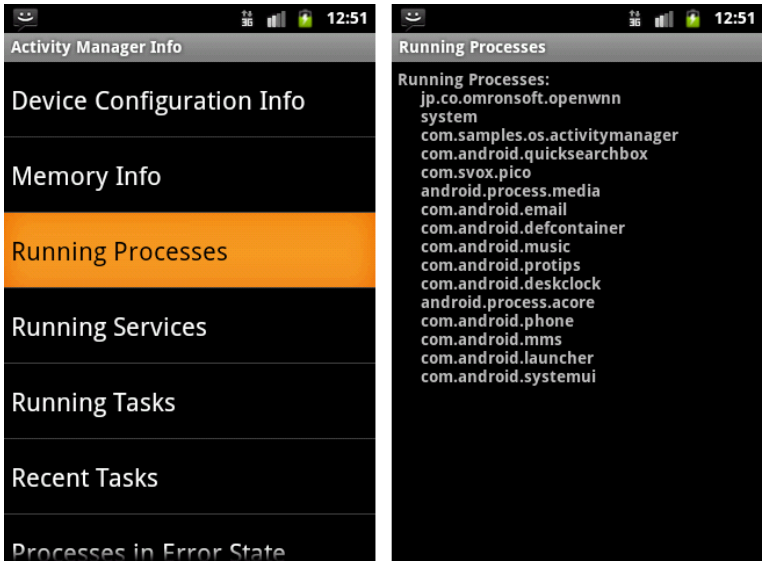


Рис. 17.4. Вывод информации о выполняющихся процессах

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 17.9.

Листинг 17.9. 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);
    }
}
```

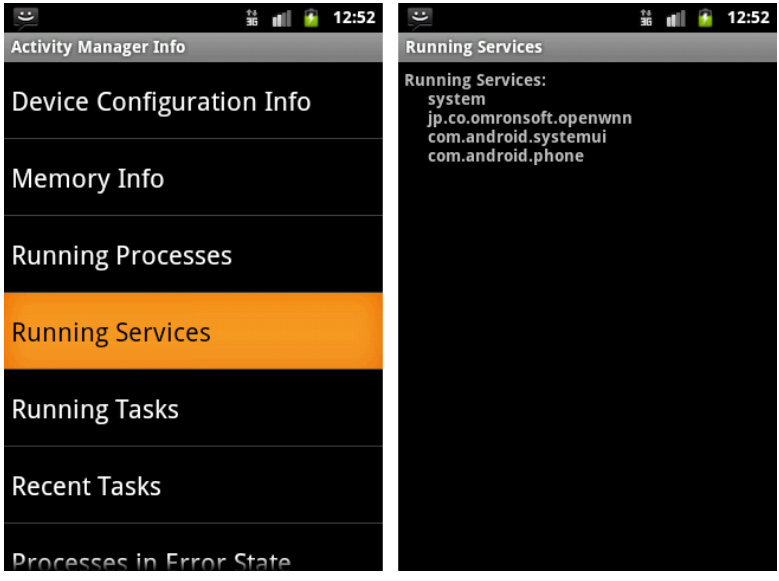


Рис. 17.5. Вывод информации о выполняющихся службах

Если запустить приложение и открыть опцию **Running Services**, то информация, отображающая работающие в системе службы, будет выглядеть так, как представлено на рис. 17.5.

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 17.10.

Листинг 17.10. 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. 17.6.

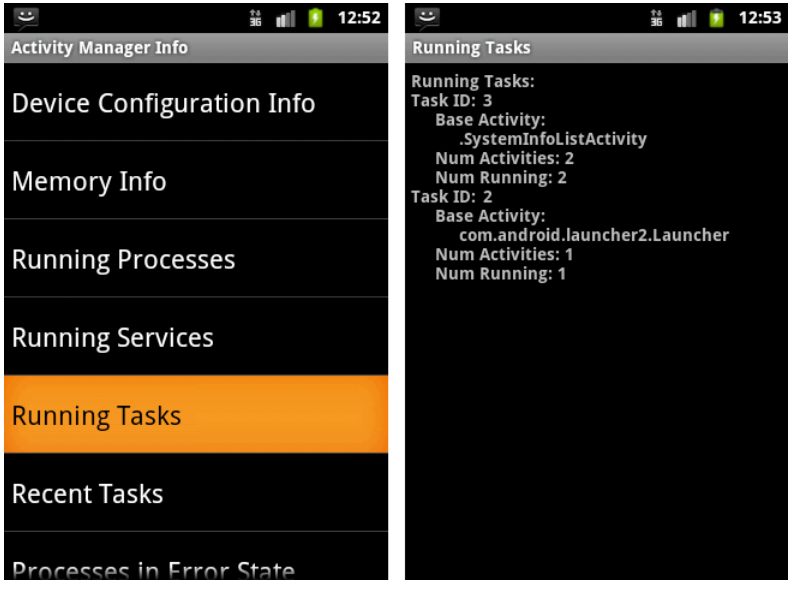


Рис. 17.6. Вывод информации о выполняющихся заданиях

Последние выполненные задания

Класс `RecentTaskInfo` предоставляет информацию о последних заданиях, которые запускал пользователь или в которые он заходил.

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 17.11.

Листинг 17.11. 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());
        }
    }
}
```

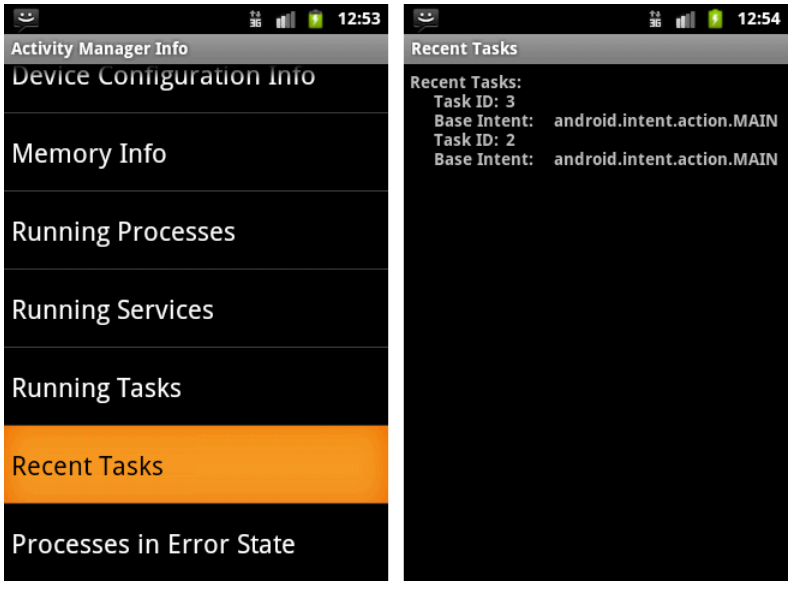


Рис. 17.7. Вывод информации о последних выполненных заданиях

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. 17.7.

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 17.12.

Листинг 17.12. 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. 17.8.

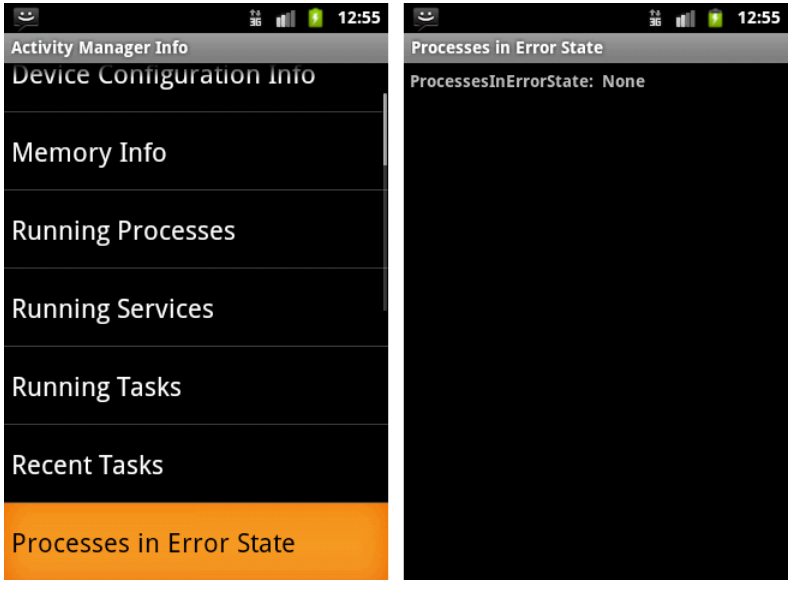


Рис. 17.8. Вывод информации о процессах в состоянии ошибки

Примечание

Если процессы в состоянии ошибки отсутствуют, метод `getProcessesInErrorState()` возвращает `null`, а не пустой список.

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. 17.9.

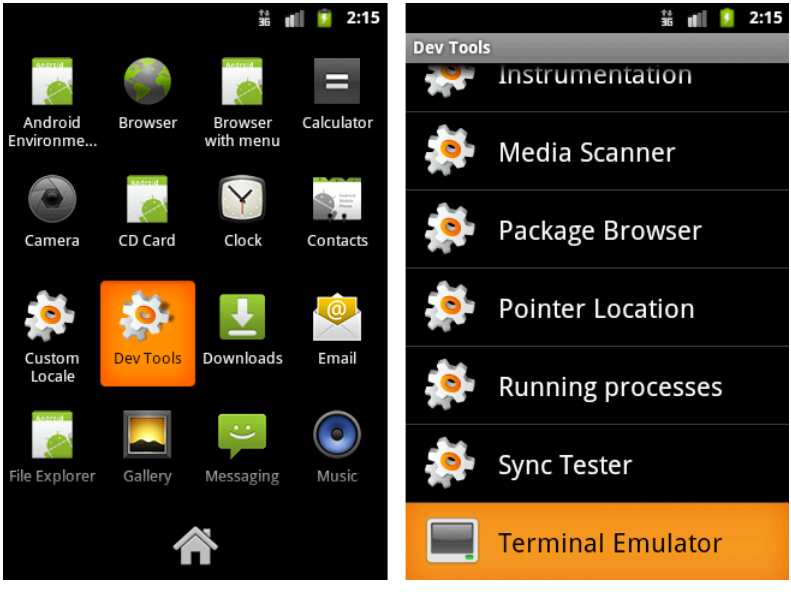


Рис. 17.9. Доступ к терминалу в системе Android

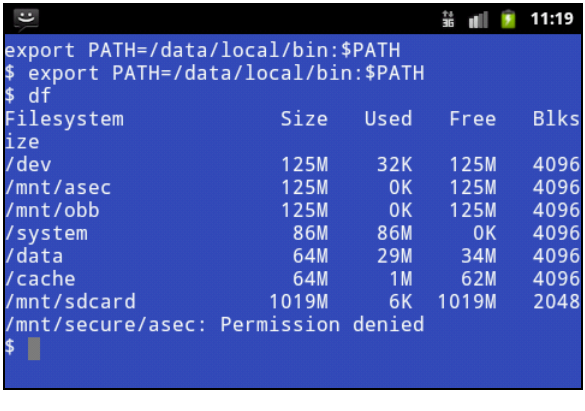


Рис. 17.10. Открытое окно терминала в Android

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 с выполненной командой `df` в Android представлен на рис. 17.10.

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.

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 на прилагаемом к книге компакт-диске в каталоге Ch17_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 17.13.

Листинг 17.13. 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` будет 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 17.14.

Листинг 17.14. 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 17.15.

Листинг 17.15. 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.ArrayAdapter;
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 17.16.

Листинг 17.16. 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. 17.11.

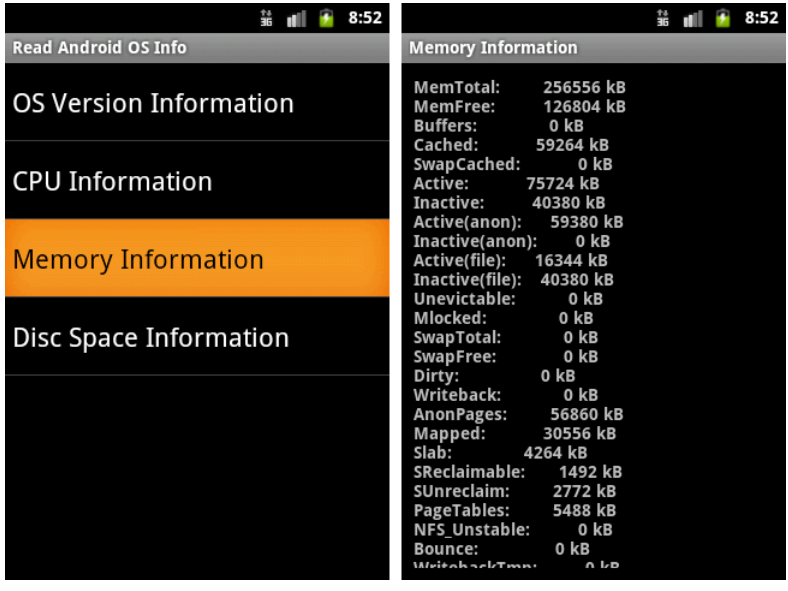


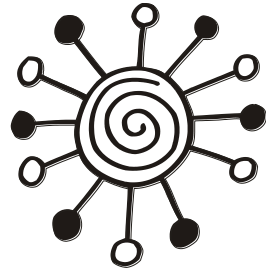
Рис. 17.11. Вывод информации о памяти мобильного устройства

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.

В следующей главе мы рассмотрим использование менеджера уведомлений для создания приложений, способных генерировать уведомления в строке состояния мобильного устройства для интерактивного взаимодействия с пользователем.



Глава 18

Управление пользовательскими уведомлениями

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 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 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`.

> ***Примечание***
>
> В версии Android 3.0 (API Level 11) для формирования уведомления используется вложенный класс `Notification.Builder`. Этот класс содержит большое количество методов для более гибкого создания структуры уведомления.

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

определено 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 виброзвонка 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, генерирующее 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 на прилагаемом к книге компакт-диске в каталоге Ch18_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 18.1.

Листинг 18.1. 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="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Create Notification"/>
```

```
    <Button
        android:id="@+id/bClear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Clear Notification"/>

</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 18.2.

Листинг 18.2. 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;
import android.widget.Button;

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);

        Button bGen = (Button)findViewById(R.id.bCreate);
        Button bClear = (Button)findViewById(R.id.bClear);

        bGen.setOnClickListener(this);
```

```
        bClear.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.bCreate:
            // получаем экземпляр менеджера уведомлений
            manager = (NotificationManager)getSystemService(
                    Context.NOTIFICATION_SERVICE);

            // создаем уведомление
            Notification notification = new Notification();

            // задаем параметры отображения
            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", отображаемых 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. 18.1.

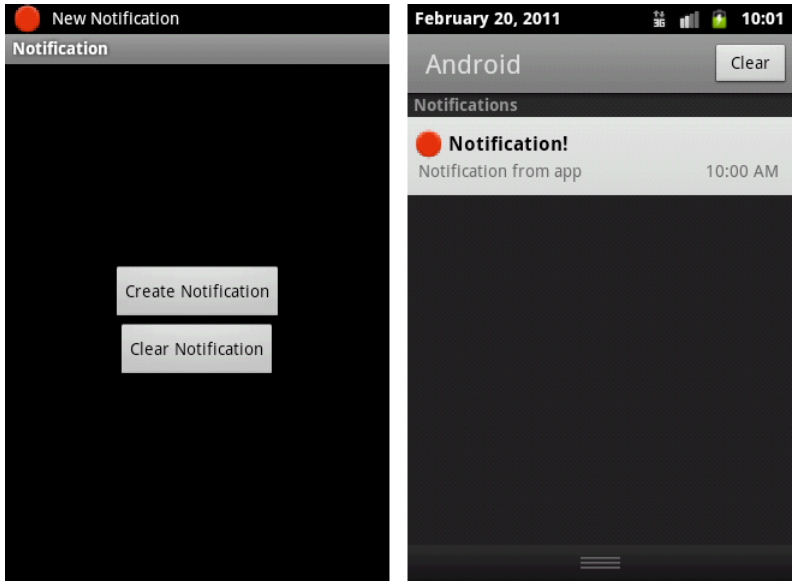


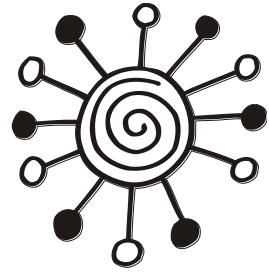
Рис. 18.1. Приложение, создающее уведомления в строке состояния

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.

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 службы Alarm Service для создания гибких приложений, которые способны создавать настраиваемые периодические оповещения для пользователя.



Глава 19

Создание пользовательских 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 отрабатывало 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 на прилагаемом к книге компакт-диске в каталоге Ch19_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 AndroidManifest.xml представлен в листинге 19.1.

Листинг 19.1. 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 19.2.

Листинг 19.2. 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"/>
    <Button
        android:id="@+id/bCancel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Cancel"/>
    <Button
        android:id="@+id/bStop"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Stop"/>
```

```
</LinearLayout>
```

В коде класса главного окна приложения `AlarmManagerActivity` реализуем принцип создания периодических оповещений, рассмотренный ранее в этом разделе. Код класса `AlarmManagerActivity` представлен в листинге 19.3.

Листинг 19.3. 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;
import android.widget.Button;

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);

        final Button bStart = (Button)findViewById(R.id.bStart);
        final Button bStop = (Button)findViewById(R.id.bStop);
        final Button bCancel = (Button)findViewById(R.id.bCancel);

        bStart.setOnClickListener(this);
        bStop.setOnClickListener(this);
        bCancel.setOnClickListener(this);

        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.bCancel:
            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 Notificaton). 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 19.4.

Листинг 19.4. 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. 19.1.

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.

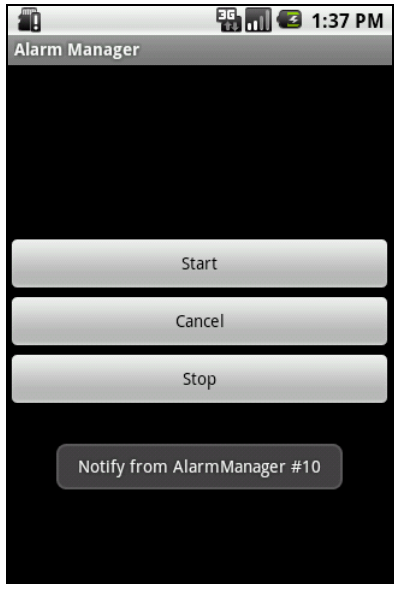


Рис. 19.1. Приложение для создания периодических оповещений

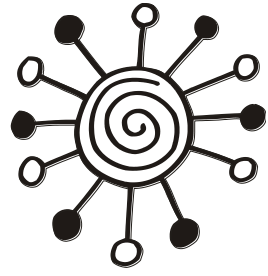
При нажатии кнопки **Stop** будет вызван метод `stopService()` и служба завершит работу и освободит ресурсы. Но поскольку в этом случае экземпляр `AlarmManager` не был закрыт методом `cancel()`, будет создан и запустится новый экземпляр службы, счетчик обнулится и начнет работу сначала. Для того чтобы 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, заключительной главе мы рассмотрим работу с менеджером буфера обмена и его возможности при работе с текстом, а также рассмотрим дополнительные функциональные возможности, предоставляемые платформой Android: библиотеки для синтеза речи на основе текста и для распознавания речи.



Глава 20

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.

> ***Примечание***
>
> В версии Android SDK 3.0 (API Level 11) определено два класса `ClipboardManager`, которые включены в разные пакеты. В этой главе мы будем рассматривать класс `ClipboardManager` из пакета `android.text` (для API Level 10). В 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 на прилагаемом к книге компакт-диске в каталоге Ch20_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 20.1.

Листинг 20.1. 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:text="Copy"
            android:layout_width="fill_parent"
            android:layout_weight="1"/>
        <Button
            android:id="@+id/bPaste"
            android:layout_height="wrap_content"
```

```
            android:layout_width="fill_parent"
            android:fadeScrollbars="false"
            android:layout_weight="1" android:text="Paste"/>
    </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 20.2.

Листинг 20.2. Файл класса главного окна приложения ClipboardManagerActivity.java

```
package com.samples.clipboardmanager;

import android.app.Activity;
import android.os.Bundle;
import android.text.ClipboardManager;
import android.view.View;
import android.widget.Button;
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);

        final Button bCopy = (Button)findViewById(R.id.bCopy);
```

```
        final Button bViewAllClips = (Button)findViewById(R.id.bPaste);

        bCopy.setOnClickListener(this);
        bViewAllClips.setOnClickListener(this);

        manager = (ClipboardManager)getSystemService(CLIPBOARD_SERVICE);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.bCopy:
            // копируем текст в буфер обмена
            manager.setText(textCopy.getText());
            break;

        case R.id.bPaste:
            // читаем текст из буфера обмена и
            // вставляем его в текстовое поле
            textPaste.append(manager.getText());
            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. 20.1.

Рис. 20.1. 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. 20.2), используя удаленный сетевой сервис.

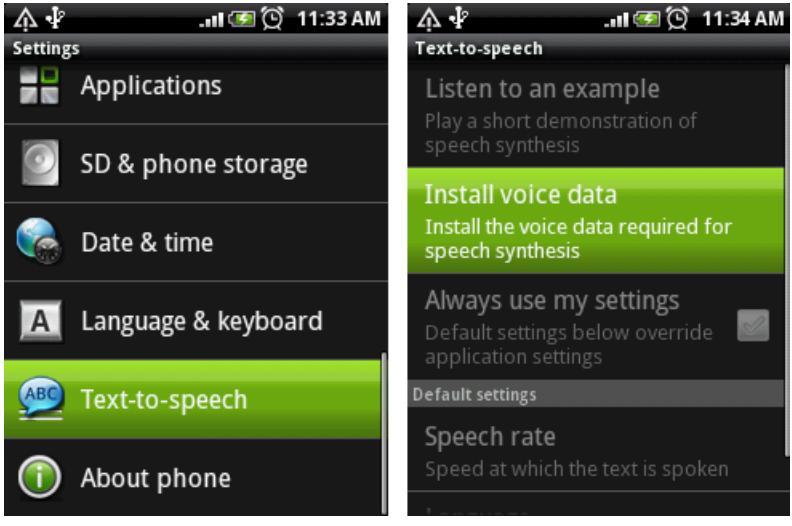


Рис. 20.2. Установка файлов языковых ресурсов

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 приложении 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 констуктор 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 на прилагаемом к книге компакт-диске в каталоге Ch20_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 20.3.

Листинг 20.3. 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: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 20.4.

Листинг 20.4. 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.speech.tts.TextToSpeech.OnInitListener;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class TtsActivity extends Activity
        implements TextToSpeech.OnInitListener {

    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.setOnClickListener(bSpeechClick);
        bSpeech.setEnabled(false);

        // создаем объект TextToSpeech
```

```
        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);
        }
    }

    public OnClickListener bSpeechClick = new OnClickListener() {
        @Override
        public void onClick(View arg0) {
            // 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. 20.3.

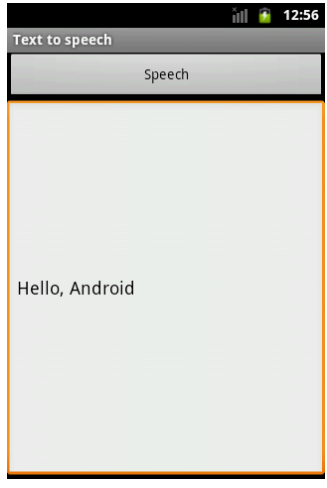


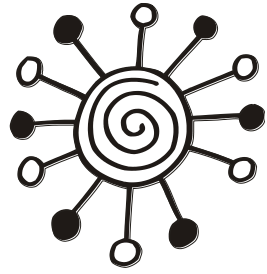
Рис. 20.3. Приложение с использованием TextToSpeech API

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!

Резюме

В этой заключительной 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.

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 изучение этой интересной и перспективной платформы.



Приложение

Установка примеров

По ссылке **ftp://85.249.45.166/9785977506663.zip**[1] можно скачать архив диска с исходными кодами примеров, приведенных в книге. Чтобы установить их на компьютер, создайте в IDE Eclipse новое рабочее пространство. Для этого выберите пункт 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. П1.1).

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 узел **Android** в левой панели. 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. П1.2.

Теперь, когда рабочее пространство для примеров создано и настроено, можно импортировать проекты с компакт-диска. Для этого выберите пункт меню **File | Import** и в открывшемся диалоговом окне **Import** выберите узел **General | Existing Projects into Workspace**, как показано на рис. П1.3. Нажмите кнопку **Next**.

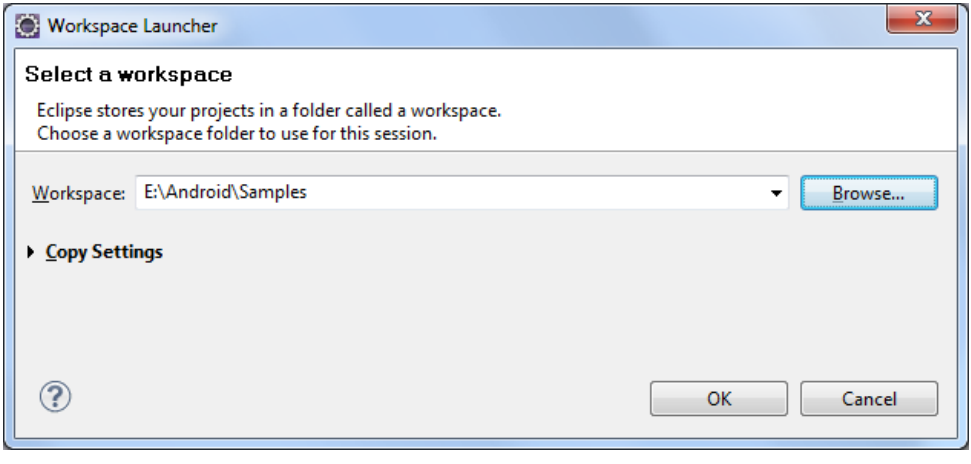


Рис. П1.1. Создание рабочего пространства в Eclipse

[1] Ссылка доступна также со страницы книги на сайте **www.bhv.ru**.

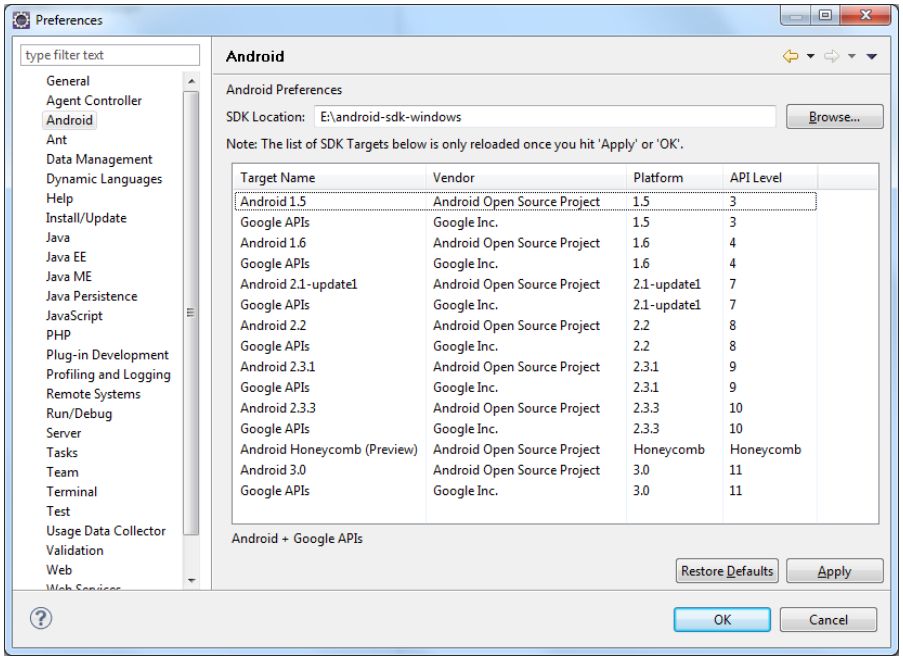


Рис. П1.2. Подключение Android SDK

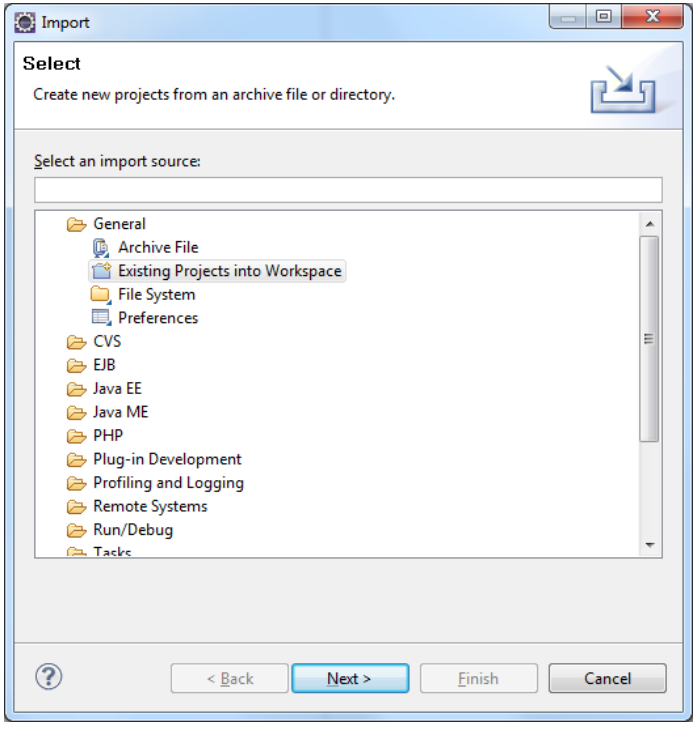


Рис. П1.3. Окно мастера импорта файлов

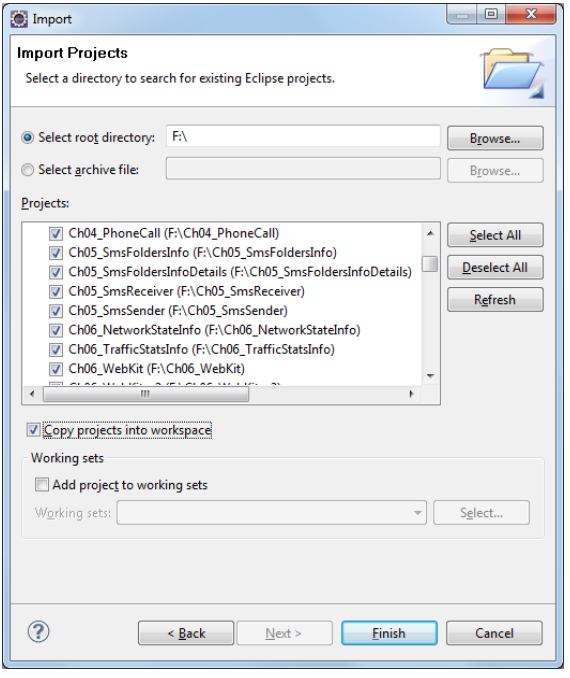


Рис. П1.4. Окно выбора импортируемого каталога и проектов

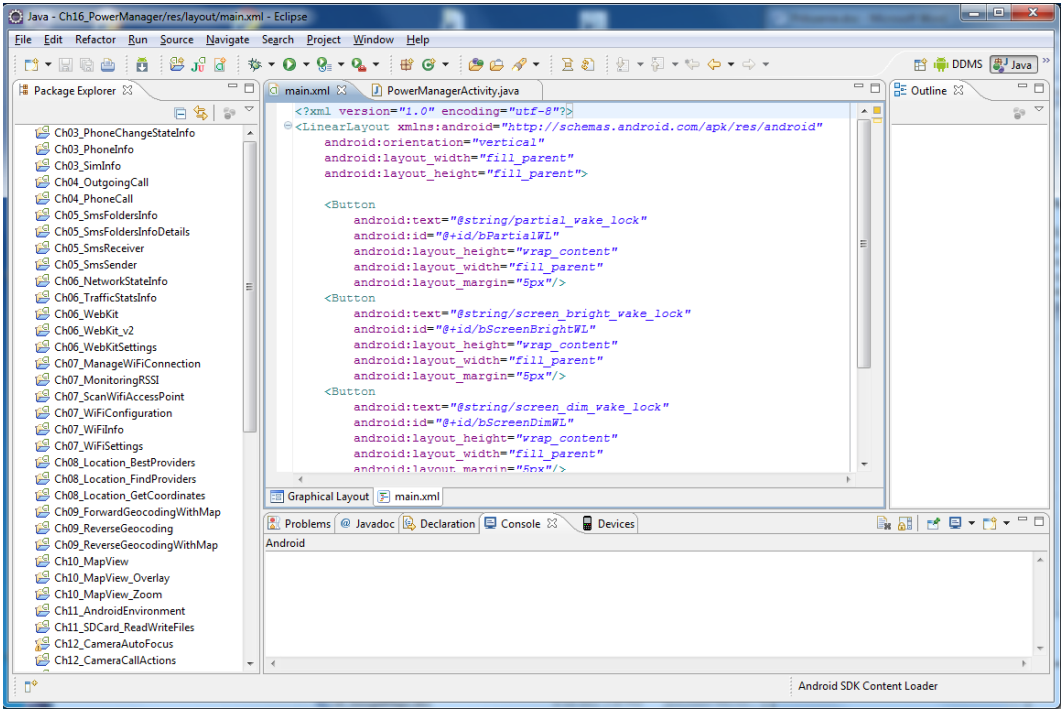


Рис. П1.5. Рабочее пространство с импортированными проектами

В следующем окне установите переключатель в положение **Select root directory** и укажите путь к каталогу Samples, находящемуся на компакт-диске. Список **Projects** отобразит все проекты, находящиеся на компакт-диске. Для импорта всех проектов из каталога нажмите кнопку **Select All**, а также установите флажок **Copy projects into workspace** для копирования файлов с компакт-диска в созданный вами каталог для рабочего пространства (рис. П1.4).

Все проекты должны быть импортированы с диска в ваш рабочий каталог без ошибок (рис. П1.5).

Кстати, обратите внимание на имена проектов в представлении **Package Explorer** на рис. П1.5: в отличие от примеров в книге, все проекты, импортированные с компакт-диска, содержат в названии префикс, указывающий на соответствующую главу, например Ch07_MonitoringRSSI, в то время как в тексте книги мы создавали проект с названием MonitoringRSSI. Это сделано для более удобной группировки проектов в рабочем пространстве Eclipse.

Литература и веб-ресурсы

- Голощапов А. Google Android: программирование для мобильных устройств. — СПб.: БХВ-Петербург, 2010. — 448с.: ил., ISBN 978-5-9775-0562-8.
- **http://developer.android.com** — официальный сайт Android.
- **http://android-developers.blogspot.com/?hl=en** — статьи и блоги.
- **http://www.androidjavadoc.com** — сайт-справочник по библиотекам Android.
- **http://grepcode.com/snapshot/repository.grepcode.com/java/ext/com.google.android/android/2.3.4_r1** — репозиторий исходного кода Android.
- **http://code.google.com/android/add-ons/google-apis/reference/index.html** — библиотеки Google API.

Предметный указатель

A

Accessibility Service 16
Account Service 17
Activity Action Intent 18
Activity Service 16
Alarm Service 16
Android Virtual Device 197
Audio Service 17

B

Battery Manager 17
Broadcast Intent 18, 282

C

Clipboard Service 16
Connectivity Service 17

D

DDMS (Dalvik Debug Monitor Server) 12, 46, 49, 198
Device_policy Service 18
Download Service 17
Dropbox Service 17

F

Forward Geocoding 158

G

Geocoding 158
GPS (Global Positioning System) 142

I, K

Input Method Service 16
Keyguard Service 16

L

Layout Inflater Service 16
Location Service 17

N

NFC Service 17
Notification Service 16

P

Power Service 17

R

Reverse Geocoding 158

S

Search Service 16
Sensor Service 17
Sensor Simulator 269
SMS (Short Message Service) 62
Storage Service 17

T

Telephony Service 17
Text To Speech 348

U, V

UI Mode Service 16
Vibrator Service 17

W

Wallpaper Service 16
Wifi Service 17
Window Service 16

Б

База данных:
  Browser 19
  CallLog 19
  ContactsContract 19
  Mediastore 19
  Settings 19
  SQLite 19
  UserDictionary 19
Базовая станция сотовой связи 27
Библиотека:
  Android SDK 9
  Google API 9, 141, 176
  Text To Speech 348
Браузер WebKit 85, 92

Д

Действие:
  ACTION_BATTERY_LOW 284
  ACTION_BATTERY_OKAY 284
  ACTION_CALL 53
  ACTION_CALL_BUTTON 53
  ACTION_DIAL 53
  ACTION_IMAGE_CAPTURE 221
  ACTION_PICK_WIFI_NETWORK 115
  ACTION_POWER_CONNECTED 284
  ACTION_POWER_
    DISCONNECTED 284
  ACTION_POWER_USAGE_
    SUMMARY 290
  ACTION_VIDEO_CAPTURE 221
  ACTION_WIFI_IP_SETTINGS 115
  ACTION_WIFI_SETTINGS 115
  ACTION_WIRELESS_SETTINGS 115
  INTENT_ACTION_STILL_IMAGE_
    CAMERA 221
  INTENT_ACTION_VIDEO_
    CAMERA 221
  NETWORK_STATE_CHANGED_
    ACTION 108
  SCAN_RESULTS_AVAILABLE_
    ACTION 128
  SUPPLICANT_CONNECTION_
    CHANGE_ACTION 109
  SUPPLICANT_STATE_CHANGED_
    ACTION 109
  WIFI_STATE_CHANGED_
    ACTION 108

И

Идентификатор:
  BSSID 109, 118
  SSID 118
Индекс ключа WEP 124
Интерфейс:
  Camera.PictureCallback 236
  Camera.ShutterCallback 236
  LocationListener 153
  OnInitListener 349
  SensorEventListener 251
  SurfaceHolder 224
  TextBasedSmsColumns 79
  WindowManager 272

К

Карта памяти 197
Каталог:
  Conversations 74
  Draft 74
  Failed 74
  Inbox 74
  Outbox 74
  Queued 74
  Sent 74
  Undelivered 74
Каталог для хранения ключей 174
Класс:
  ActivityManager 303
  AlarmManager 336
  AutoFocusCallback 241
  BatteryManager 282
  BroadcastReceiver 58, 128, 282
  Camera 213
  Camera.Parameters 214
  CdmaCellLocation 28
  CellLocation 27
  ClipboardManager 344
  ConfigurationInfo 309
  ConnectivityManager 85
  Criteria 143, 147
  Cursor 79
  Display 272
  DisplayMetrics 273
  Environment 199
  File 199
  GeoPoint 179
  GsmCellLocation 27

IntentFilter 128
LayoutInflater 232
Locale 159
Location 152
LocationManager 141
LocationProvider 143
MapActivity 178
MapController 178
MapView 177
MemoryInfo 313
NetworkInfo 86
Notification 330
Notification.Builder 330
NotificationManager 329
PhoneStateListener 37
PowerManager 294
Preference 97
Process 323
ProcessBuilder 323
ProcessErrorStateInfo 313, 320
RecentTaskInfo 313, 319
RunningAppProcessInfo 313
RunningServiceInfo 313, 316
RunningTaskInfo 313, 317
SeekBar 187
Sensor 248
SensorManager 249
Service 337
ServiceState 41
Settings 115
SmsManager 64
SmsMessage 70
SurfaceView 224
TelephonyManager 25
TextToSpeech 348
TrafficStats 89
WakeLock 294
WebView 93
WifiConfiguration 123
WifiInfo 118
WifiManager 107

Ключ Maps API Key 174
Команды Linux 323
Компонент:
Activity 15
Broadcast Receiver 15
Content Provider 15
Service 15

Константа:
ACCURACY_COARSE 147
ACCURACY_FINE 147
ACCURACY_HIGH 147
ACCURACY_LOW 147
ACCURACY_MEDIUM 147
ACQUIRE_CAUSES_WAKEUP 294
BATTERY_HEALTH_DEAD 283
BATTERY_HEALTH_GOOD 283
BATTERY_HEALTH_OVER_VOLTAGE 283
BATTERY_HEALTH_OVERHEAT 283
BATTERY_HEALTH_UNKNOWN 283
BATTERY_HEALTH_UNSPECIFIED_FAILURE 283
BATTERY_PLUGGED_AC 283
BATTERY_PLUGGED_USB 283
BATTERY_STATUS_CHARGING 283
BATTERY_STATUS_DISCHARGING 283
BATTERY_STATUS_FULL 283
BATTERY_STATUS_NOT_CHARGING 283
BATTERY_STATUS_UNKNOWN 283
CALL_STATE_IDLE 28
CALL_STATE_OFFHOOK 28
CALL_STATE_RINGING 28
CLIPBOARD_SERVICE 344
CONNECTIVITY_SERVICE 85
DEFAULT_ALL 330
DEFAULT_LIGHTS 330
DEFAULT_SOUND 330
DEFAULT_VIBRATE 330
DENSITY_HIGH 273
DENSITY_LOW 273
DENSITY_MEDIUM 273
DENSITY_XHIGH 273
DIRECTORY_ALARMS 200
DIRECTORY_DCIM 200
DIRECTORY_DOWNLOADS 200
DIRECTORY_MOVIES 200
DIRECTORY_MUSIC 200
DIRECTORY_NOTIFICATIONS 200
DIRECTORY_PICTURES 200
DIRECTORY_PODCASTS 200
DIRECTORY_RINGTONES 200
ELAPSED_REALTIME 337
(продолжение рубрики см. на стр. 364)

Константа *(продолжение)*:
ELAPSED_REALTIME_WAKEUP 337
ERROR 349
FULL_WAKE_LOCK 294
GPS_PROVIDER 143
GRAVITY_DEATH_STAR_I 263
GRAVITY_MERCURY 263
GRAVITY_MOON 263
GRAVITY_VENUS 263
INTERVAL_DAY 337
INTERVAL_FIFTEEN_MINUTES 337
INTERVAL_HALF_DAY 337
INTERVAL_HALF_HOUR 337
INTERVAL_HOUR 337
KEYBOARD_12KEY 310
KEYBOARD_NOKEYS 310
KEYBOARD_QWERTY 310
KEYBOARD_UNDEFINED 310
LIGHT_CLOUDY 253
LIGHT_FULLMOON 253
LIGHT_NO_MOON 253
LIGHT_OVERCAST 253
LIGHT_SHADE 253
LIGHT_SUNLIGHT 253
LIGHT_SUNLIGHT_MAX 253
LIGHT_SUNRISE 253
LISTEN_CALL_FORWARDING_INDICATOR 39
LISTEN_CALL_STATE 39
LISTEN_CELL_LOCATION 39
LISTEN_DATA_ACTIVITY 39
LISTEN_DATA_CONNECTION_STATE 39
LISTEN_MESSAGE_WAITING_INDICATOR 39
LISTEN_NONE 39
LISTEN_SERVICE_STATE 39
LISTEN_SIGNAL_STRENGTHS 39
LOCATION_SERVICE 141
MAGNETIC_FIELD_EARTH_MAX 267
MAGNETIC_FIELD_EARTH_MIN 267
MEDIA_BAD_REMOVAL 202
MEDIA_CHECKING 202
MEDIA_MOUNTED 202
MEDIA_MOUNTED_READ_ONLY 202
MEDIA_NOFS 202
MEDIA_REMOVED 202
MEDIA_SHARED 202
MEDIA_UNMOUNTABLE 202
MEDIA_UNMOUNTED 202
NAVIGATION_DPAD 310
NAVIGATION_NONAV 310
NAVIGATION_TRACKBALL 310
NAVIGATION_UNDEFINED 310
NAVIGATION_WHEEL 310
NETWORK_PROVIDER 143
NETWORK_TYPE_1xRTT 26
NETWORK_TYPE_CDMA 26
NETWORK_TYPE_EDGE 26
NETWORK_TYPE_EHRPD 26
NETWORK_TYPE_EVDO_0 26
NETWORK_TYPE_EVDO_A 26
NETWORK_TYPE_EVDO_B 26
NETWORK_TYPE_GPRS 26
NETWORK_TYPE_HSDPA 27
NETWORK_TYPE_HSPA 27
NETWORK_TYPE_HSUPA 27
NETWORK_TYPE_IDEN 27
NETWORK_TYPE_LTE 26
NETWORK_TYPE_UMTS 27
NETWORK_TYPE_UNKNOWN 27
NO_REQUIREMENT 147
ON_AFTER_RELEASE 294
PARTIAL_WAKE_LOCK 294
PASSIVE_PROVIDER 144
PHONE_TYPE_CDMA 26
PHONE_TYPE_GSM 26
PHONE_TYPE_NONE 26
PHONE_TYPE_SIP 26
POWER_HIGH 147
POWER_LOW 147
POWER_MEDIUM 147
QUEUE_ADD 349
QUEUE_FLUSH 349
RECENT_IGNORE_UNAVAILABLE 319
RECENT_WITH_EXCLUDED 319
RESULT_ERROR_GENERIC_FAILURE 64
RESULT_ERROR_NO_SERVICE 64
RESULT_ERROR_NULL_PDU 64
RESULT_ERROR_RADIO_OFF 64
RESULT_OK 64
ROTATION_0 273

ROTATION_180 273
ROTATION_270 273
ROTATION_90 273
RTC 337
RTC_WAKEUP 337
SCREEN_BRIGHT_WAKE_LOCK 294
SCREEN_DIM_WAKE_LOCK 294
SENSOR_DELAY_FASTEST 252
SENSOR_DELAY_GAME 252
SENSOR_DELAY_NORMAL 252
SENSOR_DELAY_UI 252
SENSOR_STATUS_ACCURACY_HIGH 251
SENSOR_STATUS_ACCURACY_LOW 251
SENSOR_STATUS_ACCURACY_MEDIUM 251
SENSOR_STATUS_UNRELIABLE 251
SIM_STATE_ABSENT 35
SIM_STATE_NETWORK_LOCKED 35
SIM_STATE_PIN_REQUIRED 35
SIM_STATE_PUK_REQUIRED 35
SIM_STATE_READY 35
SIM_STATE_UNKNOWN 35
STANDARD_GRAVITY 263
STATE_EMERGENCY_ONLY 41
STATE_IN_SERVICE 41
STATE_OUT_OF_SERVICE 41
STATE_POWER_OFF 41
STATUS_ON_ICC_FREE 71
STATUS_ON_ICC_READ 71
STATUS_ON_ICC_SEND 71
STATUS_ON_ICC_UNREAD 71
STATUS_ON_ICC_UNSENT 71
SUCCESS 349
TOUCHSCREEN_FINGER 311
TOUCHSCREEN_NOTOUCH 311
TOUCHSCREEN_STYLUS 311
TOUCHSCREEN_UNDEFINED 311
TYPE_ACCELEROMETER 248
TYPE_ALL 248
TYPE_GRAVITY 248
TYPE_GYROSCOPE 248
TYPE_LIGHT 248
TYPE_LINEAR_ACCELERATION 248
TYPE_MAGNETIC_FIELD 248
TYPE_ORIENTATION 248
TYPE_PRESSURE 248
TYPE_PROXIMITY 248
TYPE_ROTATION_VECTOR 248
TYPE_TEMPERATURE 248
WIFI_STATE_DISABLED 108
WIFI_STATE_DISABLING 108
WIFI_STATE_ENABLED 108
WIFI_STATE_ENABLING 108
WIFI_STATE_UNKNOWN 108
WINDOW_SERVICE 272
Контент-провайдер 19

M

Метод:
acquire(), класс PowerManager 294
allowedAuthAlgorithms() 124
allowedGroupCiphers() 124
allowedKeyManagement() 124
allowedPairwiseCiphers() 124
allowedProtocols() 124
animateTo() 178
calculateSignalLevel() 134
cancel(), класс AlarmManager 336
cancel(), класс NotificationManager 329
cancelAll(), класс NotificationManager 329
compareSignalLevel() 134
displayZoomControls() 185
getAccuracy() 143, 147, 153
getActiveNetworkInfo() 85
getAddressLine() 159
getAllNetworkInfo() 85
getAllProviders() 143
getAltitude() 153
getBaseStationId() 28
getBaseStationLatitude() 28
getBaseStationLongitude() 28
getBearingAccuracy() 147
getBestProvider() 147
getBSSID() 118
getCallState() 28
getCdmaDbm() 40
getCdmaEcio() 40
getCellLocation() 27
getCid() 27
getColumnName() 79
getCountry() 159
getCountryCode() 158
(продолжение рубрики см. на стр. 366)

Метод *(продолжение)*:
getCountryName() 158
getDataDirectory() 199
getDefault() 64
getDefaultDisplay() 272
getDefaultSensor() 249
getDeviceConfigurationInfo() 309
getDhcpInfo() 118
getDisplayCountry() 159
getDisplayId() 272
getDisplayLanguage() 159
getDisplayMessageBody() 70
getDisplayName() 159
getDisplayOriginatingAddress() 70
getDownloadCacheDirectory() 199
getEvdoDbm() 40
getEvdoEcio() 40
getEvdoSnr() 40
getExternalStorageDirectory() 199
getExternalStorageState() 202
getFeatureName() 159
getFromLocationName() 168
getGsmBitErrorRate() 40
getGsmSignalStrength() 40
getHeight() 273
getHolder() 224
getHorizontalAccuracy() 147
getIndexOnIcc() 70
getIsManualSelection() 41
getISO3Country() 159
getISO3Language() 159
getLac() 27
getLanguage() 159, 349
getLastKnownLocation() 152
getLatitude() 152
getLatitudeE6() 179
getLinkSpeed() 118
getLocality() 159
getLongitude() 152
getLongitudeE6() 179
getMaximumRange() 248
getMaxZoomLevel() 177
getMemoryInfo() 313
getMessageBody() 70
getMessageClass() 70
getMinDelay() 248
getMobileRxBytes() 89
getMobileRxPackets() 89
getMobileTxBytes() 89
getMobileTxPackets() 89
getName() 143, 248
getNetworkId() 28
getNetworkInfo() 85
getNetworkType() 26
getOperatorAlphaLong() 41
getOperatorAlphaShort() 41
getOperatorNumeric() 41
getOriginatingAddress() 70
getParameters(), класс Camera 214
getPdu() 70
getPhone() 159
getPhoneType() 26
getPostalCode() 159
getPower() 248
getPowerRequirement() 143, 147
getProcessesInErrorState() 320
getProcessMemoryInfo() 313
getProtocolIdentifier() 70
getProvider() 143
getProviders() 143
getPsc() 27
getRefreshRate() 273
getResolution() 248
getRoaming() 41
getRootDirectory() 199
getRssi() 133
getRunningAppProcesses() 315
getRunningServices() 316
getRunningTasks() 317
getSensorList() 249
getServiceCenterAddress() 70
getSimCountryIso() 34
getSimOperator() 34
getSimOperatorName() 34
getSimSerialNumber() 34
getSimState() 34, 35
getSpeedAccuracy() 147
getSSID() 118
getState() 41
getStatus() 70
getStatusOnIcc() 71
getSubAdminArea() 158
getSubtype() 86
getSubtypeName() 86
getSupportedColorEffects() 217
getSupportedFlashModes() 217
getSupportedFocusModes() 217
getSupportedJpegThumbnailSizes() 217

getSupportedPictureFormats() 217
getSupportedPictureSizes() 217
getSupportedPreviewFormats() 217
getSupportedPreviewFpsRange() 217
getSupportedPreviewSizes() 217
getSupportedSceneModes() 217
getSupportedVideoSizes() 217
getSupportedWhiteBalance() 217
getSystemId() 28
getSystemService() 25
getText(), класс ClipboardManager 344
getTimestampMillis() 70
getTotalRxBytes() 89
getTotalTxBytes() 89
getTotalTxPackets() 89
getType() 86
getTypeName() 86
getUidRxBytes() 90
getUidTxBytes() 90
getUserData() 71
getVendor() 248
getVersion() 248
getVerticalAccuracy() 147
getWidth() 273
getZoomLevel() 177
goToSleep() 293
hasMonetaryCost() 143
hasText(), класс ClipboardManager 344
isAltitudeRequired() 148
isAvailable() 86
isBearingRequired() 148
isConnected() 86
isConnectedOrConnecting() 86
isCostAllowed() 148
isDirectory() 199
isFile() 199
isLanguageAvailable() 349
isNetworkRoaming() 28
isRoaming() 86
isSatellite() 177
isScreenOn() 293
isSpeaking() 349
isSpeedRequired() 148
isStreetView() 177
isTraffic() 177
loadData() 95
loadDataWithBaseURL() 95
loadUrl() 93
meetsCriteria() 143
newWakeLock() 294
notify(), класс NotificationManager 329
onCallForwardingIndicatorChanged() 38
onCallStateChanged() 38
onCellLocationChanged() 38, 40
onDataActivity() 38
onDataConnectionStateChanged() 38
onListItemClick() 80
onLocationChanged() 153
onMessageWaitingIndicatorChanged() 38
onProviderDisabled() 153
onProviderEnabled() 153
onSensorChanged() 251
onServiceStateChanged() 38, 41
onSignalStrengthsChanged() 38, 39
onStatusChanged() 153
open(), класс Camera 213
preSharedKey() 124
reboot() 293
registerListener() 252
release(), класс Camera 213
release(), класс PowerManager 295
requestLocationUpdates() 154
requestSingleUpdate() 154
requiresCell() 143
requiresNetwork() 143
requiresSatellite() 143
sendMultipartTextMessage() 65
set(), класс AlarmManager 336
setAccuracy() 147
setBuiltInZoomControls() 97, 185
setCenter() 178
setCursiveFontFamily() 97
setDefaultFixedFontSize() 97
setDefaultFontSize() 97
setDefaultZoom() 97
setDisplayZoomControls() 97
setFixedFontFamily() 97
setInexactRepeating() 336
setLanguage() 349
setMinimumFontSize() 97
setMinimumLogicalFontSize() 97
setParameters(), класс Camera 214
setPowerRequirement() 147
setReferenceCounted(), класс PowerManager 295
setRepeating() 336

(окончание рубрики см. на стр. 366)

Метод *(окончание)*:
setSatellite() 177
setStreetView() 177
setSupportZoom() 97
setText(), класс ClipboardManager 344
setTextSize() 97
setTime(), класс AlarmManager 336
setTimeZone(), класс AlarmManager 336
setTraffic() 177
setZoom() 178
shutdown(), класс TextToSpeech 348
speak() 348
startActivity() 163
startScan() 128
stop(), класс TextToSpeech 348
stopAnimation() 178
supportsAltitude() 143
supportsSpeed() 143
surfaceChanged() 224
surfaceCreated() 224
surfaceDestroyed() 224
synthesizeToFile() 348
takePicture() 236
wepKeys() 124
wepTxKeyIndex() 124
zoomIn() 178
zoomInFixing() 178
zoomOutFixing() 178
zoomToSpan() 179
Метрики дисплея 273

О

Объект
Intent 18, 53
Окно:
Android Device Chooser 13
Android SDK and AVD Manager 9

П

Пакет:
android.app 303
android.location 141, 153
android.net 85
android.net.http 85
android.net.wifi 107
android.os 199
android.provider 19
android.telephony 25
android.view 272
java.io.File 199
java.net 85
Параметр:
Azimuth 258
EXTRA_BSSID 109
EXTRA_HEALTH 283
EXTRA_ICON_SMALL 283
EXTRA_LEVEL 282
EXTRA_NETWORK_INFO 108
EXTRA_NEW_STATE 109
EXTRA_PRESENT 282
EXTRA_PREVIOUS_WIFI_STATE 108
EXTRA_SCALE 282
EXTRA_SUPPLICANT_CONNECTED 109
EXTRA_SUPPLICANT_ERROR 109
EXTRA_TECHNOLOGY 283
EXTRA_TEMPERATURE 282
EXTRA_VOLTAGE 282
Lateral 263
Longitudinal 263
Pitch 258
Roll 258
Vertical 263
Плотность пиксел 273
Поле:
_id, интерфейс TextBasedSmsColumns 79
address, интерфейс TextBasedSmsColumns 79
availMem 313
baseActivity 317
baseIntent 319
body, интерфейс TextBasedSmsColumns 79
condition, класс ProcessErrorStateInfo 320
date, интерфейс TextBasedSmsColumns 79
densityDpi 273
description, класс RecentTaskInfo 319
description, класс RunningTaskInfo 317
dns1 118
dns2 118
error_code, интерфейс TextBasedSmsColumns 79
flags, класс Notification 331

frequency 128
gateway 118
heightPixels 273
id, класс RecentTaskInfo 319
id, класс RunningTaskInfo 317
interval 337
ipAddress 118
level 128
locked, интерфейс
TextBasedSmsColumns 79
longMsg,
класс ProcessErrorStateInfo 321
lowMemory 313
netmask 118
numActivities 317
numRunning 317
operation 337
origActivity 319
person, интерфейс
TextBasedSmsColumns 79
pid, класс ProcessErrorStateInfo 320
priority, класс WifiConfiguration 124
processName, класс
ProcessErrorStateInfo 320
protocol, интерфейс
TextBasedSmsColumns 79
read,
интерфейс TextBasedSmsColumns 79
reply_path_present 79
reqGlEsVersion 309
reqInputFeatures 309
reqKeyboardType 310
reqNavigation 310
reqTouchScreen 310
scaledDensity 273
serverAddress 118
service_center, интерфейс
TextBasedSmsColumns 79
shortMsg,
класс ProcessErrorStateInfo 320
stackTrace 321
status, интерфейс
TextBasedSmsColumns 79
status, класс WifiConfiguration 124
subject 79
tag, класс ProcessErrorStateInfo 320
thread_id, интерфейс
TextBasedSmsColumns 79
threshold 313
topActivity 317
triggerAtTime 337
type, интерфейс
TextBasedSmsColumns 79
uid, класс ProcessErrorStateInfo 320
widthPixels 273
xdpi 273
ydpi 273
capabilities 128
Протокол:
аутентификации 124
шифрования 124

P

Разрешение:
ACCESS_COARSE_
LOCATION 29, 40, 144
ACCESS_FINE_LOCATION 144
ACCESS_WIFI_STATE 108
BATTERY_STATS 284
CALL_PHONE 52
CALL_PRIVILEGED 52
CAMERA 214
CHANGE_WIFI_STATE 108
GET_TASKS 318
INTERNET 93
MODIFY_PHONE_STATE 52
PROCESS_OUTGOING_CALLS 52, 59
READ_PHONE_STATE 29, 35, 52
RECEIVE_SMS 65
SEND_SMS 65
WAKE_LOCK 295
WRITE_SETTINGS 277
WRITE_SMS 65
Разрешения 20
Режим отладки через USB 11
Роуминг 28

C

Сервис:
Geocoding 158
Google Maps 174
Система глобального
позиционирования 142
Сканирование точек доступа 128
Состояние соединения 108
Среда разработки Eclipse 9

Т

Терминал 322
Технология
  1xRTT 26
  CDMA 26
  EDGE 26
  EHRPD 26
  EVDO_0 26
  GPRS 26
  GSM 26
  HSDPA 27
  HSPA 27
  HSUPA 27
  IDEN 27
  LTE 26
  SIP 26
  UMTS 27
Тип экрана 310
Точка доступа 107

У, Ф

Уровень принимаемого
  сигнала 39
Файл манифеста
  приложения 29
Флаг:
  DEFAULT_LIGHTS 330
  DEFAULT_SOUND 330
  DEFAULT_VIBRATE 330
  FLAG_AUTO_CANCEL 331
  FLAG_FOREGROUND_SERVICE 331
  FLAG_HIGH_PRIORITY 331
  FLAG_INSISTENT 331
  FLAG_NO_CLEAR 331
  FLAG_ONGOING_EVENT 331
  FLAG_ONLY_ALERT_ONCE 331
  FLAG_SHOW_LIGHTS 331
  INPUT_FEATURE_FIVE_WAY_
    NAV 310
  INPUT_FEATURE_HARD_
    KEYBOARD 310

Ш, Э

Шифрование AES 124
Экран:
  емкостной 310
  резистивный 310
  сенсорный 311
Эмулятор Android 10, 13, 33, 49