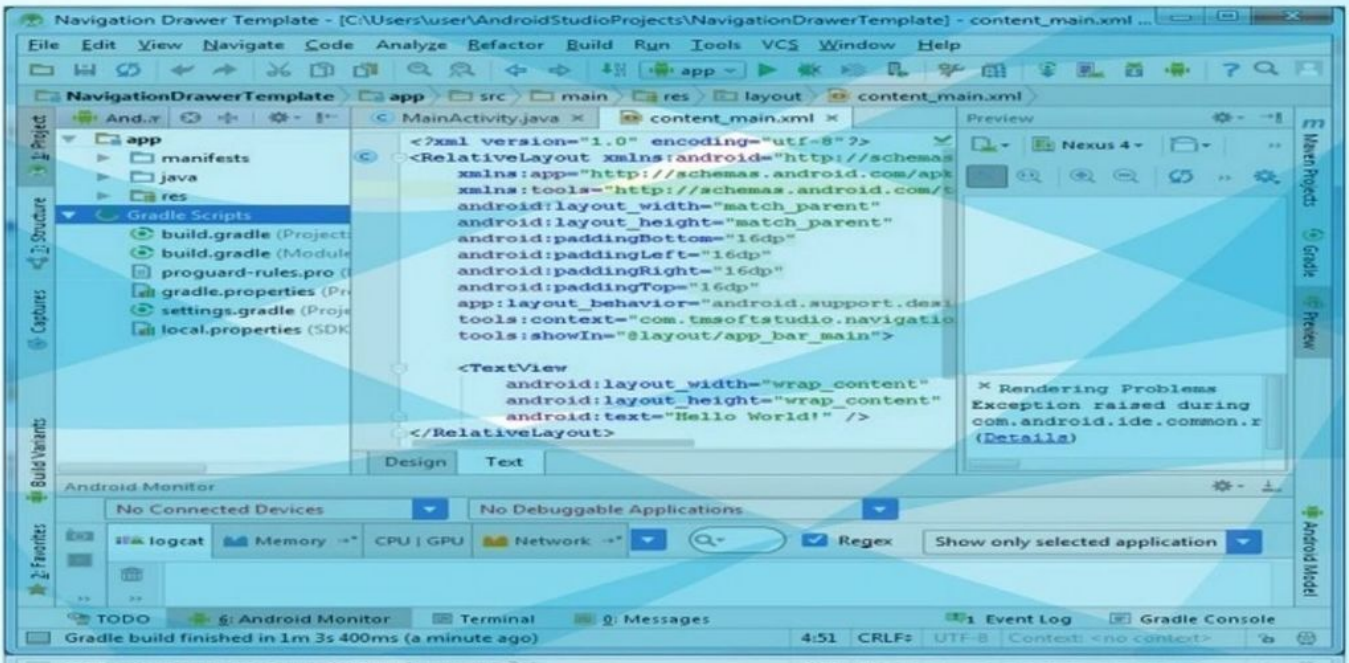
# Разработка Android приложений в деталях




Тимур Машнин

Тимур Машнин

# Разработка Android-приложений в деталях

«Издательские решения»

**Машнин Т. С.**

Разработка Android-приложений в деталях / Т. С. Машнин — «Издательские решения»,

ISBN 978-5-44-830450-7

В книге приведены некоторые рецепты разработки Android-приложений и их примеры, рассмотрена работа в среде Eclipse и Android Studio, разработка мобильных сайтов и гибридных мобильных приложений.

ISBN 978-5-44-830450-7

# Содержание

# Разработка Android-приложений в деталях

## Тимур Сергеевич Машнин

*Дизайнер обложки* Тимур Сергеевич Машнин



ISBN 978-5-4483-0450-7
Создано в интеллектуальной издательской системе Ridero

# Введение

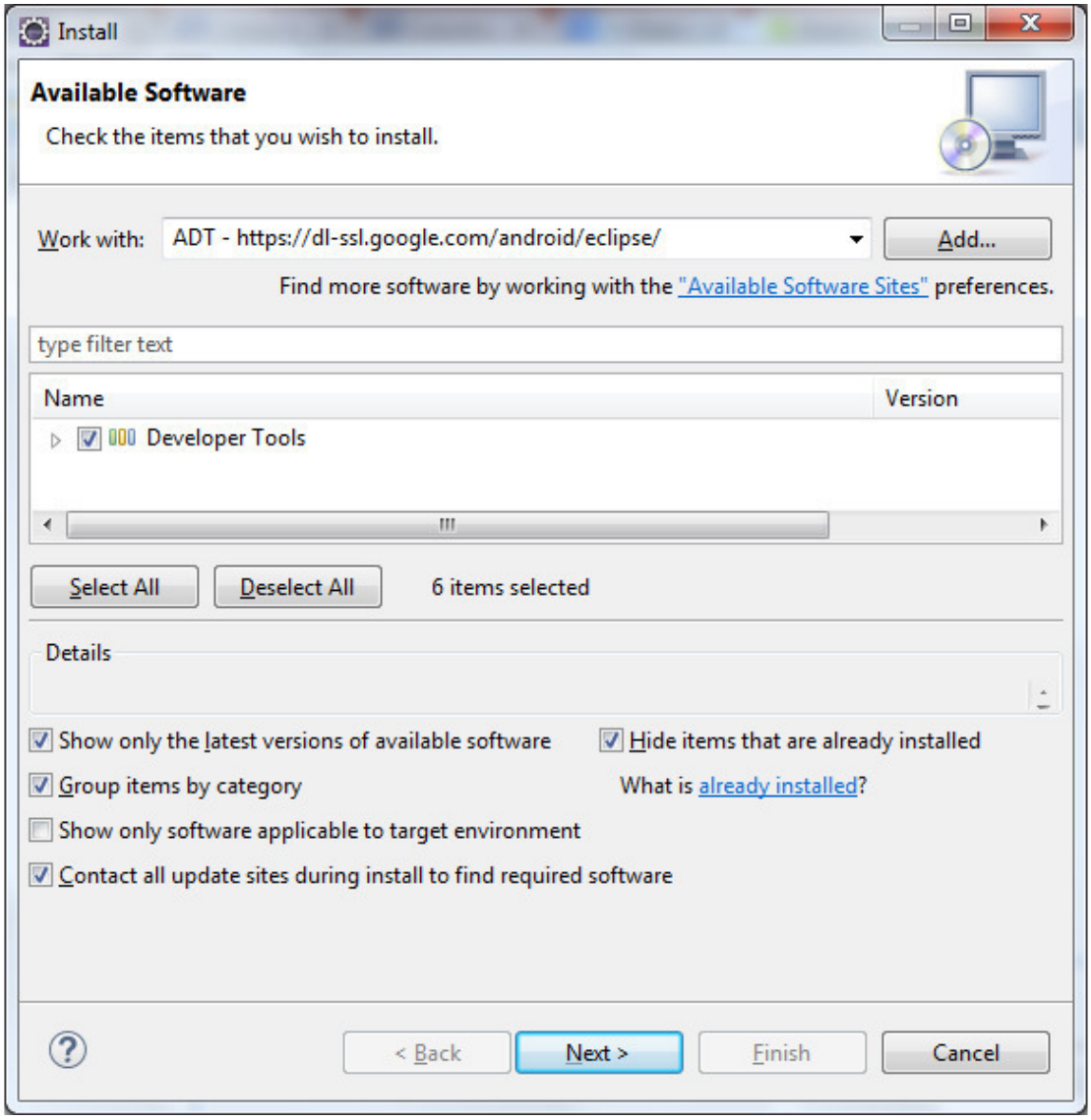
Разработку Android-приложений можно вести как в среде Eclipse, так и в среде Android Studio. Хотя среда Android Studio является официально поддерживаемой средой разработки, среда Eclipse не теряет актуальности из-за своей универсальности в разработке Java-приложений самого широкого спектра применений.

Поддержку разработки Android-приложений в среде Eclipse обеспечивает Eclipse-плагин Android Development Tools (ADT) (http://developer.android.com/sdk/eclipse-adt.html).

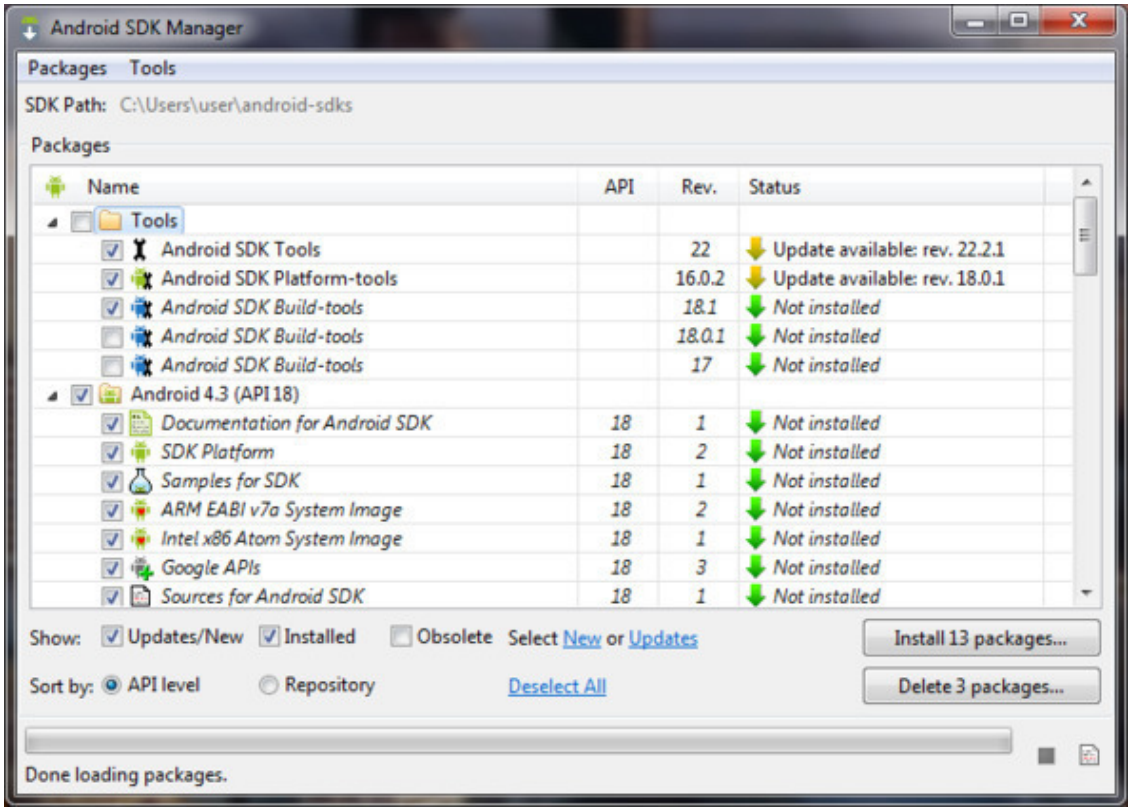
ADT-плагин помогает создать Android-проект, разработать UI-интерфейс приложения на основе программного интерфейса Android Framework API, отладить Android-приложение и подготовить подписанный файл. apk к публикации.

# Установка ADT плагина

Для установки ADT-плагина откроем среду Eclipse IDE for Java EE Developers и в меню **Help** выберем команду **Install New Software**. В списке **Work with:** нажмем кнопку **Add**, в поле **Name:** введем имя плагина ADT, а в поле **Location:** – адрес https://dl-ssl.google.com/android/eclipse/ хранилища плагина, нажмем кнопку **OK**, в мастере **Install** отметим флажок **Developer Tools** и нажмем кнопку **Next**.




После установки ADT-плагина и перезапуска среды Eclipse может открыться окно приложения Android SDK Manager набора разработчика Android SDK.




Сам по себе дистрибутив набора Android SDK (http://developer.android.com/sdk/index.html) включает в себя набор инструментов SDK Tools, SDK Platform-tools, а также приложения AVD Manager и SDK Manager.

Приложение SDK Manager дает возможность устанавливать и обновлять компоненты набора Android SDK, а также запускать приложение AVD Manager и управлять URL-адресами дополнений.

Приложение SDK Manager можно также запустить из среды Eclipse с помощью команды **Android SDK Manager** меню **Window** перспективы **Java**.

Набор инструментов SDK Tools обеспечивает отладку и тестирование Android-приложений. Набор инструментов SDK Platform-tools обеспечивает поддержку самой последней версии Android-платформы и включает в себя инструмент Android Debug Bridge (adb), позволяющий взаимодействовать с эмулятором или Android-устройством. Приложение AVD Manager предоставляет GUI-интерфейс для моделирования различных конфигураций Android-устройств, используемых Android-эмулятором запуска приложений в среде выполнения Android. Набор Android Build Tools обеспечивает сборку кода Android-приложения.

Для разработки Android-приложений требуется установка конкретной Android-платформы, включающей в себя библиотеки платформы, системные изображения, образцы кода, оболочки эмуляции.

Поэтому, используя приложение SDK Manager, установим последнюю возможную версию Android-платформы и наиболее распространенную или минимальную версию Android-платформы. Обратная совместимость между максимальной и минимальной версиями осуществляется с помощью библиотеки Android Support Library каталога Extras.

Дополнительно можно загрузить другие версии Android-платформы, документацию, примеры и различные дополнения набора Android SDK.

Различные версии API Android-платформы отличаются друг от друга наличием новых пакетов, а также изменениями в существующих пакетах.

Помимо изменений программного интерфейса API, от версии к версии Android-платформы изменялись предустановленные приложения, добавлялась поддержка новых технологий и улучшалась производительность.

# Описание ADT-плагина

В результате установки ADT-плагина в команде **New** меню **File** среды Eclipse появится раздел **Android**, содержащий следующие мастера:

Android Activity – создает класс, расширяющий класс android.app.Activity и представляющий экран приложения.

Android Application Project – обеспечивает создание проекта Android-приложения.

Android Icon Set – позволяет создать набор значков приложения:

Launcher Icons – значок, представляющий приложение.

Action Bar and Tab Icons (Android 3.0+) – значки элементов панели действий пользователя для платформы версии 3.0 и выше.

Notification Icons – значки уведомлений панели состояния.

Pre-Android 3.0 Tab Icons – значки элементов панели действий пользователя для платформы версии ниже 3.0.

Pre-Android 3.0 Menu Icons – значки меню для платформы версии ниже 3.0.

Android Object – создает различные компоненты, такие как Activity, Widget, Fragment, Receiver, Provider, Service и др.

Android Project from Existing Code – импорт проекта приложения.

Android Sample Project – при условии установки с помощью SDK Manager пакета примеров Samples for SDK, позволяет создать проект выбранного примера Android-приложения.

Android Test Project – для выбранного Android-проекта создает основу набора тестов на базе каркаса Android testing framework, являющегося расширением платформы тестирования JUnit.

Android XML File – обеспечивает создание таких ресурсов приложения как:

Layout – XML-описание GUI-интерфейса Activity-компонента.

Values – XML-файл, содержащий набор текстовых строк, стилей, различного рода значений, используемых приложением.

Drawable – XML-файл, формирующий отображаемую на экране графику.

Menu – XML-файл, определяющий меню приложения.

Color List – XML-файл, определяющий набор цветов для различных состояний GUI-компонента.

Property Animation – XML-файл, задающий анимацию свойств объекта.

Tween Animation – XML-файл, задающий анимацию View-компонента (вращение, исчезновение, перемещение и масштабирование).

AppWidgetProvider – XML-файл, содержащий метаданные для миниатюрного приложения App Widget, как правило размещаемого на главном экране Home Screen.

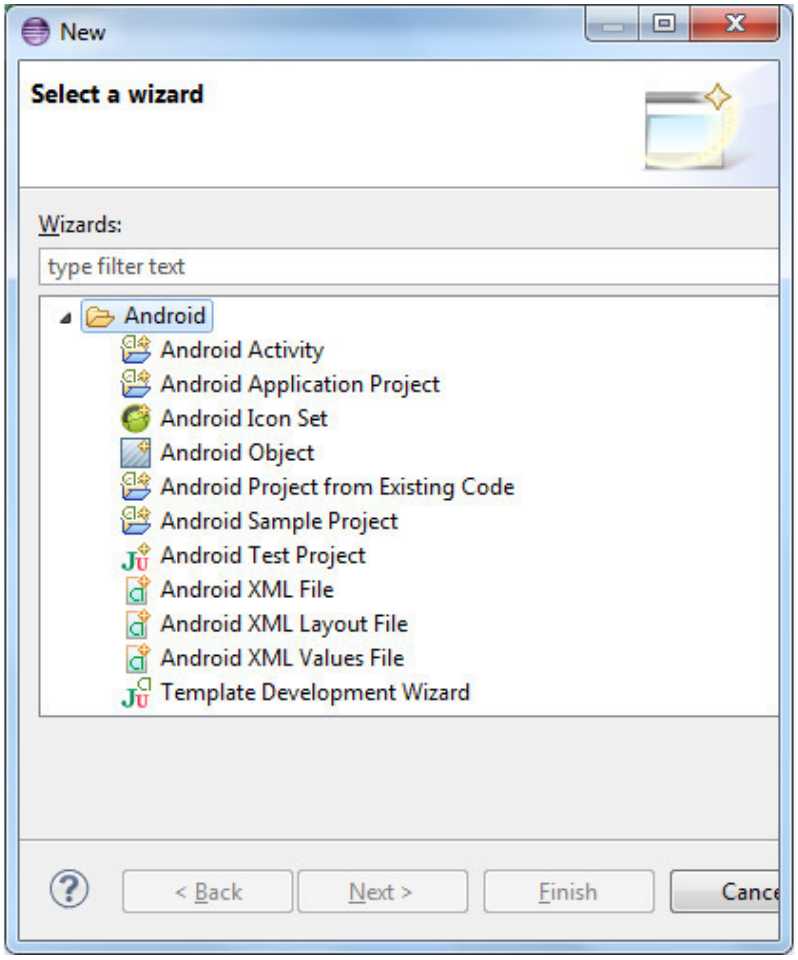
Preference – XML-описание GUI-интерфейса PreferenceActivity-операции, позволяющей пользователю персонифицировать приложение.

Searchable – XML-файл, определяющий настройки GUI-компонента поиска.

Android XML Layout File – аналог мастера **Android XML File | Layout**.

Android XML Values File – аналог мастера **Android XML File | Values**.

Template Development Wizard – генерирует код на основе файла шаблона template. xml.




После установки ADT-плагина, в контекстном меню окна **Package Explorer** появятся следующие команды:

Run As | Android Application – запускает Android-приложение в виртуальном мобильном устройстве, созданным с помощью AVD Manager.

Run As | Android JUnit Test – запускает набор тестов для Android-приложения с использованием виртуального мобильного устройства.

Android Tools | New Test Project – открывает мастер **Android Test Project** создания набора тестов для Android-приложения.

Android Tools | New Resource File – открывает мастер **Android XML File** создания ресурсов приложения.

Android Tools | Export Signed Application Package – открывает мастер **Export Android Application** экспорта пописанного цифровой подписью и готового к публикации Android-приложения.

Android Tools | Export Unsigned Application Package – экспортирует неподписанный для релиза APK-файл Android-приложения.

Android Tools | Display dex bytecode – в окне Eclipse-редактора отображает инструкции байткода, дизассемблированные из DEX-файла, который создается в процессе сборки приложения путем конвертации из Java класс-файлов для выполнения виртуальной машиной Dalvik среды выполнения Android.

Android Tools | Rename Application Package – переименовывает пакет приложения.

Android Tools | Add Support Library – запускает приложение SDK Manager для добавления в путь приложения библиотеки Android Support Package, предоставляющей дополнительный API, не являющийся частью API версии Android-платформы. Другой способ добавления

библиотеки Android Support Package – установить библиотеку с помощью раздела **Extras** приложения SDK Manager, создать папку libs в каталоге проекта, скопировать в нее библиотеку из папки extras\android\support каталога Android SDK и добавить библиотеку в путь приложения используя команду **Build Path | Configure Build Path** контекстного меню окна **Package Explorer**.

Android Tools | Fix Project Properties – в случае импорта готового Android-проекта гарантирует правильную его сборку, в частности добавляет в путь приложения необходимые библиотеки.

Android Tools | Run Lint: Check for Common Errors – сканирует Android-проект для поиска потенциальных багов с выводом сообщений о них в окно **Lint Warnings**.

Android Tools | Clear Lint Markers – очищает окно **Lint Warnings**.

Android Tools | Add Native Support – добавление поддержки Android NDK.

В меню **Windows** Workbench-окна появятся команды **Android SDK Manager**, **AVD Manager** и **Run Android Lint**, с помощью которых можно запустить приложения набора SDK Tools и сканирование Android-проекта для поиска потенциальных багов. Данные команды дублируются соответствующими кнопками панели инструментов Workbench-окна.

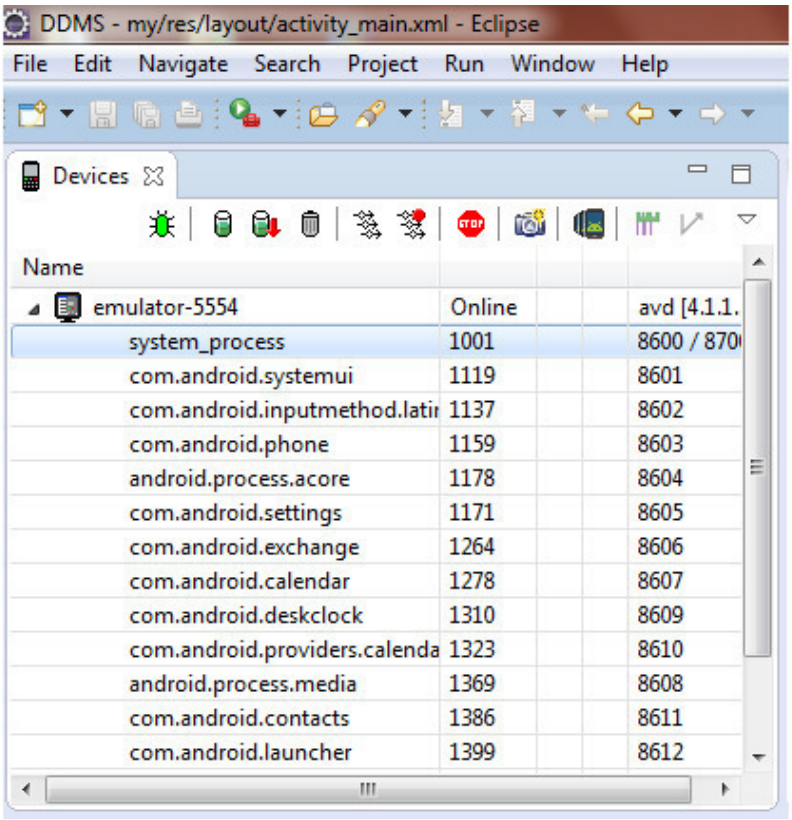
ADT-плагин добавляет в среду Eclipse перспективы **DDMS**, **Hierarchy View**, **Pixel Perfect**, **Tracer for OpenGL ES** и **XML**.

# Перспектива DDMS

Перспектива **DDMS** запускает инструмент отладки Dalvik Debug Monitor Server набора SDK Tools и отображает его GUI-интерфейс в виде набора Eclipse-представлений, обеспечивая информацию о работе эмулятора или подсоединенного Android-устройства.

Перспектива **DDMS** содержит представления **Devices**, **Emulator Control**, **LogCat**, **Threads**, **Heap**, **Allocation Tracker**, **Network Statistics**, **System Information** и **File Explorer**.

Представление **Devices** отображает подключенные Android-устройства. Для каждого подключенного Android-устройства Devices-представление показывает все запущенные на нем процессы, каждый из которых работает в своем экземпляре виртуальной машины Dalvik. Каждый отображаемый процесс представляет инсталлированное и запущенное на Android-устройстве приложение, поэтому идентификация процесса производится по имени пакета приложения. Так как виртуальная машина Dalvik работает поверх ядра Linux, каждый процесс имеет свой Linux-идентификатор, отображаемый в окне **Devices** после имени пакета. Крайний правый столбец окна **Devices** показывает номер порта, который DDMS-инструмент назначает для подсоединения Eclipse-отладчика к экземпляру Dalvik-машины с использованием протокола JDWP (Java Debug Wire Protocol). По умолчанию Eclipse-отладчик подсоединяется к статическому порту 8700, на который перенаправляются трафики экземпляров Dalvik-машины от всех портов. DDMS-инструмент взаимодействует с подключенным Android-устройством с помощью инструмента Android Debug Bridge (adb), имеющего клиент-серверную архитектуру. DDMS-инструмент создает adb-клиента, который взаимодействует с adb-демоном (фоновый процесс, работающий в Android-устройстве) через adb-сервер.




Панель инструментов представления **Devices** содержит следующие кнопки:

(Debug the selected process, …) – подсоединяет процесс, представляющий Android-приложение с открытым в среде Eclipse проектом, к Eclipse-отладчику, для работы с которым используется перспектива **Debug**.

(Update Heap) – включает информацию об использовании динамической памяти для процесса.

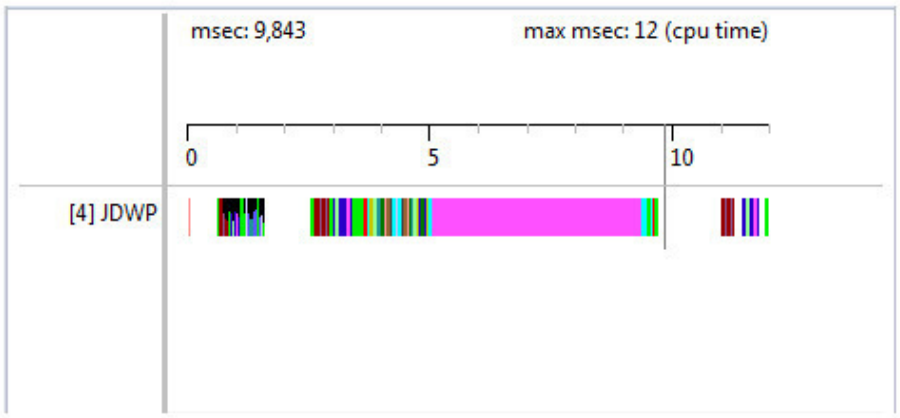
(Dump HPROF file) – создает снимок динамической памяти в виде HPROF-файла. В случае Android-устройств версии 2.1 и ранее для создания HPROF-файла требуется наличие SD-карты памяти, а также разрешения <uses-permission http://www.eclipse.org/mat/).rmission. WRITE_EXTERNAL_STORAGE»/> в файле манифеста AndroidManifest. xml Android-приложения. Анализ HPROF-файла можно выполнить с помощью Eclipse-плагина Memory Analyzer (MAT) (android:name="android.pe

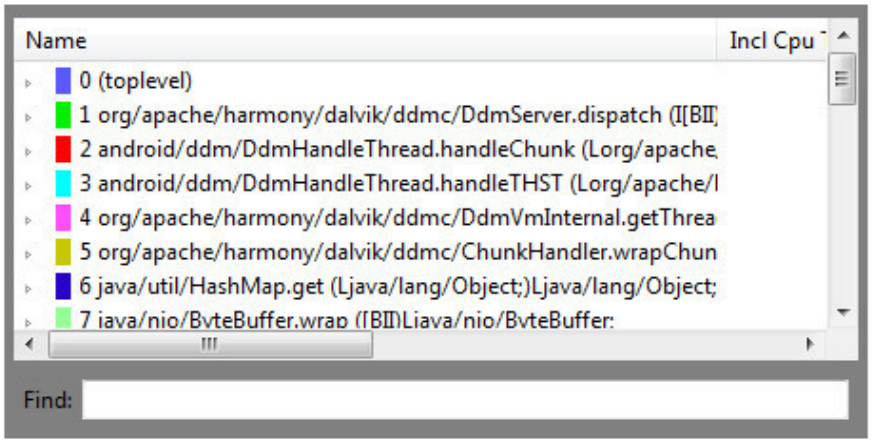
(Cause GC) – вызывает сборщика мусора, что влечет за собой сборку данных о динамической памяти.

(Update Threads) – включает информацию о запущенных потоках для выбранного процесса.

(Start Method Profiling) и (Stop Method Profiling) – запускает и останавливает запись информации о выполнении методов приложения в Trace-файл, который после остановки записи открывается в Traceview-окне, отображающем журнал выполнения в виде двух пане-

лей: Timeline Panel – с помощью цветовой гаммы и шкалы времени описывает старт и остановку выполнения метода в потоке, Profile Panel – показывает детали выполнения методов. В случае Android-устройств версии 2.1 и ранее для создания Trace-файла требуется наличие SD-карты памяти, а также разрешения <uses-permission android:name="android.permission. WRITE_EXTERNAL_STORAGE»/> в файле манифеста AndroidManifest. xml Android-приложения. За отображение окна Traceview отвечает инструмент traceview набора SDK Tools.






(Stop Process) – останавливает выбранный процесс.

(Screen Capture) – открывает окно **Device Screen Capture**, которое позволяет создавать скриншоты экрана Android-устройства.

(Dump View Hierarchy for UI Automator) – обеспечивает тестирование GUI-интерфейса приложения путем получения снимка экрана Tablet-устройства API 16 и выше, предоставляя визуальный интерфейс для проверки GUI-иерархии и просмотра свойств отдельных компонен-

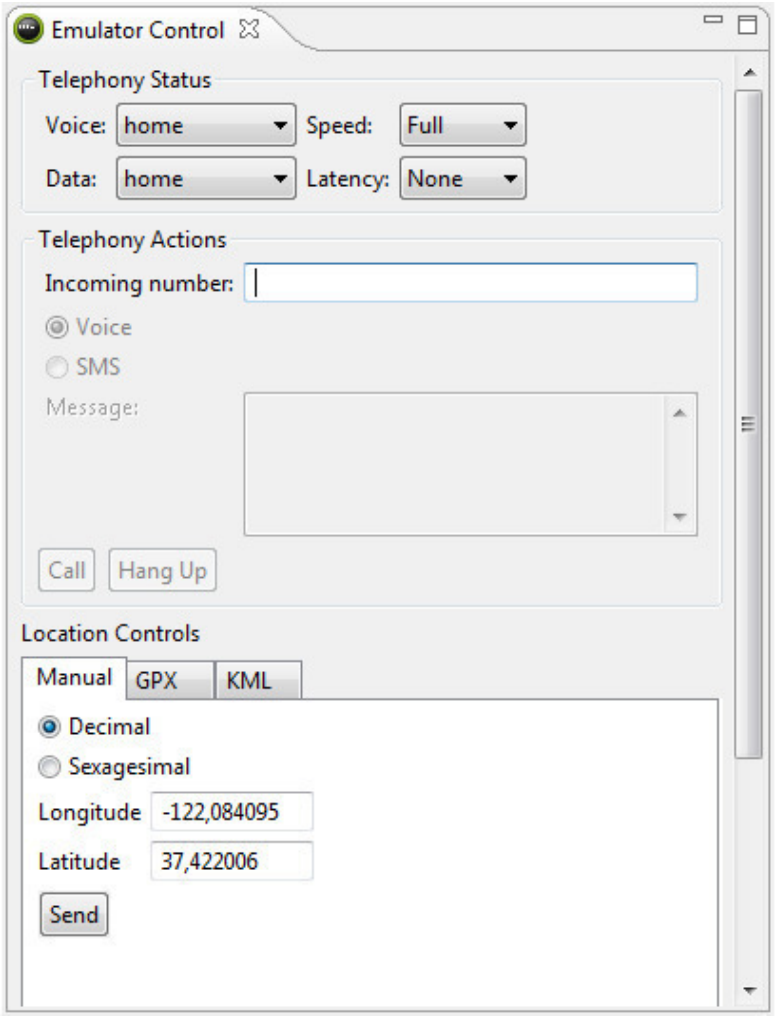
тов GUI-интерфейса. Работа команды обеспечивается инструментом uiautomatorviewer набора Android SDK.

(Capture system wide trace using Android systrace) – для устройства Android 4.1 (API Level 16) помогает анализировать производительность приложения, формируя журнал событий системы и приложения в виде HTML-файла.

(Start OpenGL Trace) – для устройства Android 4.1 (API Level 16) – помогает анализировать выполнение графических OpenGL ES команд.

Меню панели инструментов представления **Devices**, помимо вышеперечисленных команд, содержит команду **Reset adb**, обеспечивающую перезапуск adb-инструмента.

Представление **Emulator Control** дает возможность имитировать для экземпляра Android-эмулятора входящий звонок, SMS-сообщение и локализацию.




Списки **Voice:** и **Data:** раздела **Telephony Status** представления **Emulator Control** позволяют установить состояние GPRS-соединения:

unregistered – сеть отсутствует.

home – локальная сеть.

roaming – телефон в роуминге.

searching – поиск сети.

denied – только звонки экстренных служб.

Список **Speed:** раздела **Telephony Status** представления **Emulator Control** позволяет установить скорость передачи данных сети:

GSM – 14.4 килобит\сек.

HSCSD – от 14.4 до 43.2 килобит\сек.

GPRS – от 40.0 до 80.0 килобит\сек.

EDGE – от 118.4 до 236.8 килобит\сек.

UMTS – от 128.0 до 1920.0 килобит\сек.

HSDPA – от 348.0 до 14400.0 килобит\сек.

Full – без ограничений.

Список **Latency:** раздела **Telephony Status** представления **Emulator Control** позволяет имитировать уровень задержки сети:

GPRS – от 150 до 550 миллисекунд.

EDGE – от 80 до 400 миллисекунд.

UMTS – от 35 до 200 миллисекунд.

None – задержка отсутствует.

Раздел **Telephony Actions** представления **Emulator Control** дает возможность имитировать входящий звонок и SMS-сообщение.

Раздел **Location Controls** представления **Emulator Control** обеспечивает определение локализации Android-устройства вручную (вкладка **Manual**) или с помощью файлов GPS eXchange (вкладка **GPX**) и Keyhole Markup Language (вкладка **KML**).

Представление **LogCat** обеспечивает отображение всех системных сообщений от Android-устройства, в то время как представление **Console** показывает только сообщения, относящиеся к изменениям состояния Android-устройства и его приложений.

LogCat-окно отображает системные сообщения в таблице, содержащей столбцы Level (приоритет сообщения), Time (время создания сообщения), PID (Linux-идентификатор процесса), Application (имя пакета приложения), Tag (идентификатор системного компонента, от которого получено сообщение), Text (текст сообщения). Соответственно панель инструментов представления **LogCat** обеспечивает фильтрацию отображаемых сообщений по приоритету, тэгу, по идентификатору и имени пакета приложения.

Представление **Threads** показывает запущенные потоки для выбранного процесса. Для просмотра потоков необходимо в окне **Devices** выбрать процесс и нажать кнопку **Update Threads** панели инструментов окна **Devices**.

Threads-окно отображает информацию о потоках в виде двух таблиц. Верхняя таблица показывает все запущенные потоки для выбранного процесса и имеет следующие столбцы:

ID – Dalvik-идентификатор потока – нечетные числа, начиная с 3. Демоны помечаются «*».

TID – Linux-идентификатор потока.

Status – статус потока:

Wait – вызван метод Object. wait ().

Native – выполняет системный код.

Vmwait – ожидает Dalvik-ресурс.

Runnable – может быть запущен.

TimedWait – ожидает в течение определенного количества времени.

utime – общее время выполнения пользовательского кода (единица 10 мс.).

stime – общее время выполнения системного кода (единица 10 мс.).

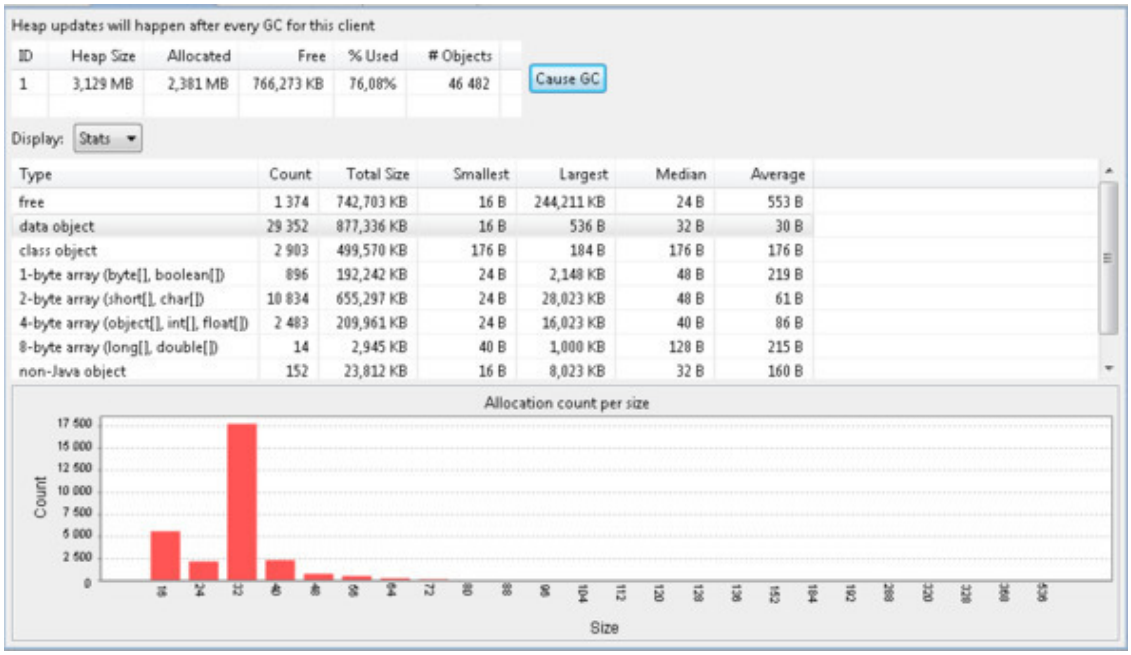
Name – имя потока.

Нижняя таблица для выбранного потока показывает выполняемый потоком код, указывая класс, метод, файл, строку и признак кода.

Представление **Heap** отображает информацию об использовании динамической памяти выбранным процессом. Для просмотра кучи процесса в Heap-окне необходимо в окне **Devices** выбрать процесс и нажать кнопку **Update Heap**, затем кнопку **Cause GC** панели инструментов окна **Devices**.

Представление **Heap** содержит три области. Самая верхняя область показывает таблицу структуры кучи процесса со столбцами ID (идентификатор кучи), Heap Size (общее количество памяти кучи), Allocated (количество занятой памяти кучи), Free (количество свободной памяти кучи), %Used (процент занятости кучи) и #Objects (количество объектов кучи), а также имеет кнопку **Cause GC** обновления информации о куче.

Далее расположена область с таблицей распределения объектов кучи по типам. Самая нижняя область отображает гистограмму распределения выбранного типа объектов по размерам занимаемой памяти.



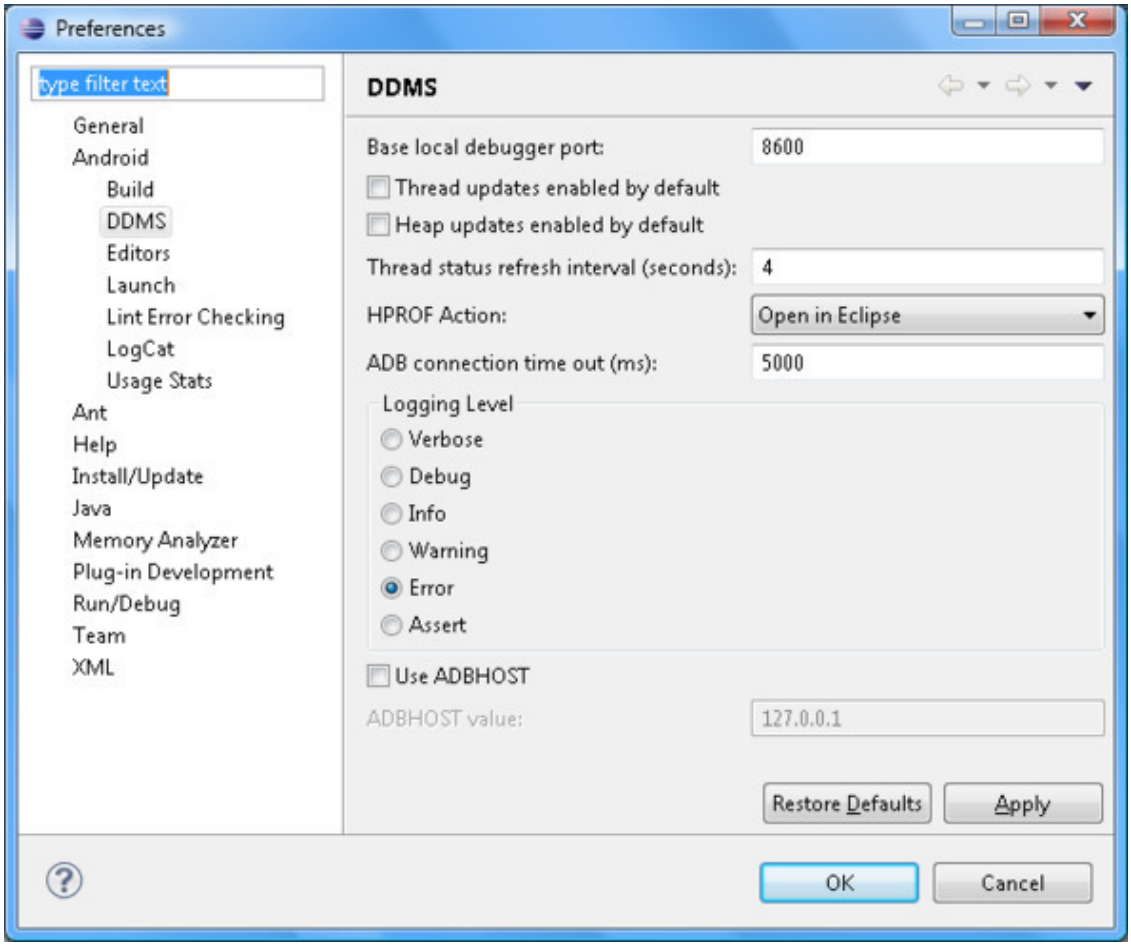

Представление **Allocation Tracker** позволяет в реальном времени отследить объекты, для которых выделяется память. Для начала просмотра журнала выделения памяти для объектов необходимо в окне **Devices** выбрать процесс и нажать кнопку **Start Tracking** в окне **Allocation Tracker**, затем кнопку **Get Allocations**. В результате верхняя область окна **Allocation Tracker** покажет список объектов, созданных с момента нажатия кнопки **Start Tracking** до момента нажатия кнопки **Get Allocations**, с указанием выделенной памяти, идентификатора потока, класса и метода, а нижняя область – более детальную информацию для выбранного объекта, с указанием класса, метода, файла, строки и признака кода.

Представление **Network Statistics** позволяет сформировать и проанализировать журнал передачи данных по сети.

Представление **System Information** отображает диаграммы использования системных ресурсов.

Представление **File Explorer** показывает файловую систему Android-устройства с возможностью экспорта и импорта файлов, удаления файлов и создания новых папок.

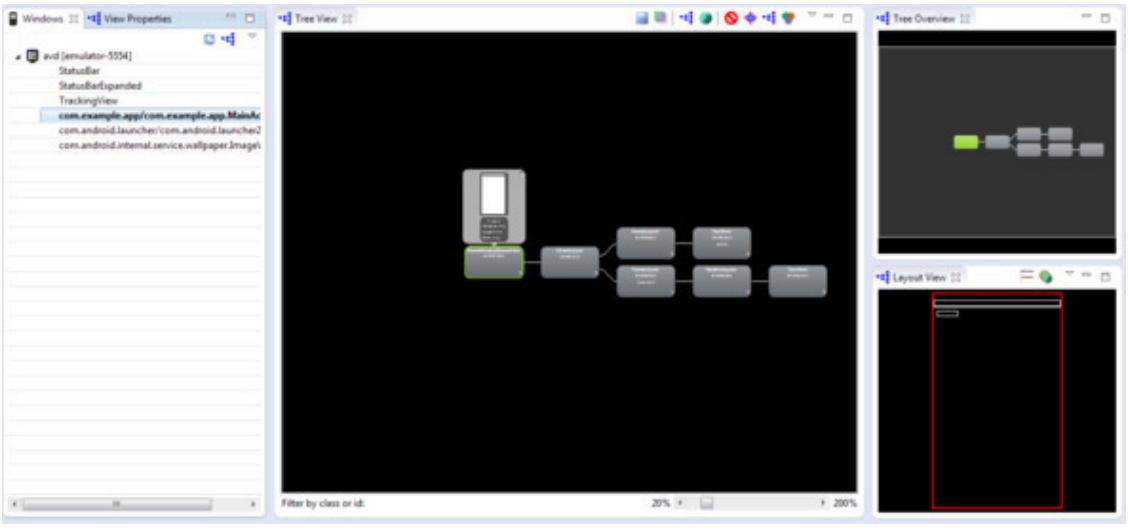
Общая настройка DDMS-инструмента осуществляется с помощью раздела **Android | DDMS** команды **Preferences** меню **Window**, где можно определить номер порта, с которого DDMS-инструмент начинает назначать порт для подсоединения Eclipse-отладчика к экземпляру Dalvik-машины по протоколу JDWP, обновление по умолчанию информации о куче и потоках с указанным интервалом, сохранение HPROF-файла или открытие его в среде Eclipse, время ожидания adb-инструмента, adb-хост для связи с Android-устройством по сети.


Preferences
type filter text
General
Android
Build
DDMS
Editors
Launch
Lint Error Checking
LogCat
Usage Stats
Ant
Help
Install/Update
Java
Memory Analyzer
Plug-in Development
Run/Debug
Team
XML
DDMS
Base local debugger port:
8600
Thread updates enabled by default
Heap updates enabled by default
Thread status refresh interval (seconds):
4
HPROF Action:
Open in Eclipse
ADB connection time out (ms):
5000
Logging Level
Verbose
Debug
Info
Warning
Error
Assert
Use ADBHOST
ADBHOST value:
127.0.0.1
Restore Defaults
Apply
OK
Cancel

# Перспективы Hierarchy View и Pixel Perfect

Перспективы **Hierarchy View** и **Pixel Perfect** запускают инструмент hierarchyviewer набора SDK Tools и отображают его GUI-интерфейс в виде набора Eclipse-представлений, помогая отладить и оптимизировать GUI-интерфейс Android-приложения.

Перспектива **Hierarchy View** содержит представления **Windows**, **View Properties**, **Tree View**, **Tree Overview**, **Layout View**.



Представление **Windows** отображает список подключенных Android-устройств, для каждого из которых показывает список Activity-объектов, включающий в себя системные объекты и объект приложения, GUI-интерфейс которых отображается на экране Android-устройства.

Представление **Tree View** показывает иерархию GUI-компонентов графического интерфейса выбранного Activity-объекта. Для просмотра диаграммы иерархии GUI-интерфейса приложения в представлении **Tree View**, в окне **Windows** необходимо выбрать Activity-объект приложения и нажать кнопку **Load the view hierarchy into the tree view** панели инструментов Windows-окна. В результате в окне **Tree View** отобразится иерархия View-объектов приложения.

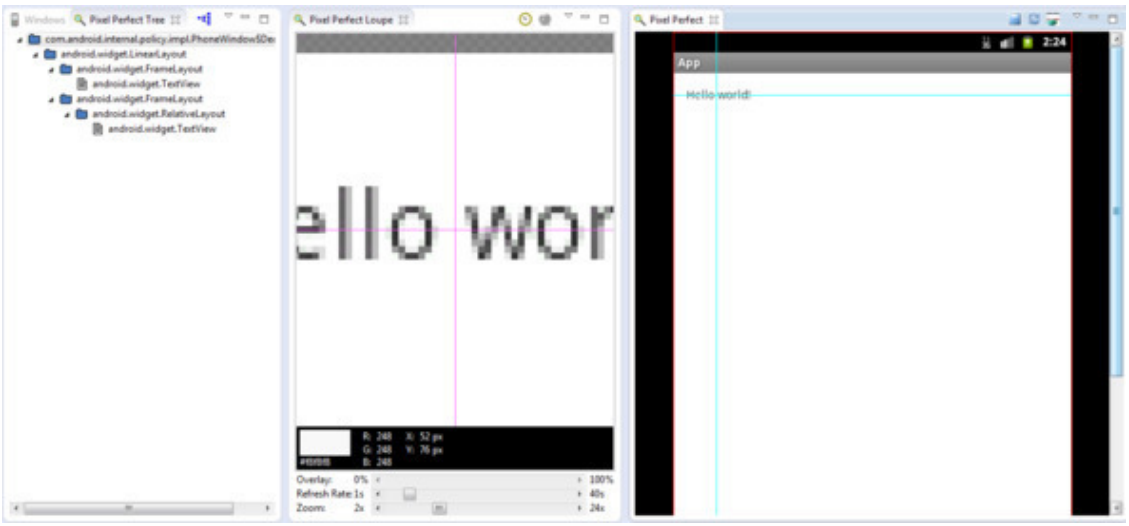
Диаграмму окна **Tree View** можно увеличивать с помощью нижнего ползунка, фильтровать, используя поле **Filter by class or id:**. Панель инструментов представления **Tree View** позволяет сохранить отображаемую диаграмму как PNG-изображение, как PSD-документ, открыть выбранный View-объект в отдельном окне и др.

Представление **Tree Overview** обеспечивает перемещение по диаграмме окна **Tree View** с помощью перетаскивания выделенного прямоугольника окна **Tree Overview**.

Окно **Layout View** является блочным представлением GUI-интерфейса. При выборе компонента диаграммы окна **Tree View** его расположение в GUI-интерфейсе подсвечивается красным цветом в окне **Layout View**. Также при выборе View-объекта диаграммы окна **Tree View**, выше его узла появляется небольшое окно с реальным изображением GUI-компонента и информацией о количестве View-объектов, представляющих компонент, и о времени отображения компонента в миллисекундах. При этом свойства выбранного View-объекта диаграммы отображаются в представлении **View Properties**.

Информация о времени визуализации компонентов GUI-интерфейса приложения помогает найти причину его медленной работы.

Перспектива **Pixel Perfect** содержит представления **Windows**, **Pixel Perfect Tree**, **Pixel Perfect Loup** и **Pixel Perfect**.




Представление **Windows** отображает список подключенных Android-устройств без детализации. При выборе устройства и нажатии кнопки **Inspect a screenshot in the pixel perfect view** панели инструментов окна **Windows** снимок экрана выбранного Android-устройства открывается в представлениях **Pixel Perfect Loup** и **Pixel Perfect**. В представлении **Pixel Perfect Tree** отображается дерево View-объектов GUI-интерфейса приложения, формирующего снимок экрана. При выборе View-объекта в окне **Pixel Perfect Tree**, его расположение обозначается красной рамкой в окне **Pixel Perfect**.

Представление **Pixel Perfect Loup** содержит перекрестье, которое дает информацию о пикселе, находящемся в центре пересечения, включающую в себя HTML-код цвета пикселя, его RGB-значение и координаты. Изображение окна **Pixel Perfect Loup** можно перемещать мышкой относительно перекрестья. Слайдер **Zoom** позволяет регулировать увеличение снимка экрана Android-устройства.

Представление **Pixel Perfect** также содержит перекрестье, расположение которого относительно снимка экрана совпадает с расположением перекрестья окна **Pixel Perfect Loup** и наоборот. Перекрестье окна **Pixel Perfect** можно передвигать мышкой, а панель инструментов окна **Pixel Perfect** дает возможность сохранить снимок экрана как PNG-изображение, а также загрузить поверх снимка экрана другое изображение, представляющее макет GUI-интерфейса приложения, при этом прозрачность загруженного изображения можно регулировать с помощью слайдера **Overlay:** окна **Pixel Perfect Loup**.

Возможность загрузки изображений поверх снимка экрана Android-устройства помогает в работе над дизайном GUI-интерфейса разрабатываемого Android-приложения.

# Wizard мастера ADT плагина

**Мастер Android Project**

Для создания Android-приложения откроем среду Eclipse с инсталлированным ADT-плагином и в меню **File** выберем команду **New | Other | Android | Android Application Project** и нажмем кнопку **Next**.

Введем имя приложения, отображаемое в устройстве, имя проекта, имя пакета. Выберем минимальную версию SDK, предпочтительную версию SDK, версию SDK относительно которой приложение будет компилироваться, тему приложения и нажмем кнопку **Next**. Оставим отмеченными флажки **Create custom launcher icon** и **Create activity** и нажмем кнопку **Next**. Определим значок приложения и нажмем кнопку **Next**. Выберем создаваемый Activity-компонент и нажмем кнопку **Next**:

Blank Activity – экран с надписью «Hello world!».

Blank Activity with Fragment – экран с фрагментом с надписью «Hello world!».

Empty Activity – все равно экран с надписью «Hello world!».

Fullscreen Activity – экран, нажатие на который вызывает переключение между обычным и полноэкранным режимами.

Master/Detail Flow – экран с боковой панелью меню.

Navigation Drawer Activity – экран с двумя фрагментами, панелью навигации и контентом.

Tabbed Activity – экран с вкладками и типом навигации: с помощью жеста Swipe Views (ViewPager), с помощью панели закладок Action Bar Tabs (with ViewPager), с помощью выпадающего списка Action Bar Spinner.

Определим имя Activity-компонента, имя компоновочного файла res/layout/ activity_main. xml и нажмем кнопку **Finish** – в результате будет сгенерирована основа проекта Android-приложения.

Модель программирования Android-приложений основывается не на конструкции с главным классом приложения, имеющим точку входа – статический метод main (), а является компонентной моделью. Android-приложение может состоять из одного или нескольких компонентов, объявленных в файле манифеста приложения AndroidManifest. xml и относящихся к четырем типам:

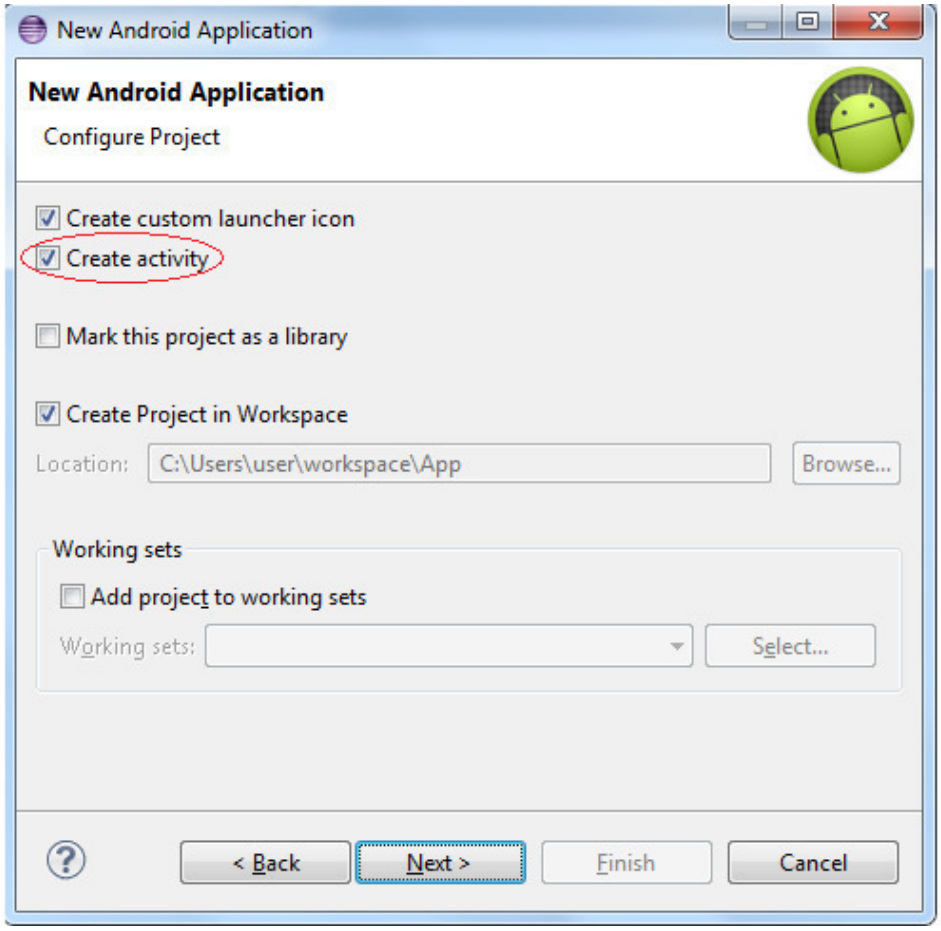
Activity – расширение класса android.app.Activity, обеспечивающее создание одного окна на экране Android-устройства с формированием в нем GUI-интерфейса.

Service – расширение класса android.app.Service, обеспечивающее выполнение операций без предоставления GUI-интерфейса.

BroadcastReceiver – расширение класса android.content.BroadcastReceiver, отвечающее за прослушивание широковещательных сообщений с запуском других компонентов Android-приложения или выводом уведомлений пользователю в строку статуса.

ContentProvider – расширение класса android.content.ContentProvider, обеспечивающее хранение и извлечение общих данных.

Существующая версия ADT-плагина при создании Android-проекта предлагает формирование основы только Activity-компонента.




Основа самого простого Android-проекта, сгенерированная средой Eclipse, состоит из следующих узлов окна **Package Explorer**:

src – содержит пакет класса, расширяющего класс android.app.Activity.

gen – содержит R-класс, автоматически генерируемый инструментом aapt набора SDK Platform-tools из существующих ресурсов проекта для программного к ним доступа, а также класс BuildConfig, содержащий константу DEBUG, которая со значением true определяет запуск приложения в режиме отладки. При экспорте подписанного приложения значение константы DEBUG автоматически становится false.

Android x.x – библиотека Android-платформы, на основе которой создается приложение.

Android Private Libraries – дополнительная библиотека android-support, обеспечивающая обратную совместимость с предыдущими версиями Android API.

assets – каталог предназначен для хранения данных приложения, доступ к которым осуществляется с помощью класса android.content.res.AssetManager. Отличие данного каталога от каталога res заключается в том, что он не должен иметь строго преопределенной структуры, которая для каталога res обеспечивает автоматическую генерацию R-класса.

bin – каталог сборки приложения.

libs – содержит JAR-файл библиотеки android-support.

res – содержит ресурсы приложения, доступ к которым осуществляется с помощью R-класса, и имеет строго предопределенную структуру:

animator – XML-файлы для создания объектов анимации.

color – XML-файлы, определяющие цветовую гамму View-объектов.

drawable – PNG, JPEG, GIF, 9-PNG и XML-файлы, формирующие графику.

layout – XML-файлы для формирования структуры GUI-интерфейса Activity-объектов.

menu – XML-файлы, описывающие меню приложения.

raw – каталог предназначен для хранения таких данных приложения как файлов в формате MP3 или Ogg.

values – XML-файлы для хранения строк, стилей, чисел, размеров и др., используемых приложением, в виде пар имя-значение.

xml – различные конфигурационные и ресурсные XML-файлы.

AndroidManifest. xml – файл манифеста приложения, определяющий запуск Android-приложения средой выполнения Android и описывающий Android-компоненты приложения, права пользователя, минимальный уровень API Android-платформы, необходимый для запуска приложения, требуемые опции Android-устройства и др.

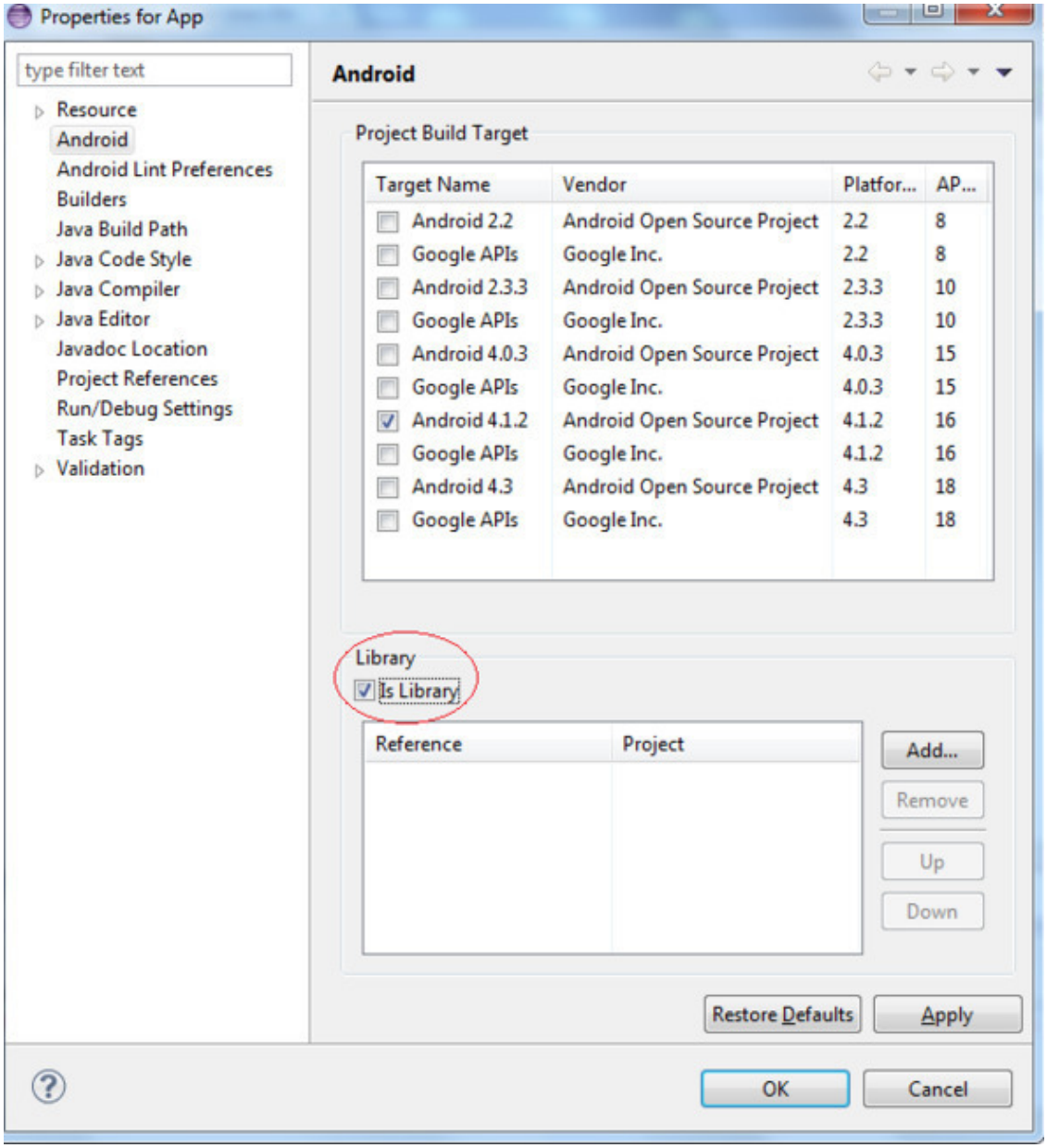
ic_launcher-web.png – значок приложения для магазина Google Play Store.

proguard.cfg – файл инструмента proguard набора SDK Tools, обеспечивающего сокращение, оптимизацию и обфускацию кода.

project.properties – содержит установки проекта.

Созданный Android-проект можно перевести в статус библиотеки, предоставляющей исходный код и ресурсы для других Android-проектов. При этом Android-библиотека не может содержать ресурсы в каталоге assets и версия Android-платформы библиотеки должна быть меньше или равна версии Android-платформы проекта, использующего библиотеку.

Для создания Android-библиотеки нужно в окне **Package Explorer** нажать правой кнопкой мышки на узле Android-проекта и в контекстном меню выбрать команду **Properties**. Далее в разделе **Android** отметить флажок **Is Library** и нажать кнопку **OK**.




Для использования созданной Android-библиотеки другим Android-проектом необходимо в окне **Package Explorer** нажать правой кнопкой мышки на узле Android-проекта и в контекстном меню выбрать команду **Properties**. Далее в разделе **Android** нажать кнопку **Add** и выбрать Android-библиотеку.

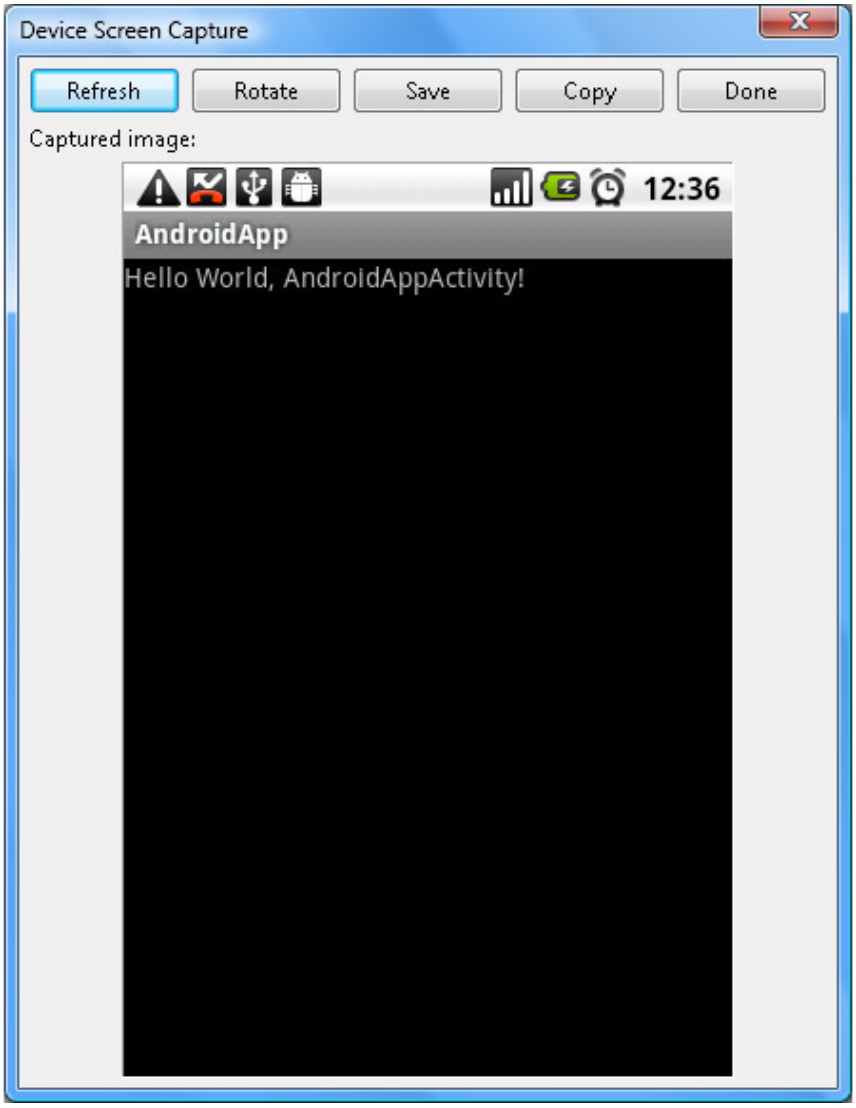
В результате в окне **Package Explorer** в Android-проект добавится узел **Library Projects**, содержащий временный JAR-файл Android-библиотеки, код и ресурсы которой можно использовать в проекте.

# Запуск Android-приложения из среды Eclipse

Перед тем как воспользоваться командой **Run As | Android Application** контекстного меню окна **Package Explorer** для тестирования Android-приложения в реальной среде выполнения, необходимо либо подсоединить к компьютеру реальное Android-устройство, либо создать экземпляр Android-эмулятора.

Для тестирования Android-приложения в реальном Android-устройстве нужно зайти в настройки устройства и открыть раздел **Приложения**. В разделе **Приложения** отметить флажок **Неизвестные источники**, затем открыть раздел **Разработка** и отметить флажок **Отладка USB**. После чего инсталлировать драйвер устройства на компьютер и подсоединить устройство к компьютеру. В результате среда Eclipse произведет опознание устройства, которое отобразится в окне **Devices** (команда **Window | Show View | Android | Devices**).

Для запуска Android-приложения выберем команду **Run As | Run Configurations** контекстного меню окна **Package Explorer** и в разделе приложения во вкладке **Target** отметим флажок **Always prompt to pick device**. Нажмем кнопку **Run**, в окне **Android Device Chooser** выберем устройство и нажмем кнопку **OK**. В результате Android-приложение будет инсталлировано и запущено в реальном Android-устройстве. Нажав кнопку **Screen Capture** панели инструментов окна **Devices** можно сделать снимок экрана реального Android-устройства.




Для тестирования Android-приложения в Android-эмуляторе нужно запустить приложение AVD Manager и для создания виртуального Android-устройства во вкладке **Android Virtual Devices** нажать кнопку **Create**:

В поле **AVD Name:** ввести имя устройства.

В списке **Device:** выбрать тип устройства.

В списке **Target:** выбрать версию Android-платформы устройства.

В списке **CPU/ABI:** выбрать тип процессора.

В списке **Skin:** выбрать оболочку эмулятора – установка аппаратной клавиатуры и отображение кнопок устройства в оболочке Android-эмулятора.

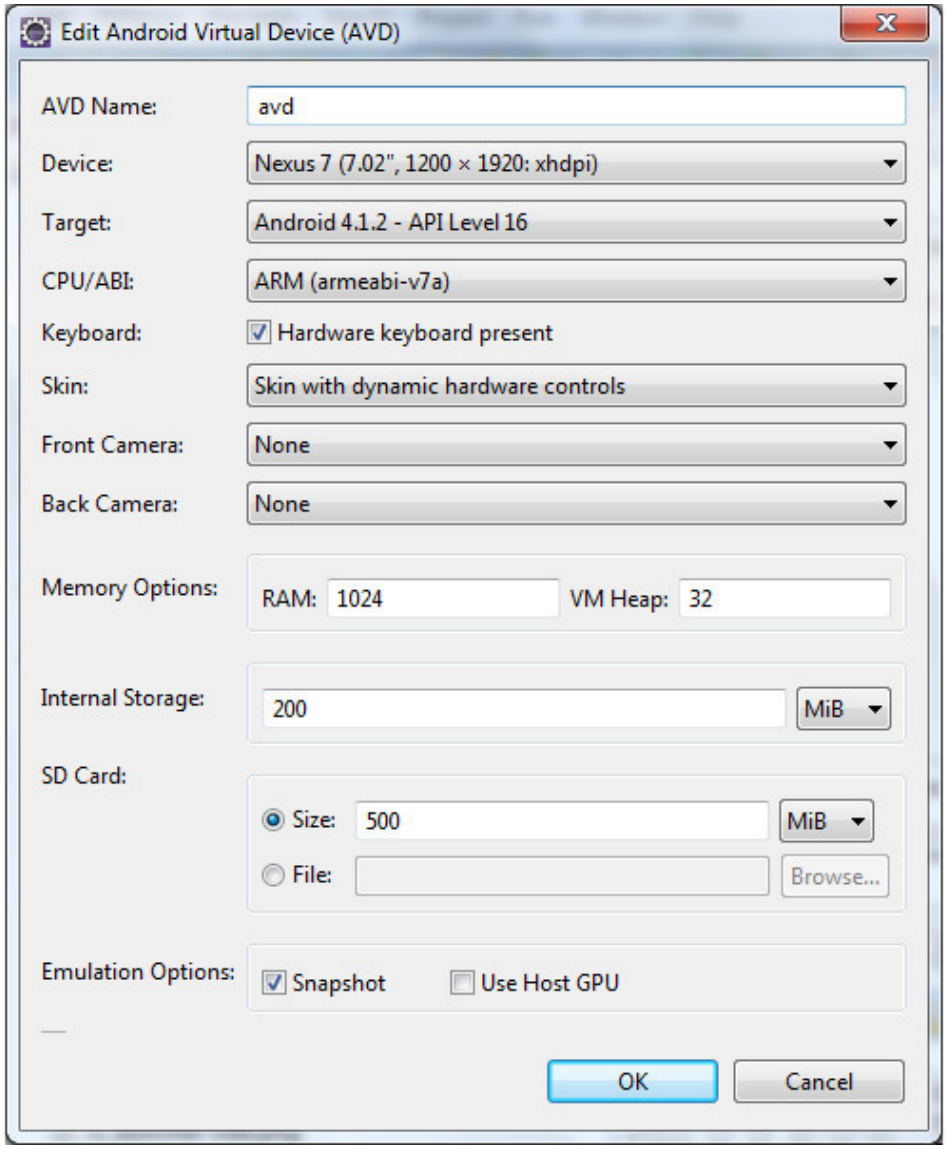
В списках **Front Camera:** и **Back Camera:** определить камеры устройства.

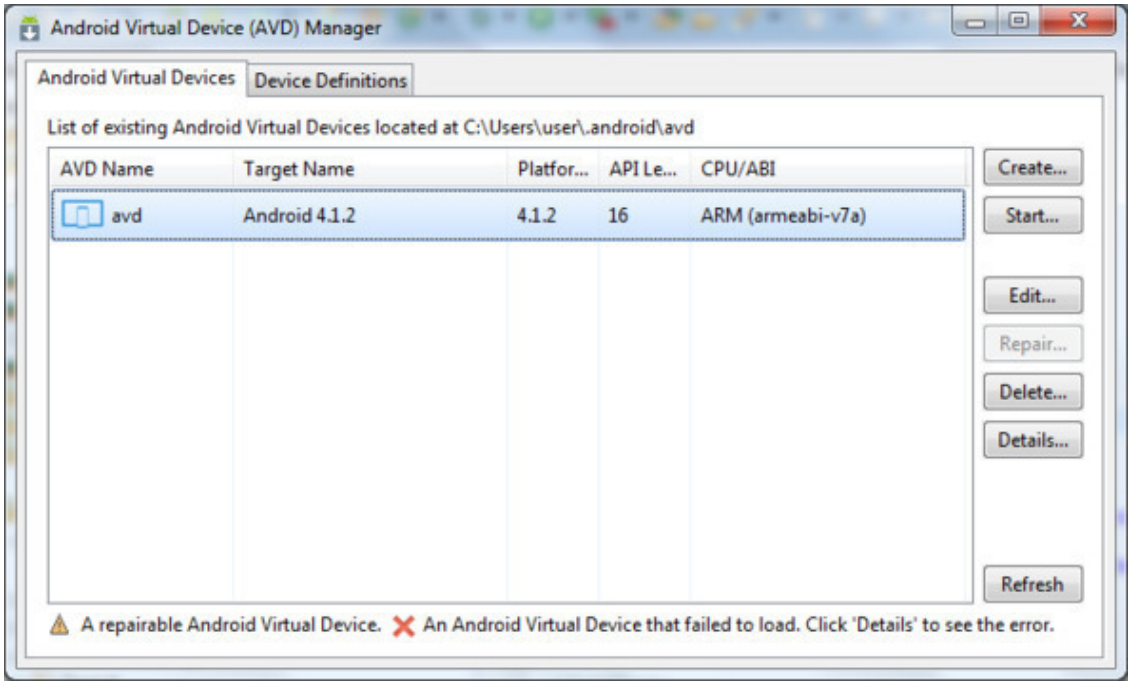
В разделе **Memory Options:** определить размер оперативной памяти и максимальный размер кучи, выделяемый для работы одного Android-приложения.

В разделах **Internal Storage:** и **SD Card:** определить размер внутренней памяти и тип/размер карты памяти устройства. Поле **File:** раздела **SD Card:** предназначено для определения образа карты памяти, созданного с использованием инструмента mksdcard набора SDK Tools.

В разделе **Emulation Options:** с помощью флажка **Snapshot** определить ускорение повторного запуска виртуального устройства, так как его состояние будет сохраняться при закрытии, с помощью флажка **Use Host CPU** определить ускорение работы эмулятора, так как он будет использовать CPU компьютера.

И нажать кнопку **OK**. В результате в окне **Android Virtual Device Manager** появится созданное виртуальное устройство, которое нужно запустить кнопкой **Start**.






Кнопка **Create Device** вкладки **Device Definitions** позволяет создать новое устройство списка **Device**.

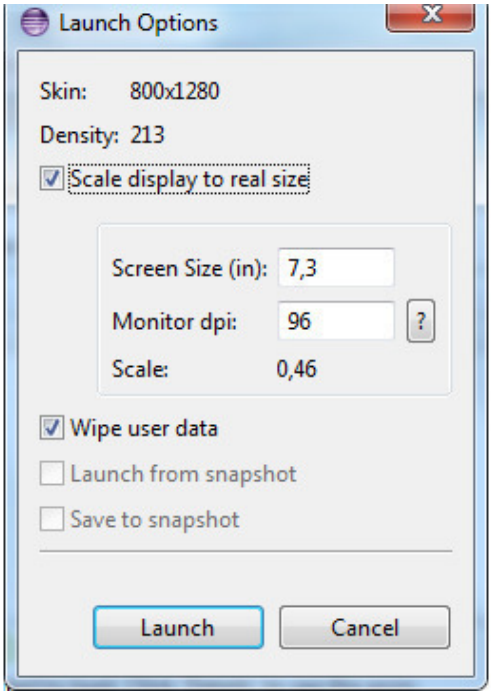
При запуске кнопкой **Start** виртуального устройства появляется окно **Launch Options** определения опций запуска, в котором:

Флажки **Launch from snapshot** и **Save to snapshot** работают в случае отмеченного флажка **Snapshot** раздела **Emulation Options:** и определяют загрузку и сохранение состояния виртуального устройства при закрытии.

Флажок **Scale display to real size** определяет масштабирование виртуального устройства путем установки диагонали и плотности экрана.

Флажок **Wipe user data** позволяет при запуске виртуального устройства стереть его сохраненное состояние, при этом будет отключена опция **Launch from snapshot**.

Виртуальное устройство окончательно запускается кнопкой **Launch** окна **Launch Options**.




По умолчанию приложение AVD Manager создает конфигурацию виртуального устройства в каталоге [user]\.android\avd файловой системы компьютера. Если в пути данного каталога будут присутствовать русские буквы, виртуальное устройство не запустится. Кроме того, запуск виртуального устройства занимает значительное время, поэтому рекомендуется как можно реже его закрывать.

После запуска виртуальное устройство отобразится на экране компьютера.


Для запуска Android-приложения в виртуальном Android-устройстве воспользуемся командой **Run As | Android Application** контекстного меню окна **Package Explorer**.

В результате запуска Android-приложения каталог bin Android-проекта заполнится файлами и папками:

Папка classes – откомпилированные Java класс-файлы, включая классы приложения и R-классы.

Папка dexedLibs – конвертированные в DEX-формат виртуальной машины Dalvik дополнительные библиотеки.

Папка res – ресурсы приложения.

Файл манифеста AndroidManifest. xml.

Бинарный файл resources.arsc строк приложения.

Файл classes.dex – конвертированные в DEX-формат виртуальной машины Dalvik Java класс-файлы.

APK-файл – ZIP-архив Android-приложения для инсталляции в отладочном режиме, содержащий папку META-INF с Java-манифестом MANIFEST. MF и сертификатами, папку res с ресурсами, файл манифеста AndroidManifest. xml, файл resources.arsc и файл classes.dex.

# Подготовка к публикации Android-приложения

Среда выполнения Android при инсталляции приложений требует, чтобы все Android-приложения были подписаны цифровой подписью с помощью сертификата, закрытый ключ которого имеется в распоряжении разработчиков приложений. Однако в отладочном режиме инструменты сборки приложения набора Android SDK автоматически подписывают приложение специальным отладочным ключом, который генерируется и по умолчанию хранится в файле debug.keystore каталога [user]\.android. По умолчанию отладочный сертификат имеет срок действия 365 дней и по истечении этого периода необходимо удалить файл debug.keystore для повторной автоматической генерации сертификата.

Для того чтобы подготовить Android-приложение к реальной инсталляции в Android-устройстве, так как среда выполнения Android не позволит инсталлировать приложение, подписанное отладочным ключом, можно воспользоваться командой **Android Tools | Export Signed Application Package** контекстного меню окна **Package Explorer –** в результате откроется мастер **Export Android Application**, в поле **Project:** которого будет отображено имя Android-проекта и в окне которого надо нажать кнопку **Next**.

Далее необходимо создать хранилище своего закрытого ключа, которым будут подписываться все версии данного Android-приложения. Важно использовать один постоянный ключ для одного Android-приложения, так как Android-система позволит обновление приложения только в том случае, если сертификаты старой и новой версии будут совпадать – иначе придется изменять имя пакета приложения, и новая версия будет уже устанавливаться как новое приложение. Кроме того, если приложение имеет модульную структуру и все модули подписаны одним сертификатом, тогда все модули будут запускаться в одном процессе, и смогут обновляться независимо друг от друга.

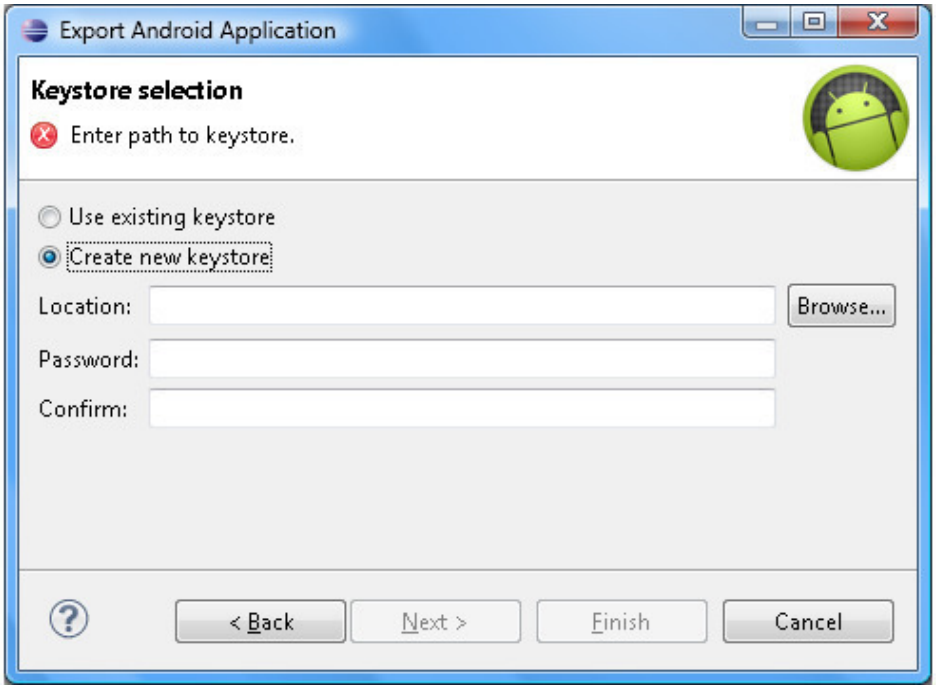
Для создания хранилища закрытого ключа в окне **Export Android Application** выберем переключатель **Create new keystore**, в поле **Location:** введем путь файла хранилища, в полях **Password:** и **Confirm:** введем пароль хранилища и нажмем кнопку **Next**.

***Примечание***

Для поля **Location:** необязательно использовать кнопку **Browse –** можно вручную ввести путь несуществующего файла, например, C:\Users\user\my_keystore, где файл my_keystore будет сгенерирован.

В поле **Alias:** введем имя ключа, в полях **Password:** и **Confirm:** введем пароль ключа, в поле **Validity (years):** введем срок действия сертификата более 25 лет, в поле **First and Last Name:** введем имя разработчика и нажмем кнопку **Next**, в поле **Destination APK file:** введем путь APK-файла Android-приложения и нажмем кнопку **Finish**.

В результате будет создан подписанный и готовый к публикации файл приложения, обработанный при этом оптимизирующим инструментом zipalign набора SDK Tools.


Export Android Application
Keystore selection
Enter path to keystore.
Use existing keystore
Create new keystore
Location:
Browse...
Password:
Confirm:
< Back
Next >
Finish
Cancel

# Activity-компонент

Сгенерированная мастером **Android Project** основа Android-проекта Blank Activity в узле **src** окна **Package Explorer** содержит файл исходного кода Activity-компонента, в котором его класс, расширяющий класс android.app.Activity, переопределяет метод onCreate ().

Данный метод является одним из методов обратного вызова Activity-компонента, которые среда выполнения Android вызывает при переходе Activity-компонента между различными состояниями его жизненного цикла. Переопределение метода onCreate () является важным, так как он вызывается при запуске Activity-компонента и предназначен для инициализации GUI-интерфейса.

Помимо метода onCreate () класс android.app.Activity предоставляет следующие методы обратного вызова для их переопределения:

onRestart – метод жизненного цикла, вызывается после того как Activity-компонент был остановлен, перед вызовом метода onStart.

onStart – метод жизненного цикла, вызывается, когда Activity-компонент становится видимым.

onResume – метод жизненного цикла, вызывается, когда Activity-компонент помещается на передний план для взаимодействия с пользователем.

onPause – метод жизненного цикла, вызывается, когда Activity-компонент помещается на задний план. После данного метода может вызываться метод onResume, если Activity-компонент помещается снова на передний план, или метод onStop, если Activity-компонент становится невидимым.

onStop – метод жизненного цикла, вызывается, когда Activity-компонент становится невидимым. После данного метода может вызываться метод onRestart или метод onDestroy.

onDestroy – метод жизненного цикла, вызывается перед уничтожением Activity-компонента программным способом методом finish класса android.app.Activity или Android-системой для освобождения ресурсов.

onActionModeFinished – вызывается при окончании работы режима контекстного меню ActionMode.

onActionModeStarted – вызывается при запуске режима контекстного меню ActionMode для Activity-компонента.

onActivityResult – при запуске другого Activity-компонента методом startActivityForResult вызывается после закрытия запущенного Activity-компонента для обработки возвращаемых им результатов.

onAttachFragment – вызывается при присоединении объекта Fragment к объекту Activity между вызовами методов жизненного цикла Fragment. onAttach и Fragment. onCreate.

onAttachedToWindow – вызывается при присоединении окна Activity-компонента к Window-менеджеру, метод может быть использован вместо метода onCreate.

onBackPressed – вызывается при нажатии пользователем клавиши Back.

onConfigurationChanged – вызывается при изменении конфигурации устройства во время работы Activity-компонента, при этом информацию о новой конфигурации предоставляет объект android.content.res.Configuration.

onContentChanged – вызывается при изменении GUI-интерфейса Activity-компонента при вызове метода setContentView.

onContextItemSelected – вызывается при выборе элемента контекстного меню.

onContextMenuClosed – вызывается при закрытии контекстного меню.

onCreateContextMenu – вызывается при создании контекстного меню – меню, которое открывается при долгом нажатии на GUI-элементе.

onCreateDescription – вызывается перед вызовом метода onPause.

onCreateNavigateUpTaskStack – вызывается при создании стека задач.

onCreateOptionsMenu – вызывается при создании меню опций – меню, которое связано с Activity-компонентом.

onCreatePanelMenu – вызывается для инициализации содержимого меню (меню опций или контекстного меню).

onCreatePanelView – вызывается при создании панели меню.

onCreateThumbnail – вызывается перед вызовом метода onPause и позволяет определить для Activity-компонента значок, а не скриншот.

onCreateView – вызывается для создания фрагментом GUI-интерфейса.

onDetachedFromWindow – вызывается при отсоединении окна Activity-компонента от Window-менеджера.

onGenericMotionEvent – вызывается для необработанного события MotionEvent.

onKeyDown – вызывается для необработанного события KeyEvent при нажатии клавиши.

onKeyLongPress – вызывается для необработанного события KeyEvent при долгом нажатии.

onKeyMultiple – вызывается для необработанного события KeyEvent при многократном нажатии одной клавиши.

onKeyShortcut – вызывается для необработанного события KeyEvent при нажатии комбинации клавиш.

onKeyUp – вызывается для необработанного события KeyEvent при освобождении клавиши.

onLowMemory – вызывается при уменьшении оперативной памяти, когда система будет вынуждена остановить все фоновые процессы, используется для освобождения всех ненужных ресурсов.

onMenuItemSelected – вызывается при выборе элемента меню.

onMenuOpened – вызывается при открытии меню.

onNavigateUp – вызывается при нажатии кнопки Up.

onNavigateUpFromChild – вызывается, если дочерний Activity-компонент использует Up-навигацию.

onNewIntent – при запуске данного Activity-компонента другим Android-компонентом вызывается для уже существующего экземпляра Activity-компонента переднего плана своей задачи, имеющего атрибут android: launchMode=«singleTop» файла манифеста, или если вызывающий Android-компонент использует метод startActivity с флагом FLAG_ACTIVITY_SINGLE_TOP Intent-объекта, вместо создания нового экземпляра Activity-компонента.

onOptionsItemSelected – вызывается при выборе элемента меню опций.

onOptionsMenuClosed – вызывается при закрытии меню опций.

onPanelClosed – вызывается при закрытии панели меню.

onPostCreate – вызывается после вызова метода onRestoreInstanceState.

onPostResume – вызывается после вызова метода onResume.

onPrepareNavigateUpTaskStack – вызывается перед созданием стека задач.

onPrepareOptionsMenu – вызывается перед открытием меню опций.

onPreparePanel – вызывается перед открытием панели меню.

onProvideAssistData – вызывается, когда пользователь запрашивает помощь.

onRestoreInstanceState – вызывается после метода onStart для восстановления состояния Activity-компонента из объекта android. os. Bundle.

onSaveInstanceState – вызывается перед уничтожением Activity-компонента, перемещенного с переднего плана, Android-системой для освобождения ресурсов памяти. Данный метод

предназначен для сохранения состояния Activity-компонента в объекте android. os. Bundle в виде пар имя-значение. Измененный объект Bundle передается Android-системой в методы onCreate (Bundle) и onRestoreInstanceState (Bundle).

onSearchRequested – вызывается при запуске поиска.

onTouchEvent – вызывается для необработанного события MotionEvent при прикосновении к экрану.

onTrackballEvent – вызывается для необработанного события MotionEvent при перемещении указателя.

onTrimMemory – вызывается при сокращении ненужной памяти у процесса.

onUserInteraction – вызывается при взаимодействии с пользователем.

onUserLeaveHint – вызывается, когда Activity-компонент перемещается на задний план в результате действий пользователя.

onWindowAttributesChanged – вызывается при изменении атрибутов окна.

onWindowFocusChanged – вызывается при потере или получении фокуса окном.

onWindowStartingActionMode – вызывается при запуске режима ActionMode для окна.

Другой метод обратного вызова класса android.app.Activity, который рекомендуется переопределять – это метод onPause (), вызываемый при потере фокуса Activity-компонентом и который предназначен для сохранения состояния Activity-компонента, так как Android-приложение не контролирует полностью жизненный цикл своих компонентов – Android-система может уничтожать приостановленные Activity-компоненты для освобождения ресурсов памяти.

В методе onPause () производится сохранение данных, общих для приложения или для использования другими приложениями, с помощью ContentProvider-компонента, или прямое сохранение измененных данных с помощью объекта SharedPreferences (сохранение пар имя-значение примитивных типов данных), метода openFileOutput () класса android.content.Context (сохранение данных во внутреннем хранилище устройства), метода getCacheDir () класса android.content.Context (кэширование данных), метода getExternalStorageDirectory () класса android. os. Environment (сохранение данных в карте памяти), сохранение данных в базе данных SQLite, в Web-сервисах с использованием пакетов java.net.* и android.net.*.

Использование метода onPause () для сохранения состояния Activity-компонента имеет свои преимущества, по сравнению с применением метода onSaveInstanceState (), так как метод onSaveInstanceState () не будет вызываться Android-системой, если Activity-компонент был уничтожен пользователем, например, нажатием клавиши BACK.

Переопределение методов onCreate (), onStart (), onRestart (), onResume (), onPause (), onStop (), onDestroy () и др. должно сопровождаться вызовом суперкласса с помощью ключевого слова super.

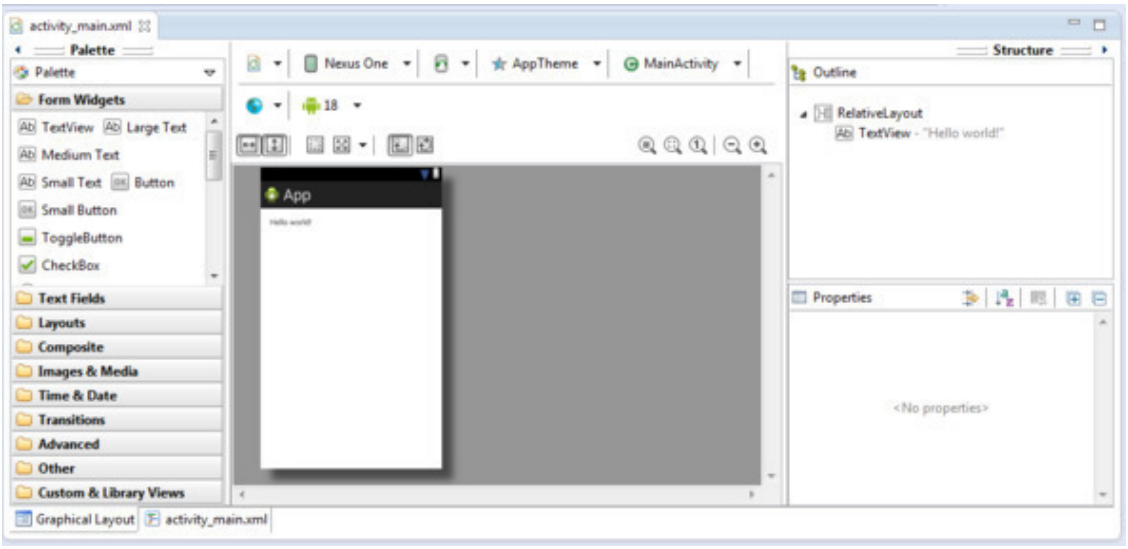
В переопределенном методе onCreate () класса Activity-компонента сгенерированной основы Android-проекта вызывается метод setContentView () класса android.app.Activity, устанавливающий GUI-интерфейс Activity-компонента на основе XML-файла activity_main. xml каталога ресурсов res/layout проекта.

В переопределенном методе onCreateOptionsMenu () класса Activity-компонента сгенерированной основы Android-проекта методом getMenuInflater () получается объект android.view.MenuInflater, отвечающий за создание объектов меню из XML-описания.

Переопределенный метод onOptionsItemSelected () класса Activity-компонента сгенерированной основы Android-проекта представляет каркас обработки выбора элементов меню.

# Layout-редактор ADT-плагина

Для работы с XML-описанием GUI-интерфейса Activity-компонента ADT-плагин предлагает визуальный графический редактор.




Layout-редактор ADT-плагина имеет вкладку **Graphical Layout** для визуального редактирования GUI-интерфейса и XML-вкладку, отображающую код Layout-файла.

XML-код Layout-файла сгенерированной основы Android-проекта Blank Activity определяет GUI-интерфейс, состоящий из RelativeLayout-контейнера, содержащего TextView-компонент.

RelativeLayout-контейнер представлен классом android.widget.RelativeLayout, обеспечивающим компоновку дочерних компонентов android.view.View друг относительно друга и относительно родительского компонента.

Компоновку View-компонентов определяют XML-атрибуты.

Атрибуты android: layout_above и android: layout_below располагают компонент выше или ниже компонента с указанным идентификатором.

Атрибуты android: layout_toLeftOf, android: layout_toStartOf и android: layout_toRightOf, android: layout_toEndOf располагают компонент слева или справа компонента с указанным идентификатором.

Атрибуты android: layout_alignLeft, android: layout_alignStart, android: layout_alignRight, android: layout_alignEnd, android: layout_alignBottom, android: layout_alignTop выравнивают стороны компонента по сторонам компонента с указанным идентификатором.

Атрибут android: layout_alignBaseline выравнивает компонент по базовой линии компонента с указанным идентификатором.

Атрибуты android: layout_alignParentBottom, android: layout_alignParentEnd, android: layout_alignParentTop, android: layout_alignParentStart, android: layout_alignParentLeft, android: layout_alignParentRight располагают компонент внизу, вверху, в левой и в правой части родительского компонента.

Атрибут android: layout_alignWithParentIfMissing со значением true определяет расположение компонента по умолчанию относительно родительского компонента.

Атрибуты android: layout_centerHorizontal, android: layout_centerInParent, android: layout_centerVertical располагают компонент по центральной горизонтальной линии, по центру и по центральной вертикальной линии родительского компонента.

Атрибуты android: layout_marginBottom, android: layout_marginEnd, android: layout_marginLeft, android: layout_marginRight, android: layout_marginStart, android: layout_marginTop определяют отступы компонента.

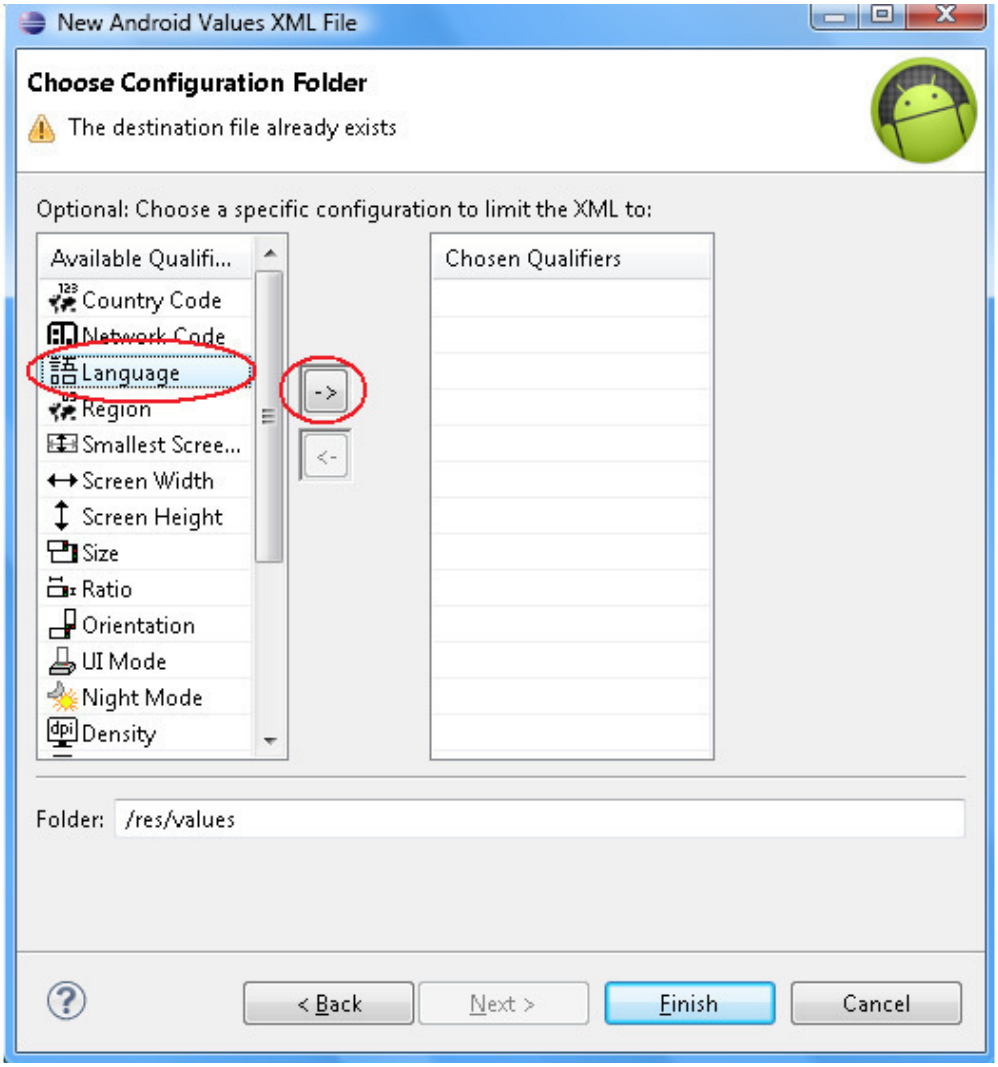
Атрибуты android: layout_height и android: layout_width указывают размеры компонента, при этом константы FILL_PARENT, MATCH_PARENT и WRAP_CONTENT определяют заполнение родительского компонента, соответствие размерам родительского компонента и соответствие содержимому. Данные атрибуты могут принимать значения в виде px (пиксели), dp (виртуальные пиксели, px = dp * (dpi / 160)), sp (масштабируемые пиксели, основанные на предпочтительном размере шрифта), in (дюймы), mm (миллиметры).

TextView-компонент представлен классом android. widget. TextView, обеспечивающий отображение текста пользователю. XML-атрибуты android: layout_height и android: layout_width со значением «wrap_content» устанавливает высоту и ширину компонента, определяемые размером его содержимого. XML-атрибут android: text со значением @string/hello_world устанавливает текстовое содержимое компонента в виде значения строкового ресурса файла strings. xml Android-проекта с именем hello_world.

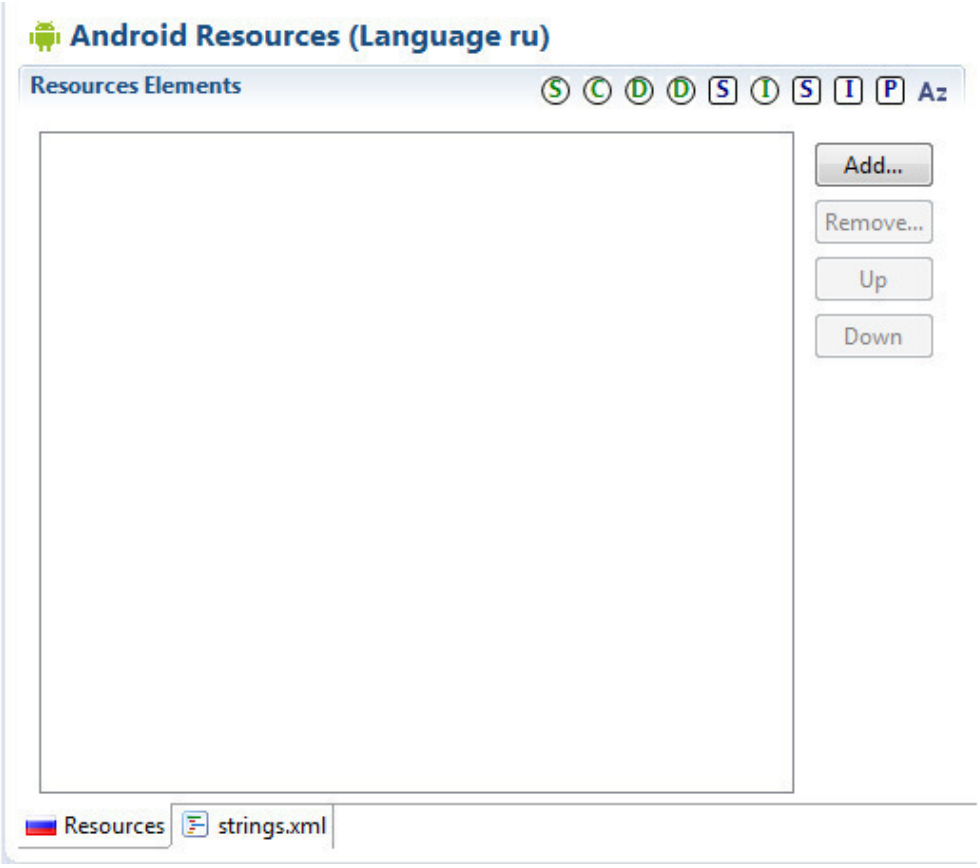
# Интернационализация

Кнопка **Any** вкладки **Graphical Layout** указывает, что данный Android-проект не обеспечивает интернационализацию и локализацию приложения.

Для интернационализации Android-приложения в окне **Package Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML Values File** и нажмем кнопку **Next**. В окне мастера в поле **File:** введем имя файла strings. xml и нажмем кнопку **Next**, в списке **Optional: Choose a specific configuration to limit the XML to:** выберем **Language** и нажмем кнопку **->**. В поле **Language** введем ru и нажмем кнопку **Finish**. В результате в каталоге res проекта будет создана папка values-ru с файлом strings. xml.




Для работы с Values-файлами Android-проекта ADT-плагин также предлагает визуальный графический редактор, имеющий вкладку **Resources** для визуального редактирования и XML-вкладку, отображающую код Values-файла.




Для создания локализованных строк Android-приложения откроем файл strings. xml каталога res/values-ru в редакторе и нажмем кнопку **Add** вкладки **Resources**, в предложенном списке выберем элемент **String** и нажмем кнопку **OK**. В поле **Name** введем имя элемента «hello_world», а в поле **Value** введем строку «Привет!». Еще раз нажмем кнопку **Add** вкладки **Resources**, в предложенном списке выберем элемент **String** и нажмем кнопку **OK**. В поле **Name** введем имя элемента «app_name», а в поле **Value** введем строку «Приложение Андроид». Таким образом, файл strings. xml каталога res/values будет локализован для России.

Откроем файл activity_main. xml каталога ресурсов res/layout Android-проекта и увидим, что кнопка **Locale…** вкладки **Graphical Layout** изменилась на список с элементами **Russian** (**ru)** (русская версия) b **Other** (английская версия), при выборе которых в окне конечного вида GUI-интерфейса вкладки **Graphical Layout** будет отображаться соответствующий текст TextView-компонента.

После инсталляции и запуска Android-приложения в виртуальном устройстве с помощью выбора команды **Run As | Android Application** контекстного меню окна **Package Explorer**, нажмем кнопки **Home** и **Settings** устройства и выберем настройки **Language & keyboard**, в настройке **Select locale** выберем **Русский** – в результате Android-приложение будет отображать GUI-интерфейс в русской версии.

Другой, более быстрый способ интернационализации Android-приложения – это использование команды **Add New Translation** кнопки **Locale…**, открывающей диалоговое окно, в котором список **Language** позволяет выбрать язык локализации, а поля **New Translation** – ввести локализованные значения строковых ключей.

# Панель инструментов Graphical Layout

Кнопка **Android…** вкладки **Graphical Layout** позволяет посмотреть конечный вид GUI-интерфейса относительно установленных версий Android-платформы.


Меню кнопки **Configuration…** вкладки **Graphical Layout** позволяет посмотреть конечный вид GUI-интерфейса для различных типов устройств, различных размеров экрана, различных локализаций, фрагментов и версий. Команда **Manual Previews** в сочетании с командой **Add As Thumbnail** дает возможность сформировать свой список просмотра.

Команда **Create New** кнопки **Configuration…** вкладки **Graphical Layout** обеспечивает создание альтернативных версий файла activity_main. xml описания GUI-интерфейса Activity-компонента для различных конфигураций Android-устройства. При запуске Android-приложения среда выполнения Android-устройства будет загружать подходящий ее конфигурации Layout-файл. Команда **Create New** предлагает следующие спецификаторы Android-конфигураций:

Country Code и Network Code – альтернатива языковой и региональной локализации.

LTR – layout-direction-left-to-right (определитель ldltr) направление письменности слева направо.

sw [n] dp – создает Layout-файл каталога res/layout-sw [n] dp для наименьшего размера из высоты и ширины ndp.

w [n] dp – создает Layout-файл каталога res/layout-w [n] dp для минимальной ширины экрана ndp.

h [n] dp – создает Layout-файл каталога res/layout-h [n] dp для минимальной высоты экрана ndp.

Small, Normal, Large, Xlarge – создает Layout-файл каталога res/layout- [small, normal, large, xlarge] для различных разрешений экрана 320x426, 320x470, 480x640, 720x960.

Long, Not Long – создает Layout-файл каталога res/layout-long и res/layout-notlong для широких экранов WQVGA, WVGA, FWVGA и для экранов QVGA, HVGA, VGA.

Portrait, Landscape – создает Layout-файл каталога res/layout-port и res/layout-land для вертикальной и горизонтальной ориентации экрана.

Not Night, Night – создает Layout-файл каталога res/layout-notnight и res/layout-night для работы в дневное и ночное время.

Low Density, Medium Density, High Density, Extra High Density, TV Density – создает Layout-файл каталога res/layout-ldpi, res/layout-mdpi, res/layout-hdpi, res/layout-xhdpi, res/layout-tvdpi для плотности экрана 120dpi, 160dpi, 240dpi, 320dpi, 213dpi.

Finger – создает Layout-файл каталога res/layout-finger для сенсорного экрана.

Soft – создает Layout-файл каталога res/layout-keyssoft для устройства с виртуальной клавиатурой.

No Keys – создает Layout-файл каталога res/layout-nokeys для устройства без аппаратной клавиатуры.

Hidden, Exposed – создает Layout-файл каталога res/layout-navhidden, res/layout-navexposed для устройства без и с кнопками навигации.

None, Trackball – создает Layout-файл каталога res/layout-nonav, res/layout-trackball для устройства, предоставляющим навигацию только с помощью сенсорного экрана, и для устройства с трекболом.

800x480 – создает Layout-файл каталога res/layout-1280x800 для экрана с разрешением 1280x800.

API 18 – создает Layout-файл каталога res/layout-v18 для устройства с Android-платформой 4.3 и выше.

Меню кнопки обеспечивает просмотр конечного вида GUI-интерфейса для различных типа экранов.

Меню кнопки дает возможность посмотреть конечный вид GUI-интерфейса для вертикальной и горизонтальной ориентации экрана (Portrait и Landscape), в нормальном состоянии, в настольном и автомобильном держателях, при соединении с телевизором, без экрана (Normal, Car Dock, Desk Dock, Television, Appliance), для Android-устройства, работающего в дневное и ночное время (Day Time и Night Time).

Меню кнопки обеспечивает просмотр конечного вида GUI-интерфейса с применением различных стилей для приложения.

Для всего приложения стиль устанавливается с помощью атрибута android: theme="@style/AppTheme» тэга <application> файла манифеста AndroidManifest. xml и ресурса res/values/styles. xml.

Для Activity-компонента стиль устанавливается с помощью атрибута android: theme="@style/ActivityTheme» тэга <activity> файла манифеста AndroidManifest. xml и ресурса res/values/styles. xml.

Применение стиля к Activity-компоненту может существенно менять отображение его GUI-интерфейса на экране Android-устройства. Например, при установке стиля Theme. Dialog, Activity-компонент отображается в виде диалогового окна, не заполняя полностью весь экран.

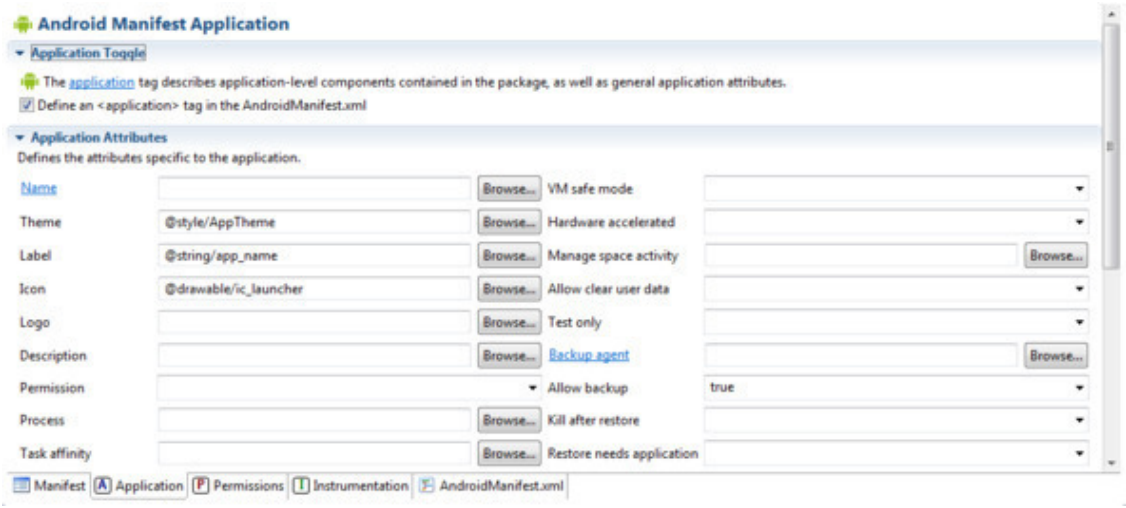
Кнопки вкладки **Graphical Layout**, расположенные ниже панели кнопок с меню, обеспечивают регулировку значений атрибутов android: layout_width, android: layout_height и др. корневого контейнера, а также эмуляцию размера экрана и увеличение-уменьшение изображения экрана.

Palette-палитра вкладки **Graphical Layout** позволяет визуально заполнить GUI-интерфейс Activity-компонента View-компонентами с помощью перетаскивания элементов Palette-палитры в область просмотра конечного вида GUI-интерфейса.

Кроме того, вкладка **Graphical Layout** имеет контекстное меню, открывающееся при нажатии правой кнопкой мышки на View-компоненте в окне просмотра конечного вида GUI-интерфейса, с помощью опций которого можно изменять свойства выбранного View-компонента.

# Редактор файла AndroidManifest. xml ADT-плагина

Для файла манифеста AndroidManifest. xml ADT-плагин также предоставляет визуальный графический редактор.




Редактор файла AndroidManifest. xml ADT-плагина имеет вкладки **Manifest**, **Application**, **Permissions**, **Instrumentation** и **AndroidManifest. xml**.

Набор опций вкладок **Manifest** и **Application** зависит от версии Android-платформы, на основе которой создан Android-проект.

Вкладка **Manifest** ADT-редактора файла AndroidManifest. xml содержит следующие поля и ссылки:

Package – редактирование имени пакета Android-приложения, значение атрибута package элемента <manifest>.

Version Code – редактирование версии Android-приложения, значение атрибута android: versionCode элемента <manifest>.

Version name – редактирование строки, представляющей пользователю версию Android-приложения, значение атрибута android: versionName элемента <manifest>.

Shared user id – если данное приложение является одним из модулей большого Android-приложения, установка данного идентификатора одинаковым для всех модулей с подписанием их одним сертификатом дает взаимный доступ к данным, значение атрибута android: sharedUserId элемента <manifest>.

Shared user label – отображаемая пользователю метка sharedUserId-идентификатора, значение атрибута android: sharedUserLabel элемента <manifest>.

Раздел Manifest Extras – с помощью кнопки **Add** обеспечивает добавление в манифест следующих тэгов:

<uses-sdk> (элемент **Uses Sdk**) – указывает совместимость с версиями Android-платформы.

<supports-screens> (элемент **Supports Screens**) – указывает поддержку Android-приложением различных экранов.

<uses-configuration> (элемент **Uses Configuration**) – указывает, какие опции устройства требуются для работы Android-приложения.

<uses-feature> (элемент **Uses Feature**) – указывает для других Android-приложений, от какой опции устройства зависит работа данного Android-приложения.

<protected-broadcast> (элемент **Protected Broadcast**) – указывает Broadcasts-сообщения, которые может посылать только Android-система.

<compatible-screens> (элемент **Compatible Screens**) – указывает для Android Market совместимость приложения с конфигурациями экрана, используя тэги <screen> (элемент **Screen** кнопки **Add**).

<original-package> (элемент **Original Package**) – предназначен только для системных приложений.

<package-verifier> (элемент **Package Verifier**) – указывает имя пакета приложения, которое вызывается PackageManager-сервисом при инсталляции данного приложения. PackageManager-сервис посылает Broadcast-сообщение ACTION_PACKAGE_NEEDS_VERIFICATION указываемому пакету, который должен содержать BroadcastReceiver-компонент для верификации инсталляции.

Exporting – содержит ссылки **Use the Export Wizard** и **Export an unsigned APK**, запускающие опции экспорта подписанного и неподписанного для публикации Android-приложения.

Links – содержит ссылки Application (открывает вкладку **Application** редактора), Permission (открывает вкладку **Permission** редактора), Instrumentation (открывает вкладку **Instrumentation** редактора), XML Source (открывает вкладку **AndroidManifest. xml** редактора), Documentation (пытается открыть локализованную страницу документации).

Вкладка **Application** ADT-редактора файла AndroidManifest. xml помогает редактировать тэг <application> файла манифеста с помощью разделов **Application Toggle**, **Application Attributes** и **Application Nodes**.

Раздел **Application Toggle** вкладки **Application** содержит ссылку **application** – открывает страницу документации элемента <application> и флажок **Define an <application> tag in the AndroidManifest. xml** – включает элемент <application> в файл манифеста.

Раздел **Application Attributes** вкладки **Application** определяет атрибуты элемента <application> с помощью следующих полей и списков:

Name – при нажатии открывает мастер создания Java-класса, расширяющего класс android. app. Application. Созданный Application-класс указывается в качестве значения атрибута android: name тэга <application>. Если приложение содержит несколько Activity-компонентов, решить проблему обеспечения для них общих глобальных в рамках приложения данных и сервисов поможет Application-класс. При запуске приложения Android-система создаст единственный экземпляр Application-класса и будет вызывать его методы жизненного цикла. Рекомендуется реализовать Application-класс как Singleton-класс со статическим доступом к глобальным данным и сервисам.

Theme – общий для Activity-компонентов стиль, указываемый как значение атрибута android: theme тэга <application>. Предварительно необходимо создать ресурсный файл каталога res/values со стилем, используя команду **New | Other | Android | Android XML Values File** контекстного меню окна **Package Explorer**, дополнить его тэгом <style>, нажать кнопку **Browse** поля **Theme** и выбрать созданный ресурс – в результате у тэга <application> появится атрибут android: theme.

Label – отображаемое пользователю имя приложения, указываемое значением атрибута android: label тэга <application>. Кнопка **Browse** поля **Label** позволяет выбрать значение атрибута в ресурсном файле каталога res/values, содержащем тэги <string>.

Icon – значок приложения, определяемый значением атрибута android: icon тэга <application>. Кнопка **Browse** поля **Icon** позволяет выбрать значение атрибута как имя файла изображения, расположенного в каталоге res/drawable. Папки drawable могут иметь спецификаторы ldpi, mdpi, hdpi, xhdpi, nodpi и tvdpi, обеспечивающие отображение значка на экранах с различной плотностью.

Logo – определяет значение атрибута android: logo тэга <application>, указывающего логотип приложения для отображения в панели ActionBar.

Description – краткое описание приложения, которое указывается значением атрибута android: description тэга <application> и должно определяться ссылкой на строковый ресурс. Кнопка **Browse** поля **Description** позволяет выбрать значение атрибута в ресурсном файле каталога res/values, содержащем тэги <string>.

Permission – список позволяет выбрать разрешение, которое должно иметь стороннее Android-приложение для взаимодействия с данным Android-приложением в целом, указывается значением атрибута android: permission тэга <application>.

Process – определяет значение атрибута android: process тэга <application>, указывающего имя процесса приложения. Если данное приложение является одним из модулей большого Android-приложения, которые имеют одинаковый sharedUserId-идентификатор и подписаны одним сертификатом, тогда установка значения атрибута android: process одинаковым для всех модулей обеспечивает их запуск в одном процессе.

Task affinity – определяет значение атрибута android: taskAffinity тэга <application>, указывающего имя задачи для всех Activity-компонентов приложения, по умолчанию – имя пакета приложения. Task-задача представляет собой набор Activity-компонентов, с которыми пользователь взаимодействует для выполнения своей задачи, при этом Activity-компоненты задачи организуются в обратный стек, в порядке, в котором каждый Activity-компонент был запущен другим Activity-компонентом.

Allow task reparenting – определяет значение атрибута android: allowTaskReparenting тэга <application> – если true, тогда Activity-компоненты приложения могут перемещаться из задачи, которая их запустила, в задачу переднего плана, с которой Activity-компоненты имеют общее taskAffinity-значение, по умолчанию false.

Has code – определяет значение атрибута android: hasCode тэга <application> – если false, тогда приложение не содержит Java-кода, а полностью реализовано на основе программного интерфейса NDK API, по умолчанию true.

Persistent – определяет значение атрибута android: persistent тэга <application> – если true, тогда приложение работает до тех пор, пока работает устройство, обычно используется системными приложениями, по умолчанию false.

Enabled – определяет значение атрибута android: enabled тэга <application> – если false, тогда Android-система не может создавать экземпляры компонентов приложения, по умолчанию true.

Debuggable – определяет значение атрибута android: debuggable тэга <application>. Android-инструменты сборки ADT-плагина автоматически добавляют значение атрибута true в отладочном режиме и удаляют данный атрибут, имеющий по умолчанию значение false, при экспорте релиза приложения.

Vm safe mode – определяет значение атрибута android: vmSafeMode тэга <application> – если true, тогда JIT-оптимизация отключается.

Hardware accelerated – определяет значение атрибута android: hardwareAccelerated тэга <application> – если true, тогда включается аппаратное ускорение визуализации, по умолчанию false.

Manage space activity – определяет значение атрибута android: manageSpaceActivity тэга <application>, указывает имя Activity-компонента, который запускается дополнительной кнопкой **Управление местом** в разделе настроек **Приложения | Управление приложениями** Android-устройства.

Allow clear user data – определяет значение атрибута android: allowClearUserData тэга <application> – применимо только для системных приложений, для обычных приложений игнорируется.

Test only – определяет значение атрибута android: testOnly тэга <application> – если true, тогда приложение находится в стадии тестирования и не может быть инсталлировано в Android-устройстве.

Backup agent – определяет значение атрибута android: backupAgent тэга <application>, указывает имя класса, расширяющего класс android.app.backup.BackupAgent, который вызывается сервисом Backup Manager для определения настроек приложения, сохраняемых в облачном хранилище, и их восстановления при реинсталляции приложения в случае обновления Android-системы устройства.

Allow backup – определяет значение атрибута android: allowBackup тэга <application> – если false, тогда приложение не обслуживается сервисом Backup Manager, по умолчанию true.

Kill after restore – определяет значение атрибута android: killAfterRestore тэга <application> – используется системными приложениями.

Restore needs application – определяет значение атрибута android: restoreNeedsApplication тэга <application> – используется системными приложениями.

Restore any version – определяет значение атрибута android: restoreAnyVersion тэга <application> – если true, тогда сервис Backup Manager будет восстанавливать приложение даже в том случае, если версии облачного хранилища и текущей инсталляции не совпадают, по умолчанию false.

Never encrypt – определяет значение атрибута android: neverEncrypt тэга <application> – если true, тогда приложение отказывается от защиты хранения своих данных.

Large heap – определяет значение атрибута android: largeHeap тэга <application> – если true, тогда приложению может понадобиться расширение размера кучи.

Cant save state – определяет значение атрибута android: cantSaveState тэга <application> – если true, тогда приложение является ресурсоемким и отказывается участвовать в сохранении-восстановлении Android-системой своего состояния. При таком работающем приложении, если пользователь пытается загрузить другое приложение, он запрашивается на выход из первого приложения.

Ui options – определяет значение атрибута android: uiOptions тэга <application>, указывающее дополнительные опции GUI-интерфейса Activity-компонентов приложения с помощью двух значений: none (по умолчанию, нет дополнительных опций) и splitActionBarWhenNarrow (добавляет панель ActionBar, разделенную на секцию навигации и панель действий).

Supports rtl – определяет значение атрибута android: supportsRtl тэга <application> – если true, тогда приложение поддерживает right-to-left (RTL) письменность справа налево.

Раздел **Application Nodes** вкладки **Application** кнопкой **Add** обеспечивает добавление в тэг <application> тэгов <activity> (элемент **Activity**), <activity-alias> (элемент **Activity Alias**), <meta-data> (элемент **Meta Data**), <provider> (элемент **Provider**), <receiver> (элемент **Receiver**), <service> (элемент **Service**), <uses-library> (элемент **Uses Library**).

Кнопка **Add** позволяет добавлять в тэги <activity>, <receiver> и <service> тэги <intent-filter> (элемент **Intent Filter**) и <meta-data> (элемент **Meta Data**), при этом в тэг <intent-filter> могут добавляться кнопкой **Add** тэги <action> (элемент **Action**), <category> (элемент **Category**), <data> (элемент **Data**).

В тэг <provider> кнопка **Add** добавляет тэги <grant-uri-permission> (элемент **Grant Uri Permission**), <meta-data> (элемент **Meta Data**), <path-permission> (элемент **Path Permission**).

Тэг <activity> (элемент **Activity**) описывает Activity-компонент приложения (класс, расширяющий класс android.app.Activity). При выборе элемента **Activity** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Activity**, позволяющий определить атрибуты тэга <activity> с помощью следующих полей и списков:

Name – при нажатии открывает мастер создания Java-класса, расширяющего класс android.app.Activity. Созданный Activity-компонент указывается в качестве значения атрибута android: name.

Theme – определяет для Activity-компонента стиль, указываемый как значение атрибута android: theme.

Label – отображаемая пользователю метка Activity-компонента, указываемая значением атрибута android: label.

Icon – значок Activity-компонента, определяемый значением атрибута android: icon.

Logo – определяет значение атрибута android: logo, указывающего логотип приложения для отображения в панели ActionBar.

Launch mode – список позволяет выбрать значение атрибута android: launchMode, определяющего загрузку Activity-компонента при получении вызывающего Intent-объекта:

standart (по умолчанию) – Android-система всегда создает новый экземпляр Activity-компонента в целевой задаче и передает ему Intent-объект.

singleTop – если экземпляр Activity-компонента уже существует на переднем плане целевой задачи, вызывается метод onNewIntent () уже существующего экземпляра, вместо создания нового экземпляра Activity-компонента.

singleTask – Android-система создает новый экземпляр Activity-компонента в новой задаче и передает ему Intent-объект. Если экземпляр Activity-компонента уже существует, тогда вызывается его метод onNewIntent (), вместо создания нового экземпляра Activity-компонента.

singleInstance – работает аналогично singleTask, за исключением того, что задача может содержать только один Activity-компонент.

Screen orientation – список позволяет выбрать значение атрибута android: screenOrientation, определяющего ориентацию отображения Activity-компонента на экране:

unspecified (по умолчанию) – ориентацию выбирает Android-система.

user – ориентация определяется пользовательскими предпочтениями.

behind – ориентация такая же, как и у предыдущего Activity-компонента.

landscape – альбомная (горизонтальная) ориентация.

portrait – портретная (вертикальная) ориентация.

reverseLandscape – альбомная (горизонтальная) ориентация в противоположном направлении.

reversePortrait – портретная (вертикальная) ориентация в противоположном направлении.

sensorLandscape – альбомная (горизонтальная) ориентация, направление которой определяется Android-системой на основе сенсора.

sensorPortrait – портретная (вертикальная) ориентация, направление которой определяется Android-системой на основе сенсора.

sensor – ориентация определяется Android-системой на основе сенсора.

fullSensor – ориентация определяется Android-системой на основе сенсора с возможностью ориентаций landscape, portrait, reverseLandscape и reversePortrait.

nosensor – сенсор устройства игнорируется.

Config changes – кнопка **Select** позволяет выбрать значение атрибута android: configChanges, определяющего изменения конфигурации, при которых Activity-компонент не перезапускается, а вызывается его метод onConfigurationChanged ():

mcc – изменение MCC-кода страны.

mnc – изменение MNC-кода сети.

locale – изменение локализации устройства.

touchscreen – изменение сенсорного экрана.

keyboard – изменение типа клавиатуры устройства.

keyboardHidden – изменение доступности клавиатуры.

navigation – изменение механизма навигации устройства.

screenLayout – изменение компоновки экрана.

fontScale – изменение размера шрифта.

uiMode – изменение состояния устройства (устройство помещено в держатель).

orientation – изменилась ориентация экрана.

screenSize – при изменении ориентации экрана изменились пропорции экрана.

smallestScreenSize – при подключении устройства к внешнему дисплею изменился размер экрана.

Permission – список позволяет выбрать разрешение, которое должно иметь стороннее Android-приложение для вызова Activity-компонента, указывается значением атрибута android: permission.

Multiprocess – определяет значение атрибута android: multiprocess – если true, тогда Activity-компонент запускается в том же процессе, что и вызвавший его Android-компонент.

Process – определяет значение атрибута android: process, указывающего имя процесса, в котором запускается Activity-компонент.

Task affinity – определяет значение атрибута android: taskAffinity, указывающего имя задачи, в которой запускается Activity-компонент с флагом FLAG_ACTIVITY_NEW_TASK.

Allow task reparenting – определяет значение атрибута android: allowTaskReparenting – если true, тогда Activity-компонент может перемещаться из задачи, которая его запустила, в задачу переднего плана, с которой Activity-компонент имеет общее taskAffinity-значение, по умолчанию false.

Finish on task launch – определяет значение атрибута android: finishOnTaskLaunch – если true, тогда существующий экземпляр Activity-компонента уничтожается, если пользователь снова запускает его задачу, по умолчанию false.

Finish on close system dialogs – определяет значение атрибута android: finishOnCloseSystemDialogs – если true, тогда Activity-компонент уничтожается при закрытии текущего окна, например при нажатии кнопки HOME или при блокировке устройства.

Clear task on launch – определяет значение атрибута android: clearTaskOnLaunch – если true, тогда при перезапуске задачи из домашнего экрана, задача очищается от всех Activity-компонентов до данного корневого Activity-компонента, по умолчанию false.

No history – определяет значение атрибута android: noHistory – если true, тогда Activity-компонент удаляется из стека задачи и уничтожается, когда становится невидимым на экране, по умолчанию false.

Always retain task state – определяет значение атрибута android: alwaysRetainTaskState – если true, тогда Android-система не очищает задачу данного корневого Activity-компонента, а сохраняет ее последнее состояние, по умолчанию false.

State not need – определяет значение атрибута android: stateNotNeeded – если true, тогда метод onSaveInstanceState () Activity-компонента не вызывается, а его метод onCreate () в качестве аргумента всегда получает null, по умолчанию false.

Exclude from recents – определяет значение атрибута android: excludeFromRecents – если true, тогда Activity-компонент не появляется в списке недавно запущенных Activity-компонентов, который отображается при долгом нажатии на кнопку HOME устройства, по умолчанию false.

Enabled – определяет значение атрибута android: enabled – если false, тогда Android-система не может создавать экземпляры Activity-компонента, по умолчанию true.

Exported – определяет значение атрибута android: exported – если true, тогда Activity-компонент может запускаться другими Android-приложениями, если false, тогда Activity-компо-

нент может запускаться только Android-компонентами своего приложения или другими модулями с общим sharedUserId-идентификатором.

Window soft input mode – кнопка **Select** позволяет выбрать значение атрибута android: windowSoftInputMode, определяющего как окно Activity-компонента взаимодействует с окном экранной клавиатуры:

stateUnspecified (по умолчанию) – состояние видимости или нет экранной клавиатуры выбирает Android-система.

stateUnchanged – экранная клавиатура сохраняет свое последнее состояние.

stateHidden – экранная клавиатура скрыта когда пользователь переходит вперед к Activity-компоненту.

stateAlwaysHidden – экранная клавиатура всегда скрыта.

stateVisible – экранная клавиатура появляется когда пользователь переходит вперед к Activity-компоненту.

stateAlwaysVisible – экранная клавиатура всегда появляется.

adjustUnspecified (по умолчанию) – будет окно Activity-компонента изменять свои размеры и включать в себя окно экранной клавиатуры или экранная клавиатура будет накладываться на окно Activity-компонента с его панорамированием определяет Android-система.

adjustResize – окно Activity-компонента изменяет свои размеры и включает в себя окно экранной клавиатуры.

adjustPan – экранная клавиатура накладывается на окно Activity-компонента, которое панорамируется на ввод.

adjustNothing – окно Activity-компонента не изменяет свои размеры и не панорамируется.

Immersive – определяет значение атрибута android: immersive – если true, тогда Activity-компонент не прерывается другими Activity-компонентами и уведомлениями.

Hardware accelerated – определяет значение атрибута android: hardwareAccelerated – если true, тогда включается аппаратное ускорение визуализации, по умолчанию false.

Ui options – определяет значение атрибута android: uiOptions, указывающее дополнительные опции GUI-интерфейса Activity-компонента с помощью двух значений: none (по умолчанию, нет дополнительных опций) и splitActionBarWhenNarrow (добавляет панель ActionBar, разделенную на секцию навигации и панель действий).

Parent activity name – определяет значение атрибута android: parentActivityName, указывающее имя класса Activity-компонента, являющегося логическим родителем данному Activity-компоненту и к которому будет осуществляться переход с помощью кнопки Up.

Тэг <intent-filter> (элемент **Intent Filter**) обеспечивает создание объекта android.content.IntentFilter, который указывает Android-системе какие неявные (не указывающие целевой класс) объекты android.content.Intent может обрабатывать Android-компонент. При выборе элемента **Intent Filter** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Intent Filter**, позволяющий определить атрибуты тэга <intent-filter> с помощью следующих полей:

Label – определяет значение атрибута android: label, указывающего отображаемую пользователю метку Android-компонента, запущенного соответствующим фильтру Intent-объектом.

Icon – определяет значение атрибута android: icon, указывающего значок Android-компонента, запущенного соответствующим фильтру Intent-объектом.

Logo – определяет значение атрибута android: logo, указывающего логотип панели ActionBar Android-компонента, запущенного соответствующим фильтру Intent-объектом.

Priority – определяет значение атрибута android: priority, указывающего приоритет обработки соответствующих фильтру Intent-объектов для случая, когда несколько Android-компонентов соответствуют Intent-объекту.

Дочерний тэг <action> (элемент **Action**) тэга <intent-filter> указывает действие Intent-объекта, поддерживаемое Android-компонентом. При выборе элемента **Action** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Action**, позволяющий определить атрибут тэга <action> с помощью списка **Name**, обеспечивающего выбор действия android.intent.action.* как значения атрибута android: name.

Дочерний тэг <category> (элемент **Category**) тэга <intent-filter> указывает, к какому типу принадлежит Android-компонент, чтобы соответствовать категории Intent-объекта. При выборе элемента **Category** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Category**, позволяющий определить атрибут тэга <category> с помощью списка **Name**, обеспечивающего выбор категории android.intent.category.* как значения атрибута android: name.

Дочерний тэг <data> (элемент **Data**) тэга <intent-filter> описывает, какие данные могут быть переданы Intent-объектом Android-компоненту. При выборе элемента **Data** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Data**, позволяющий определить атрибуты тэга <data> с помощью полей **Mime type** (атрибут android: mimeType указывает MIME-тип данных Intent-объекта), **Scheme**, **Host**, **Port**, **Path**, **Path prefix**, **Path pattern** (URI-адрес данных в формате scheme://host: port/path, атрибуты android: scheme, android: host, android: port, android: path, android: pathPrefix, android: pathPattern).

Тэг <meta-data> (элемент **Meta Data**) позволяет добавить дополнительные данные к Android-компоненту, доступ к которым можно получить программным способом:

ApplicationInfo ai = getPackageManager().getApplicationInfo(activity.getPackageName (), PackageManager.GET_META_DATA);

Bundle bundle = ai.metaData;

String myValue = bundle.getString («myKey»);

При выборе элемента **Meta Data** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Meta Data**, позволяющий определить атрибуты тэга <meta-data> с помощью полей **Name** (атрибут android: name определяет имя элемента метаданных), **Value** (атрибут android: value определяет значение элемента метаданных), **Resource** (атрибут android: resource указывает ссылку на ресурс).

Тэг <activity-alias> (элемент **Activity Alias**) обеспечивает запуск целевого Activity-компонента под другим именем, меткой, с другим Intent-фильтром. При выборе элемента **Activity Alias** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Activity Alias**, позволяющий определить атрибуты тэга <activity-alias> с помощью полей и списков:

Name (атрибут android: name указывает псевдоним для целевого Activity-компонента),

Target activity (атрибут android: targetActivity указывает имя целевого Activity-компонента),

Label (атрибут android: label определяет метку псевдонима),

Description (атрибут android: description определяет описание псевдонима),

Icon (атрибут android: icon указывает значок псевдонима),

Logo (атрибут android: logo определяет логотип панели ActionBar),

Permission (атрибут android: permission указывает разрешение, которое должно иметь стороннее Android-приложение для вызова Activity-компонента через псевдоним),

Enabled (атрибут android: enabled указывает возможность создания экземпляра целевого Activity-компонента через псевдоним),

Exported (атрибут android: exported указывает возможность запуска целевого Activity-компонента сторонними Android-приложениями через псевдоним).

Parent activity name – определяет значение атрибута android: parentActivityName, указывающее имя класса Activity-компонента, являющегося логическим родителем данному Activity-компоненту и к которому будет осуществляться переход с помощью кнопки Up.

Тэг <provider> (элемент **Provider**) описывает ContentProvider-компонент приложения (класс, расширяющий класс android.content.ContentProvider), обеспечивающий управление данными приложения. При выборе элемента **Provider** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Provider** с полями и списками, позволяющий определить атрибуты тэга <provider>.

Поле со ссылкой **Name** при нажатии открывает мастер создания Java-класса, расширяющего класс android.content.ContentProvider. Созданный ContentProvider-компонент указывается в качестве значения атрибута android: name.

Поля **Label**, **Description**, **Icon**, **Logo**, **Process**, **Permission**, **Multiprocess**, **Enabled**, **Exported** элемента **Provider** работают аналогично соответствующим полям элемента **Activity** раздела **Application Nodes** вкладки **Application**.

Поле **Authorities** элемента **Provider** определяет значение атрибута android: authorities тэга <provider>, указывающего один или несколько URI-адресов, идентифицирующих для Android-системы ContentProvider-компонент.

Список **Syncable** определяет значение атрибута android: syncable тэга <provider> – если true, тогда данные ContentProvider-компонента синхронизированы с данными сервера.

Поле **Read permission** и поле **Write permission** определяют значения атрибутов android: readPermission и android: writePermission, указывающих разрешения, необходимые для чтения и изменения данных ContentProvider-компонента.

Поле **Grand URI permissions** определяет значение атрибута android: grantUriPermissions – если true, тогда приложению, вызывающему ContentProvider-компонент Intent-объектом с флагами FLAG_GRANT_READ_URI_PERMISSION и FLAG_GRANT_WRITE_URI_PERMISSION, предоставляется одноразовый доступ к данным.

Поле **Init order** определяет значение атрибута android: initOrder, указывающего номер в очереди инициализации ContentProvider-компонентов приложения.

Дочерний тэг <grant-uri-permission> (элемент **Grant Uri Permission**) тэга <provider> указывает URI-адрес ContentProvider-компонента, к которому может быть дан одноразовый доступ стороннему приложению, с помощью полей **Path**, **Path prefix** и **Path pattern**, определяющих значения атрибутов android: path, android: pathPrefix и android: pathPattern.

Дочерний тэг <path-permission> (элемент **Path Permission**) тэга <provider> указывает для URI-адреса ContentProvider-компонента разрешения доступа к его данным сторонним приложениям, используя поля **Path**, **Path prefix, Path pattern**, **Permission**, **Read permission**, **Write permission**, определяющих значения атрибутов android: path, android: pathPrefix, android: pathPattern, android: permission, android: readPermission и android: writePermission.

Тэг <receiver> (элемент **Receiver**) описывает BroadcastReceiver-компонент приложения (класс, расширяющий класс android.content.BroadcastReceiver), позволяющий обрабатывать Intent-объекты, посылаемые широковещательным способом Android-системой или другими приложениями. При выборе элемента **Receiver** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Receiver** с полями и списками, позволяющий определить атрибуты тэга <receiver>. Поле со ссылкой **Name** при нажатии открывает мастер создания Java-класса, расширяющего класс android.content.BroadcastReceiver. Созданный BroadcastReceiver-компонент указывается в качестве значения атрибута android: name.

Поля **Label**, **Description**, **Icon**, **Logo**, **Process**, **Permission**, **Enabled**, **Exported** элемента **Receiver** работают аналогично соответствующим полям элемента **Activity** раздела **Application Nodes** вкладки **Application**.

Тэг <service> (элемент **Service**) описывает Service-компонент приложения (класс, расширяющий класс android.app.Service), предназначенный для выполнения продолжительных операций без предоставления GUI-интерфейса. При выборе элемента **Service** кнопкой **Add**,

во вкладке **Application** появляется раздел **Attributes for Service** с полями и списками, позволяющий определить атрибуты тэга <service>. Поле со ссылкой **Name** при нажатии открывает мастер создания Java-класса, расширяющего класс android.app.Service. Созданный Service-компонент указывается в качестве значения атрибута android: name. Поля **Label**, **Description**, **Icon**, **Logo**, **Process**, **Permission**, **Enabled**, **Exported** элемента **Service** работают аналогично соответствующим полям элемента **Activity** раздела **Application Nodes** вкладки **Application**.

Список **Stop with task** элемента **Service** определяет значение атрибута android: stopWithTask тэга <service> – если true, тогда сервис автоматически завершит свою работу при удалении пользователем задачи приложения, по умолчанию false.

Список **Isolated process** элемента **Service** определяет значение атрибута android: isolatedProcess – если true, тогда сервис будет работать в изолированном процессе, не имеющим те разрешения, которые даны остальной части приложения.

Тэг <uses-library> (элемент **Uses Library**) указывает Android-библиотеку, которая требуется для работы приложения. При выборе элемента **Uses Library** кнопкой **Add**, во вкладке **Application** появляется раздел **Attributes for Uses Library** с полями и списками, позволяющий определить атрибуты тэга <uses-library>. Поле Name определяет значение атрибута android: name, указывающего имя Android-библиотеки, с которой связано приложение, а список Required – значение атрибута android: required – если true (по умолчанию), тогда приложение не может работать и быть инсталлированным без наличия указанной библиотеки в устройстве.

Вкладка **Permissions** ADT-редактора файла AndroidManifest. xml с помощью кнопки **Add** обеспечивает добавление в тэг <manifest> тэгов <permission> (элемент **Permission**), <permission-group> (элемент **Permission Group**), <permission-tree> (элемент **Permission Tree**), <uses-permission> (элемент **Uses Permission**).

Тэг <permission> (элемент **Permission**) позволяет объявить пользовательское разрешение, которое должно получить стороннее приложение для доступа к Android-компонентам данного приложения. При выборе элемента **Permission** кнопкой **Add**, во вкладке **Permissions** появляется раздел **Attributes for Permission** с полями и списками, позволяющими определить атрибуты тэга <permission>.

Поля **Name**, **Label**, **Description**, **Icon** и **Logo** определяют значения атрибутов android: name, android: label, android: description, android: icon и android: logo, указывающих имя, метку, описание, значок и логотип пользовательского разрешения.

Поле **Permission group** определяет значение атрибута android: permissionGroup, указывающего группу разрешений, к которой относится данное разрешение.

Список **Protection level** определяет значение атрибута android: protectionLevel, указывающего уровень риска, который несет данное разрешение:

normal – минимальный риск для других приложений, Android-системы, пользователя.

dangerous – может причинить вред пользователю, например, разрешает доступ к данным пользователя.

signature – Android-система даст данное разрешение запрашивающему его приложению, только если запрашивающее разрешение приложение подписано тем же сертификатом, что и данное приложение, которое объявило пользовательское разрешение.

signatureOrSystem – используется только для системных приложений или приложений, подписанных тем же сертификатом, что и приложение, которое объявило пользовательское разрешение.

system – используется только для системных приложений.

development – разрешения даются только при разработке, но не при инсталляции.

Тэг <permission-group> (элемент **Permission Group**) объявляет группу пользовательских разрешений. При выборе элемента **Permission Group** кнопкой **Add**, во вкладке **Permissions**

появляется раздел **Attributes for Permission Group** с полями и списками, позволяющими определить атрибуты тэга <permission-group>.

Поля **Name**, **Label**, **Description**, **Icon** и **Logo** определяют значения атрибутов android: name, android: label, android: description, android: icon и android: logo, указывающих имя, метку, описание, значок и логотип группы пользовательских разрешений.

Поле **Priority** определяет значение атрибута android: priority, указывающего приоритет обработки Intent-объекта.

Тэг <permission-tree> (элемент **Permission Tree**) объявляет базовое имя дерева разрешений, которые могут быть добавлены программным способом с помощью метода addPermission () класса android.content.pm.PackageManager. При выборе элемента **Permission Tree** кнопкой **Add**, во вкладке **Permissions** появляется раздел **Attributes for Permission Tree** с полями, позволяющими определить атрибуты тэга <permission-tree>. Поля **Name**, **Label**, **Icon** и **Logo** определяют значения атрибутов android: name, android: label, android: icon и android: logo, указывающих базовое имя, метку, значок и логотип дерева динамически добавляемых разрешений.

Тэг <uses-permission> (элемент **Uses Permission**) обеспечивает при инсталляции приложения запрос на предоставление ему определенного разрешения, которое указывается атрибутом android: name и может быть выбрано с помощью списка **Name** раздела **Attributes for Uses Permission** вкладки **Permissions**.

Вкладка **Instrumentation** ADT-редактора файла AndroidManifest. xml с помощью кнопки **Add** обеспечивает добавление в тэг <manifest> тэга <instrumentation>, который используется в файле манифеста проекта Android-тестирования (основа проекта Android-тестирования создается с помощью мастера **Android Test Project**).

При открытии в ADT-редакторе специфических для Android-разработки файлов, таких как activity_main. xml, strings. xml и AndroidManifest. xml, в меню **Refactor** Workbench-окна появляется подменю **Android**, содержащее опции Android-рефакторинга.

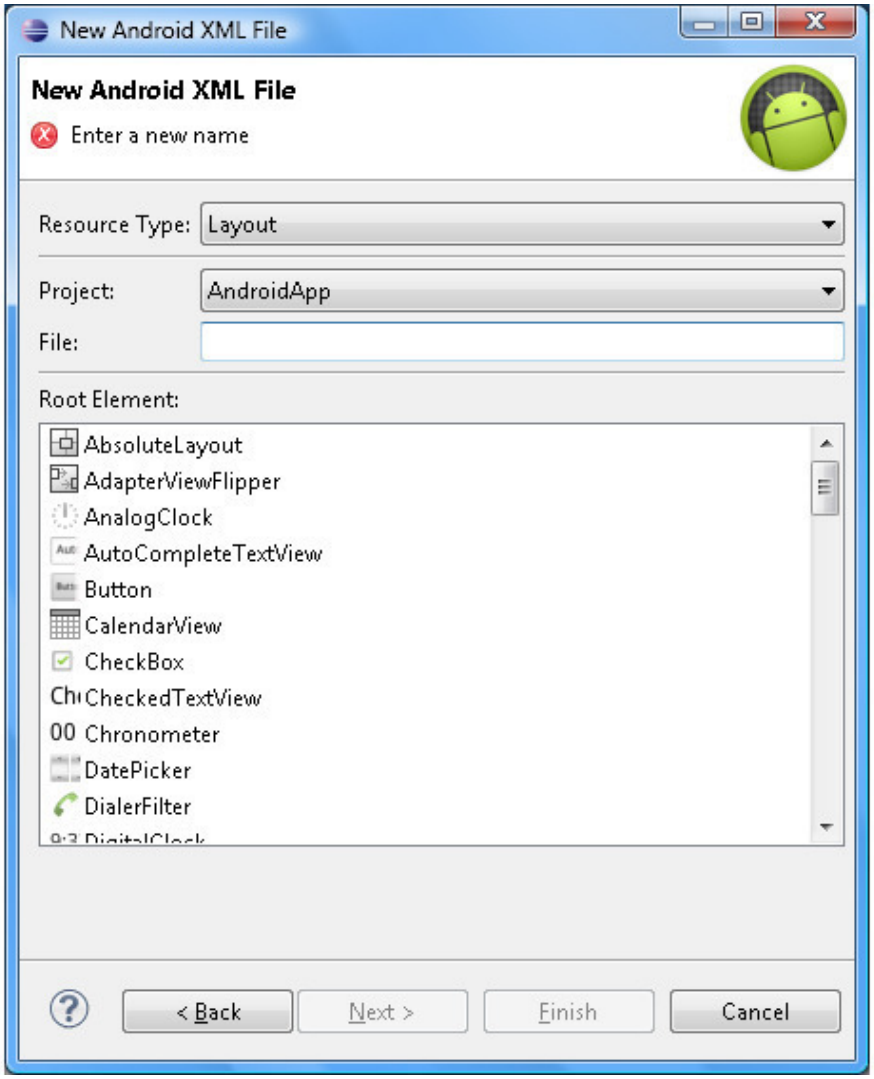
# Мастер Android XML File

Мастер **Android XML File**, доступный в разделе **Android** команды **New**, обеспечивает создание набора ресурсов Android-приложения, состоящего из:

XML-описаний GUI-интерфейса Activity-компонентов (тип ресурса Layout),
различного рода значений, используемых приложением (тип ресурса Values),
графики (тип ресурса Drawable),
меню приложения (тип ресурса Menu),
наборов цветов (тип ресурса Color List),
анимации (тип ресурса Property Animation и Tween Animation),
метаданных приложения App Widgets (тип ресурса AppWidget Provider),
GUI-интерфейса PreferenceActivity-операции (тип ресурса Preference),
настроек поиска (тип ресурса Searchable).

# Тип ресурса Layout

Для создания Layout-файла Android-приложения в окне **Project Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML File** или **Android XML Layout File**, нажмем кнопку **Next** – в результате откроется окно мастера создания Layout-файла, в списке **Resource Type** которого выбран тип **Layout**.




Поле **File:** мастера создания Layout-файла предлагает ввести имя нового файла XML-описания GUI-интерфейса, который затем с расширением. xml появится в каталоге res/layout Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса R.layout. [имя Layout-файла] или в XML-коде с помощью ссылки @ [package: ] layout/ [имя Layout-файла].

Раздел **Root Element:** мастера создания Layout-файла предлагает выбрать корневой View-компонент GUI-интерфейса, который может быть как контейнером для других GUI-компонентов, так и отдельным GUI-компонентом.

В качестве контейнера обычно используются ViewGroup-компоненты LinearLayout (компоновка в столбец или строку), RelativeLayout (якорная компоновка) и FrameLayout (стековая

компоновка), а индивидуальные GUI-компоненты представлены такими View-компонентами как кнопка, флажок, переключатель, текстовая область и др.

Помимо контейнера и индивидуального GUI-компонента корневым элементом Layout-файла может служить элемент <merge>, который предназначен для создания Layout-файла, включаемого в другой Layout-файл с помощью тэга <include>. Тэг индивидуального GUI-компонента может также содержать тэг <requestFocus>, дающий первоначальный фокус View-компоненту.

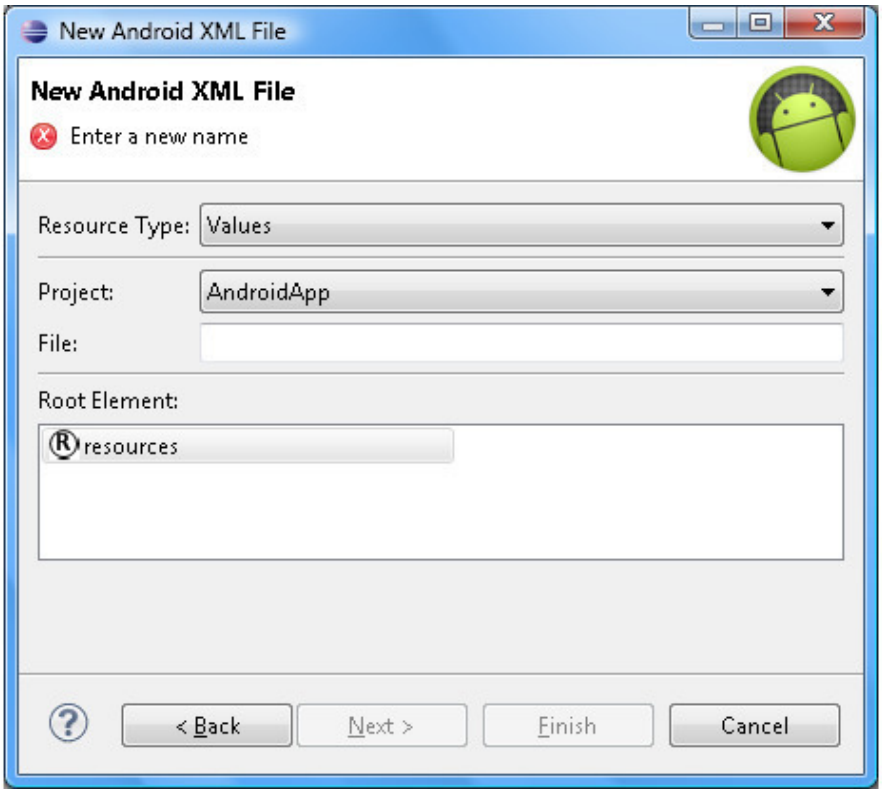
Тэг <fragment>, начиная с версии Android 3.0 (API level 11), обеспечивает модульность GUI-интерфейса Activity-компонента, описывая его часть, которая имеет свой жизненный цикл, свое взаимодействие с пользователем и с другими компонентами приложения и которая может добавляться или удаляться во время работы родительского Activity-компонента. Тэг <fragment> не может быть корневым элементом Layout-файла, а включается в основной Layout-файл в качестве дочернего тэга контейнера LinearLayout, RelativeLayout или FrameLayout. При этом атрибут android: name указывает имя класса фрагмента, расширяющего класс android.app.Fragment или android.support.v4.app.Fragment.

После ввода имени нового Layout-файла, выбора его корневого элемента и нажатия кнопки **Next**, появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки layout, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка layout с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового Layout-файла он будет открыт в Layout-редакторе ADT-плагина, обеспечивающим визуальное редактирование GUI-интерфейса.

# Тип ресурса Values

Для создания ресурсного файла Android-приложения в окне **Project Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML File** или **Android XML Values File**, нажмем кнопку **Next** – в результате откроется окно мастера, в списке **Resource Type** которого выберем тип **Values**.




Поле **File:** мастера создания ресурсного файла предлагает ввести имя нового файла XML-описания значений приложения, который затем с расширением. xml появится в каталоге res/values Android-проекта и будет доступен в Java-коде с помощью сгенерированного R-класса или в XML-коде с помощью ссылки на имя ресурса.

Раздел **Root Element:** мастера создания ресурсного файла показывает, что корневым элементом XML-файла служит тэг <resources>.

После ввода имени нового ресурсного файла и нажатия кнопки **Next**, появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки values, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка values с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового ресурсного файла он будет открыт в редакторе Values-файлов ADT-плагина. Кнопка **Add** вкладки **Resources** редактора Values-файлов обеспечивает добавление в корневой тэг <resources> ресурсного файла тэгов <color> (элемент **Color**), <dimen> (элемент **Dimension**), <drawable> (элемент **Drawable**), <integer-array> (элемент **Integer Array**), <item> (элемент **Item**), <plurals> (элемент **Quantity Strings (Plurals)**), <string> (элемент **String**), <string-array> (элемент **String Array**), <style> (элемент **Style/Theme**).

Тэг <color> (элемент **Color**) определяет цвет, используя синтаксис <color name=«color_name»> hex_color </color>, где hex_color – значение цвета в формате #RGB,

#ARGB, #RRGGBB, #AARRGGBB. При добавлении элемента **Color** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for Color** с полем **Name** (определяет значение атрибута name) и полем **Value** (определяет значение цвета). Тэг <color> как правило используется в ресурсном файле с именем colors. xml каталога res/values. Именованный ресурс цвета может использоваться для определения цвета различных объектов, таких как Drawable или TextView. Созданный ресурс доступен в

Java-коде с помощью сгенерированного класса R.color.color_name, или в XML-коде – с помощью ссылки @ [package: ] color/color_name.

Тэг <dimen> (элемент **Dimension**) определяет величину измерения, используя синтаксис <dimen name=«dimension_name»> dimension </dimen>, где dimension – значение в формате dp (независимый от плотности пиксель), sp (независимый от масштаба пиксель), pt (точка, 1/72 дюйма), px (пиксель), mm (миллиметр), in (дюйм). При добавлении элемента **Dimension** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for Dimension** с полем **Name** (определяет значение атрибута name) и полем **Value** (определяет значение). Созданный ресурс доступен в

Java-коде с помощью сгенерированного класса R. dimen. dimension_name, или в XML-коде – с помощью ссылки @ [package: ] dimen/dimension_name.

Тэг <drawable> (элемент **Drawable**) обеспечивает создание объекта android.graphics.drawable.PaintDrawable, представляющего прямоугольник, заполненный цветом, используя синтаксис <drawable name=color_name> color_value </drawable>, где color_value – значение цвета в формате #RGB, #ARGB, #RRGGBB, #AARRGGBB. При добавлении элемента **Drawable** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for Drawable** с полем **Name** (определяет значение атрибута name) и полем **Value** (определяет значение). Созданный ресурс доступен в Java-коде с помощью сгенерированного класса R. drawable. drawable_name, или в XML-коде – с помощью ссылки @ [package: ] drawable/ drawable_name.

Тэг <integer-array> (элемент **Integer Array**) определяет массив целых чисел, используя синтаксис <integer-array name=«integer_array_name»> <item> integer </item> </integer-array>. При добавлении элемента **Integer Array** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for Integer Array** с полем **Name** – определяет значение атрибута name. Созданный ресурс доступен в Java-коде с помощью сгенерированного класса R.array.integer_array_name или в XML-коде с помощью ссылки @ [package: ] array/ integer_array_name.

Тэг <item> (элемент **Item**) позволяет определить различного типа константы, для их последующего использования в Java-коде с помощью сгенерированного класса R. [тип константы]. [имя константы]. При добавлении элемента **Item** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for Item** с полями и списками:

Поле Name – определяет значение атрибута name, указывающего имя константы.

Список Type – определяет значение атрибута type, указывающего тип константы.

Поле Format – определяет значение атрибута format, указывающего формат значения константы.

Поле Value – определяет значение константы.

Тэг <plurals> (элемент **Quantity Strings (Plurals)**) обеспечивает строки для правильного склонения разного количества ключа ресурса. При добавлении элемента **Quantity Strings (Plurals)** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for Quantity Strings (Plurals)** с полем **Name**, определяющим значение атрибута name – ключ ресурса. Для созданного элемента кнопка **Add** позволяет добавлять дочерние элементы **Item** – тэги <item> содержащие строки ресурса. При этом раздел **Attributes for Item** содержит список **Quantity**, с помощью которого выбирается количество ресурса, которому должна соответствовать строка

(атрибут quantity). Созданный ресурс доступен в Java-коде с помощью метода getQuantityString (int id, int quantity) класса android.content.res.Resources.

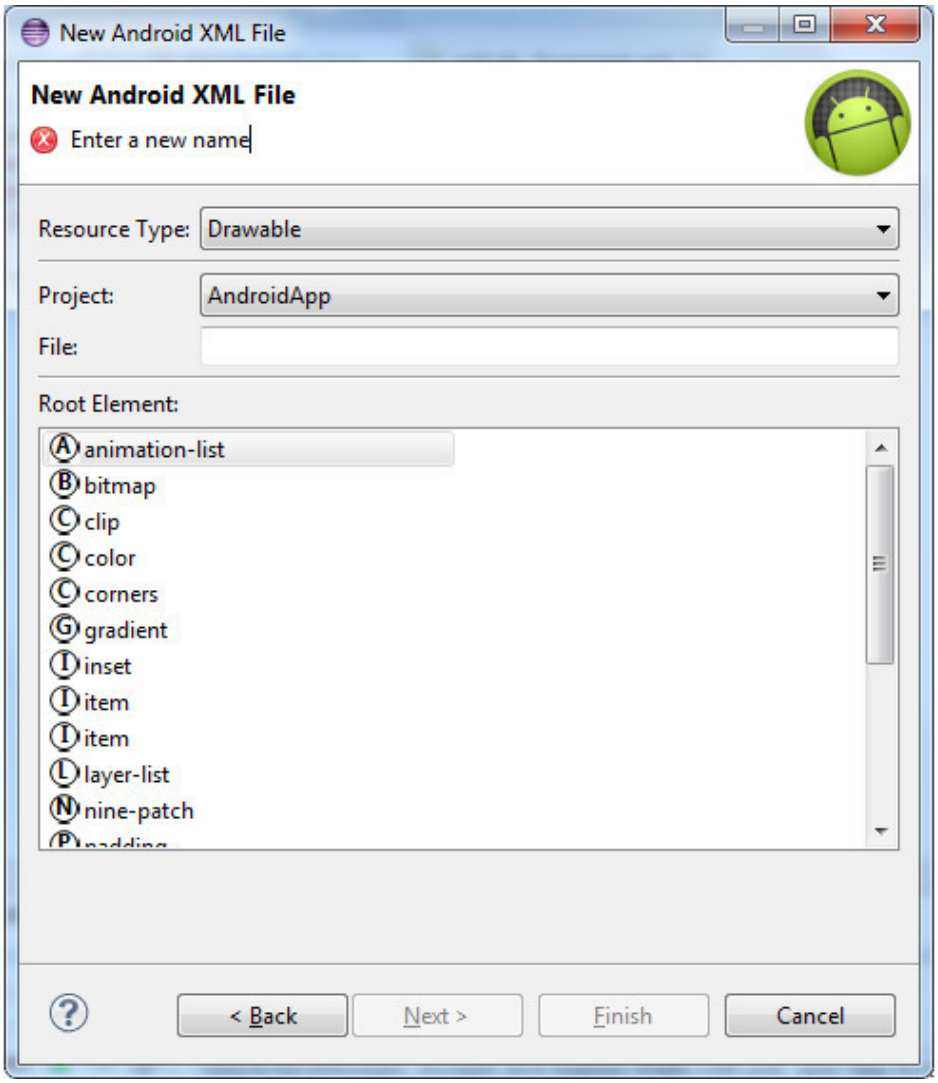
Тэг <string> (элемент **String**) определяет именованную строку, используя синтаксис <string name=«string_name»> text_string </string>. При добавлении элемента **String** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for String** с полем **Name** (определяет значение атрибута name) и полем **Value** (определяет строку). Созданный строковый ресурс доступен в Java-коде с помощью сгенерированного класса R.string.string_name, или в XML-коде – с помощью ссылки @string/string_name.

Тэг <string-array> (элемент **String Array**) определяет массив строк, используя синтаксис <string-array name=«string_array_name»> <item> text_string </item> </string-array>. При добавлении элемента **String Array** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for String Array** с полем **Name** – определяет значение атрибута name. Созданный ресурс доступен в Java-коде с помощью сгенерированного класса R.array.string_array_name или в XML-коде с помощью ссылки @ [package: ] array/string_array_name.

Тэг <style> (элемент **Style/Theme**) позволяет определить стиль для индивидуального View-компонента, для Activity-компонента и для приложения в целом, используя синтаксис <style name=«style_name» parent="@ [package: ] style/style_to_inherit»> <item name=» [package: ] style_property_name»> style_value </item> </style>. При добавлении элемента **Style/Theme** кнопкой **Add**, во вкладке **Resources** появляется раздел **Attributes for Style/Theme** с полем **Name** – определяет значение атрибута name и полем Parent – определяет значение атрибута parent. Созданный стиль доступен в Java-коде с помощью сгенерированного класса R.style.style_name или в XML-коде с помощью ссылки @ [package: ] style/style_name.

# Тип ресурса Drawable

Для создания графического ресурса Android-приложения в окне **Project Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML File**, нажмем кнопку **Next** – в результате откроется окно мастера, в списке **Resource Type** которого выберем тип **Drawable**.




Поле **File:** мастера создания графического ресурса предлагает ввести имя нового XML-файла, который затем появится в каталоге res/drawable Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса R.drawable.filename или в XML-коде с помощью ссылки @ [package: ] drawable/filename.

Раздел **Root Element:** мастера создания графического ресурса предлагает выбрать корневой XML-элемент ресурса:

<animation-list> – обеспечивает покадровую анимацию, каждый кадр которой представлен Drawable-объектом, определяемым дочерним тэгом <item>. Тэг <animation-list> имеет атрибуты android: visible (true/false, определяет видимость объекта), android: variablePadding (true/false, определяет изменяемость отступов), android: oneshot (true/false, определяет однора-

зовую или повторяющуюся анимацию). Тэг <item> имеет атрибуты android: drawable (ссылка на Drawable-объект кадра) и android: duration (продолжительность кадра в миллисекундах).

<bitmap> – обертывает PNG, JPG, GIF изображение, имеет атрибуты android: src (ссылка на обертываемое изображение), android: antialias (true/false, сглаживание изображения), android: filter (true/false, сглаживание при масштабировании изображения), android: dither (true/false, сглаживание переходов при несовпадении конфигураций изображения и экрана), android: gravity (выравнивание изображения, возможные значения top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, fill, clip_vertical, clip_horizontal), android: tileMode (режим повторения изображения для заполнения им контейнера, возможные значения disabled, clamp, repeat, mirror).

<clip> – накладывает маску на Drawable-объект, основываясь на Level-значении и используя атрибуты android: clipOrientation (ориентация маски, возможные значения horizontal, vertical), android: gravity (выравнивание маски, возможные значения top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, fill, clip_vertical, clip_horizontal), android: drawable (ссылка на исходный Drawable-объект).

<color> – создает прямоугольник, заполненный цветом, используя атрибут android: color (цвет заполнения).

<corners> – дочерний тэг тэга <shape>, определяет закругленные углы прямоугольника с помощью атрибутов android: radius (радиус всех 4 углов как ресурс <dimen>), android: topLeftRadius (радиус верхнего левого угла как ресурс <dimen>), android: topRightRadius (радиус верхнего правого угла как ресурс <dimen>), android: bottomLeftRadius (радиус нижнего левого угла как ресурс <dimen>), android: bottomRightRadius (радиус нижнего правого угла как ресурс <dimen>).

<gradient> – дочерний тэг тэга <shape>, определяет градиентную заливку геометрической формы с помощью атрибутов android: angle (угол градиента в градусах), android: centerX (относительный центр градиента по оси X, от 0 до 1.0), android: centerY (относительный центр градиента по оси Y, от 0 до 1.0), android: centerColor (промежуточный цвет градиента), android: endColor (конечный цвет градиента), android: gradientRadius (радиус для радиального градиента), android: startColor (начальный цвет градиента), android: type (тип градиента, возможные значения linear, radial, sweep), android: useLevel (true/false, если геометрическая форма участвует в <level-list>, тогда если true – количество отображений градиента зависит от уровня формы).

<inset> – вставляет Drawable-объект с отступами, используя атрибуты android: drawable (ссылка на вставляемый Drawable-объект), android: insetTop (верхний отступ как ресурс <dimen>), android: insetRight (правый отступ как ресурс <dimen>), android: insetBottom (нижний отступ как ресурс <dimen>), android: insetLeft (левый отступ как ресурс <dimen>).

<item> – дочерний тэг тэгов <animation-list>, <layer-list>, <level-list>, <selector>.

<layer-list> – стек Drawable-объектов, определяемых дочерними элементами <item> с атрибутами android: drawable (ссылка на Drawable-объект), android: id (идентификатор в форме @+id/name), android: top (верхний отступ в пикселях), android: right (правый отступ в пикселях), android: bottom (нижний отступ в пикселях), android: left (левый отступ в пикселях).

<nine-patch> – обертывает 9PNG-изображение с изменяющимися размерами, создаваемое инструментом draw9patch SDK Tools из PNG-изображения, используя атрибуты android: src (ссылка на 9PNG-изображение), android: dither (true/false, сглаживание переходов при несовпадении конфигураций изображения и экрана).

<padding> – дочерний тэг тэга <shape>, определяет отступы для содержимого формы с помощью атрибутов android: top (верхний отступ как ресурс <dimen>), android: right (правый

отступ как ресурс <dimen>), android: bottom (нижний отступ как ресурс <dimen>), android: left (левый отступ как ресурс <dimen>).

<rotate> – поворачивает Drawable-объект, основываясь на Level-значении и используя атрибуты android: visible (true/false, определяет видимость объекта), android: fromDegrees (первоначальный угол вращения), android: toDegrees (конечный угол вращения), android: pivotX (центр вращения по оси X в процентном соотношении к ширине объекта), android: pivotY (центр вращения по оси Y в процентном соотношении к высоте объекта), android: drawable (ссылка на вращаемый объект).

<scale> – масштабирует Drawable-объект, основываясь на Level-значении и используя атрибуты android: scaleWidth (масштабирование ширины в процентах), android: scaleHeight (масштабирование высоты в процентах), android: scaleGravity (выравнивание после масштабирования), android: drawable (ссылка на первоначальный Drawable-объект), android: useIntrinsicSizeAsMinimum (true/false, определяет использование собственных размеров объекта как минимальных).

<selector> – содержит набор Drawable-объектов для различных состояний View-компонента. Набор Drawable-объектов описывается дочерними тэгами <item>, которые связываются с определенными состояниями с помощью атрибутов android: drawable (ссылка на Drawable-объект), android: state_pressed (true/false), android: state_focused (true/false), android: state_hovered (true/false), android: state_selected (true/false), android: state_checkable (true/false), android: state_checked (true/false), android: state_enabled (true/false), android: state_activated (true/false), android: state_window_focused (true/false).

<shape> – описывает геометрическую форму, используя атрибут android: shape (возможные значения rectangle, oval, line, ring) и дочерние тэги <corners>, <gradient>, <padding>, <size>, <solid>, <stroke>.

<size> – дочерний тэг тэга <shape>, определяет размеры геометрической формы, используя атрибуты android: height (высота как ресурс <dimen>), android: width (ширина как ресурс <dimen>).

<solid> – дочерний тэг тэга <shape>, определяет цвет заполнения формы с помощью атрибута android: color.

<stroke> – дочерний тэг тэга <shape>, определяет контур геометрической формы, используя атрибуты android: width (ширина контура как ресурс <dimen>), android: color (цвет контура), android: dashGap (расстояние между пунктирами как ресурс <dimen>), android: dashWidth (ширина пунктира как ресурс <dimen>).

После ввода имени нового графического ресурса, выбора его корневого элемента и нажатия кнопки **Next**, появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки drawable, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка drawable с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового графического ресурса он будет открыт в XML-редакторе кода.

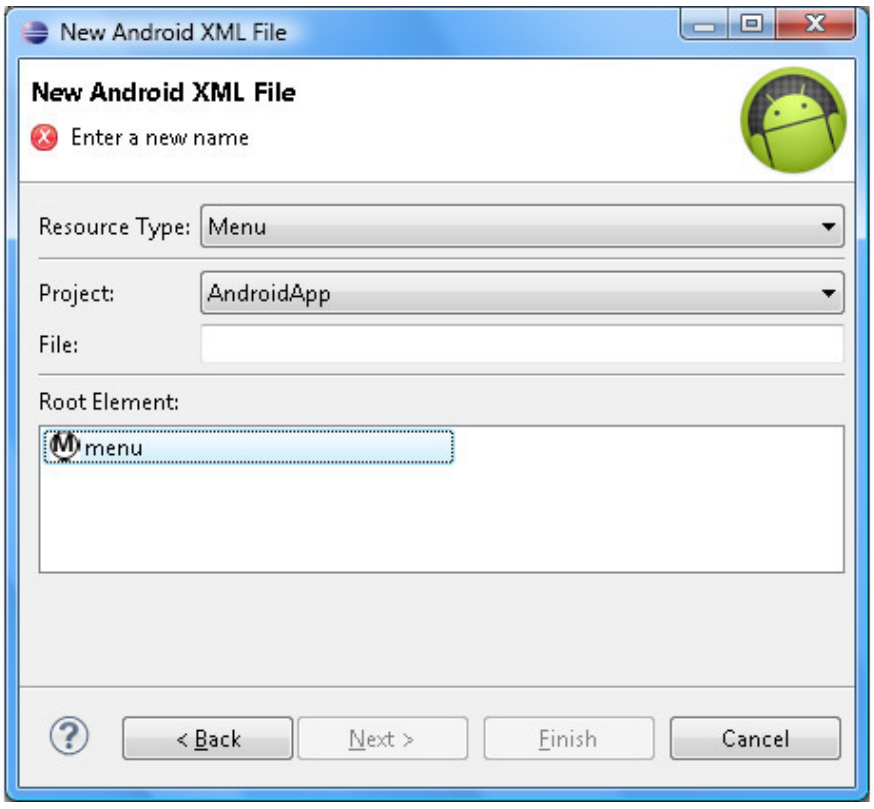
# Тип ресурса Menu

Android-платформа обеспечивает создание трех видов меню для Android-приложения – меню опций, которое открывается при нажатии кнопки **MENU** устройства или для Android-версии 3.0 и выше элементы которого могут быть помещены в ActionBar-панель, контекстное меню View-компонента и подменю элемента меню.

Все три вида меню могут быть созданы программным способом или на основе XML-описания. Создание меню на основе XML-описания является предпочтительным способом, так как позволяет разделить содержимое меню и его бизнес-логику. После создания XML-описания меню для меню опций необходимо в классе Activity-компонента переопределить метод onCreateOptionsMenu (), в котором необходимо создать программный объект из XML-описания, используя метод MenuInflater.inflate (), а также переопределить метод onOptionsItemSelected (), обрабатывающий выбор элемента меню.

Для контекстного меню необходимо в методе onCreate () Activity-компонента зарегистрировать View-компонент как имеющий контекстное меню с помощью метода registerForContextMenu (), переопределить метод onCreateContextMenu (), в котором необходимо создать программный объект из XML-описания, используя метод MenuInflater.inflate (), а также переопределить метод onContextItemSelected (), обрабатывающий выбор элемента меню. Подменю элемента меню определяется простым вложением его XML-описания в тэг элемента меню.

Для создания XML-описания меню Android-приложения в окне **Project Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML File**, нажмем кнопку **Next** – в результате откроется окно мастера, в списке **Resource Type** которого выберем тип **Menu**.



Поле **File:** мастера создания Menu-файла предлагает ввести имя нового файла XML-описания меню, который затем с расширением. xml появится в каталоге res/menu Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса R.menu. [имя Menu-файла] или в XML-коде с помощью ссылки @ [package: ] menu. [имя Menu-файла].

Раздел **Root Element:** мастера создания Menu-файла показывает, что корневым элементом XML-файла служит тэг <menu>.

После ввода имени нового Menu-файла и нажатия кнопки **Next**, появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки menu, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка menu с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового Menu-файла он будет открыт в редакторе ADT-плагина, обеспечивающим визуальное редактирование XML-описания меню. Кнопка **Add** вкладки **Layout** редактора Menu-файла обеспечивает добавление в корневой тэг <menu> тэги <group> (элемент **Group**) и <item> (элемент **Item**).

Тэг <item> описывает элемент меню, может быть дочерним тэгом тэга <menu> и <group> и иметь в качестве дочернего тэг <menu>, представляющий подменю (элемент **Sub-Menu**). Тэг <item> имеет следующие атрибуты:

android: id – идентификатор элемента в виде @+id/name.

android: menuCategory – категория элемента меню, определяющая его приоритет (номер в списке) при отображении, возможные значения container, system, secondary, alternative.

android: orderInCategory – номер элемента в списке отображения в пределах категории.

android: title – текстовая метка элемента.

android: titleCondensed – укороченная текстовая метка элемента.

android: icon – ссылка на Drawable-ресурс, представляющий значок элемента, который отображается для первых 6 элементов меню опций.

android: alphabeticShortcut – символ быстрого вызова элемента.

android: numericShortcut – цифра быстрого вызова элемента.

android: checkable – если true, тогда элемент содержит флажок выбора.

android: checked – если true, тогда флажок элемента отмечен по умолчанию.

android: visible – если true, тогда элемент видим.

android: enabled – если true, тогда элемент доступен.

android: onClick – имя метода, вызываемого при нажатии элемента.

android: showAsAction – определяет как элемент отображается в ActionBar-панели, возможные значения ifRoom (отображается при наличии места в панели), never (не отображается), withText (отображается с меткой), always (всегда отображается), collapseActionView (с элементом связан разворачивающийся View-компонент).

android: actionLayout – ссылка на Layout-файл, описывающий View-компонент элемента ActionBar-панели.

android: actionViewClass – имя класса View-компонента элемента ActionBar-панели.

android: actionProviderClass – имя ActionProvider-класса, связанного с элементом ActionBar-панели.

Тэг <group> позволяет сгруппировать элементы меню так, что для них всех одновременно можно регулировать видимость, доступность и отображение флажка выбора. Тэг <group> имеет следующие атрибуты:

android: id – идентификатор группы в виде @+id/name.

android: menuCategory – категория группы элементов меню, определяющая ее приоритет (номер в списке) при отображении, возможные значения container, system, secondary, alternative.

android: orderInCategory – номер группы в списке отображения в пределах категории.

android: checkableBehavior – тип группировки элементов, возможные значения none (элементы не отображают флажок выбора), all (элементы группируются как флажки checkbox), single (элементы группируются как переключатели radio button).

android: visible – видимость элементов группы, true/false.

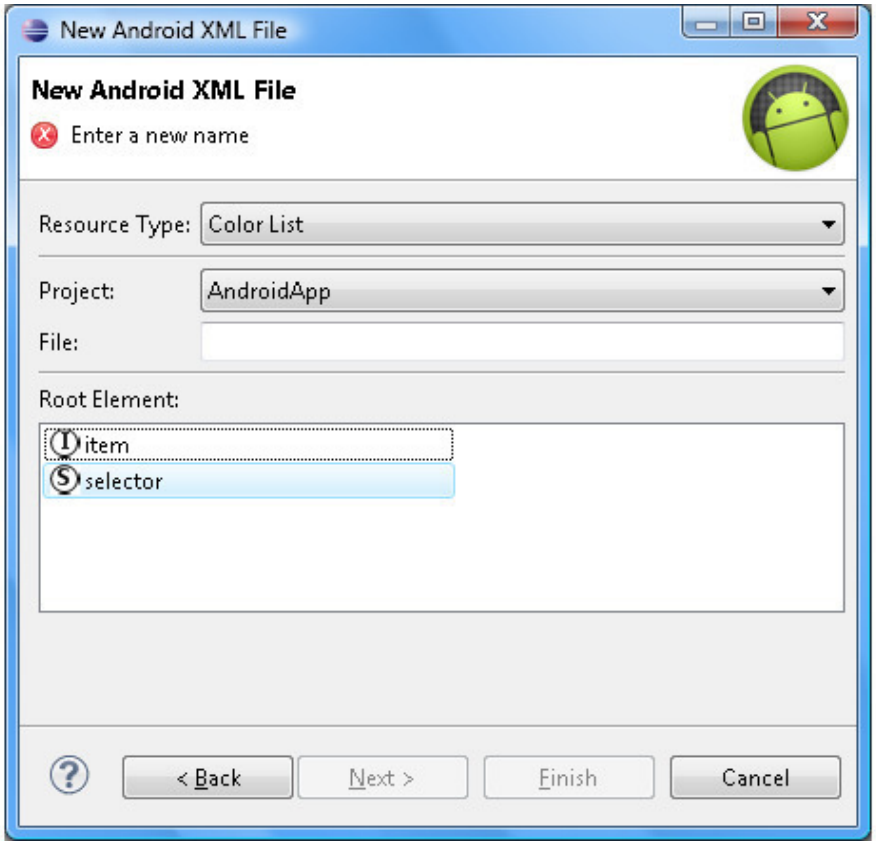
android: enabled – доступность элементов группы, true/false.

При выборе элементов **Group** и **Item** кнопкой **Add**, во вкладке **Layout** появляются разделы **Attributes for Group** и **Attributes for Item** с полями, позволяющими определить атрибуты тэгов <group> и <item>.

# Тип ресурса Color List

Данный тип ресурса является аналогом Drawable-ресурса State List с корневым элементом <selector>. Разница состоит в том, что ресурс Drawable State List определяет набор изображений для представления различных состояний View-компонента, а ресурс Color State List определяет набор цветов для представления различных состояний View-компонента.

Для создания ресурса Color State List в окне **Project Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML File**, нажмем кнопку **Next** – в результате откроется окно мастера, в списке **Resource Type** которого выберем тип **Color List**.




Поле **File:** мастера создания ресурса Color State List предлагает ввести имя нового файла XML-описания набора цветов различных состояний View-компонента, который затем с расширением. xml появится в каталоге res/color Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса R.color.filename или в XML-коде с помощью ссылки @ [package: ] color/filename.

Раздел **Root Element:** мастера отображает элементы **item** и **selector**, представляющие тэги <item> и <selector> соответственно, при этом тэг <selector> является корневым тэгом XML-файла ресурса Color State List и поэтому в разделе **Root Element:** необходимо выбрать элемент **selector**.

Тэг <selector> может содержать один или несколько тэгов <item>, определяющих цвета для различных состояний View-объекта, используя атрибуты:

android: color – цвет состояния в формате #RGB, #ARGB, #RRGGBB, #AARRGGBB.

android: state_pressed – состояние нажатия, true/false.

android: state_focused – компонент в фокусе, true/false.
android: state_selected – компонент выбран, true/false.
android: state_checkable – компонент содержит флажок выбора, true/false.
android: state_checked – флажок компонента отмечен, true/false.
android: state_enabled – компонент доступен, true/false.
android: state_window_focused – окно приложения на переднем плане, true/false.

После ввода имени нового XML-файла ресурса Color State List, выбора элемента **selector** и нажатия кнопки **Next**, появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки color, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка color с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

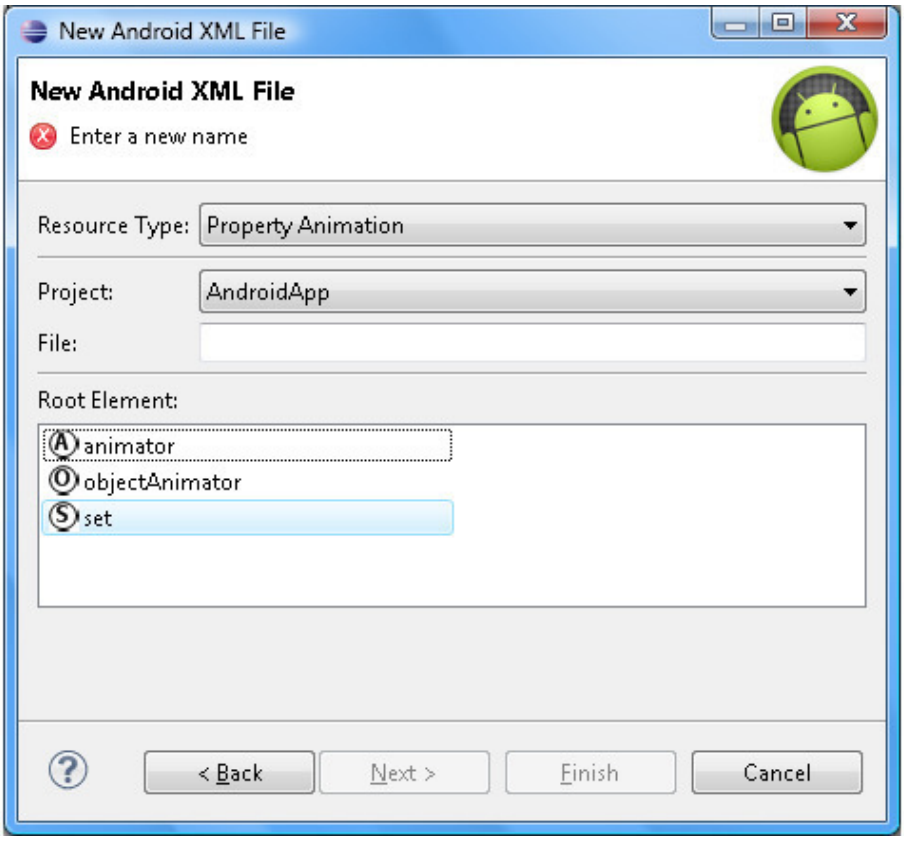
После создания нового XML-файла ресурса Color State List он будет открыт в XML-редакторе кода.

# Тип ресурса Property Animation и Tween Animation

Ресурс Property Animation описывает изменение свойства объекта в течение определенного промежутка времени. Анимация свойств объектов представлена в версиях Android-платформы, начиная с версии 3.0.

Для запуска анимации свойства объекта на основе XML-описания необходимо создать из XML-ресурса Property Animation объект android.animation.AnimatorSet, android.animation. ObjectAnimator или android.animation.ValueAnimator, используя статический метод android.animation.AnimatorInflater. loadAnimator (), и связать анимацию с объектом с помощью метода setTarget () суперкласса android.animation.Animator, после чего запустить анимацию методом start () суперкласса Animator.

Для создания ресурса Property Animation в окне **Project Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML File**, нажмем кнопку **Next** – в результате откроется окно мастера, в списке **Resource Type** которого выберем тип **Property Animation**.




Поле **File:** мастера создания ресурса Property Animation предлагает ввести имя нового файла XML-описания анимации, который затем с расширением. xml появится в каталоге res/ animator Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса R.animator.filename или в XML-коде с помощью ссылки @ [package: ] animator/filename.

Раздел **Root Element:** мастера отображает элементы **animator**, **objectAnimator** и **set**, представляющие тэги <animator>, <objectAnimator> и <set> соответственно, при этом каждый из них может служить единственным корневым тэгом XML-файла ресурса Property Animation.

Тэг <animator> представляет класс ValueAnimator и описывает анимацию значения типа float, int или color в течение определенного промежутка времени, используя атрибуты:

android: duration – время анимации в миллисекундах, по умолчанию 300ms.

android: valueFrom – начальное значение.

android: valueTo – конечное значение.

android: startOffset – задержка анимации в миллисекундах.

android: repeatCount – количество циклов анимации, значение -1 означает бесконечную анимацию.

android: repeatMode – режим повторения анимации, возможные значения repeat и reverse.

android: valueType – тип значения для анимации, для значения типа color не указывается, возможные значения intType и floatType (по умолчанию).

Тэг <objectAnimator> представляет класс ObjectAnimator и описывает анимацию значения свойства объекта в течение определенного промежутка времени, используя атрибуты:

android: propertyName – имя свойства объекта, например android: propertyName=«alpha».

android: duration – время анимации в миллисекундах, по умолчанию 300ms.

android: valueFrom – начальное значение свойства.

android: valueTo – конечное значение свойства.

android: startOffset – задержка анимации в миллисекундах.

android: repeatCount – количество циклов анимации, значение -1 означает бесконечную анимацию.

android: repeatMode – режим повторения анимации, возможные значения repeat и reverse.

android: valueType – тип значения для анимации, для значения типа color не указывается, возможные значения intType и floatType (по умолчанию).

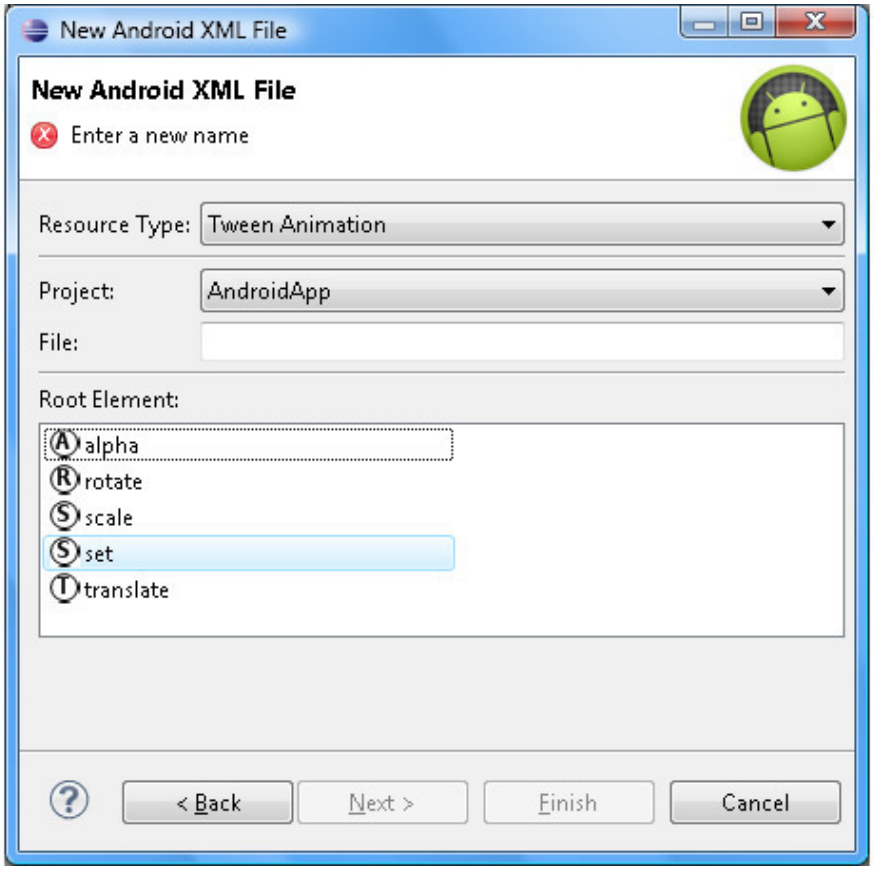
Тэг <set> представляет класс AnimatorSet и обеспечивает группировку анимаций, используя атрибут android: ordering с возможными значениями together (анимации проигрываются параллельно) и sequentially (анимации проигрываются последовательно).

После ввода имени нового XML-файла ресурса Property Animation, выбора корневого элемента и нажатия кнопки **Next**, появляется окно **Choose Configuration Folder**, позволяющее выбрать спецификатор папки animator, обеспечивающий поддержку специфической конфигурации Android-устройства, в соответствии с которой папка animator с нужным спецификатором будет выбрана Android-системой для загрузки при выполнении кода приложения.

После создания нового XML-файла ресурса Property Animation он будет открыт в XML-редакторе кода.

Ресурс Tween Animation описывает анимацию вращения, исчезновения, перемещения и масштабирования View-компонента. Для запуска анимации View-компонента на основе XML-описания необходимо создать из XML-ресурса Tween Animation объект android.view.animation.Animation, используя статический метод android.view.animation.AnimationUtils. loadAnimation () и запустить анимацию методом startAnimation (Animation animation) суперкласса android.view.View.

Для создания ресурса Tween Animation в окне **Project Explorer** нажмем правой кнопкой мышки на узле проекта и в контекстном меню выберем команду **New | Other | Android | Android XML File**, нажмем кнопку **Next** – в результате откроется окно мастера, в списке **Resource Type** которого выберем тип **Tween Animation**.




Поле **File:** мастера создания ресурса Tween Animation предлагает ввести имя нового файла XML-описания анимации, который затем с расширением. xml появится в каталоге res/anim Android-проекта и будет доступен в Java-коде с помощью сгенерированного класса R.anim.filename или в XML-коде с помощью ссылки @ [package: ] anim/filename.

Раздел **Root Element:** мастера отображает элементы **alpha**, **rotate**, **scale**, **set** и **translate**, представляющие тэги <alpha>, <rotate>, <scale>, <set> и <translate> соответственно, при этом каждый из них может служить единственным корневым тэгом XML-файла ресурса Tween Animation.

Вышеупомянутые тэги имеют общие атрибуты, унаследованные от суперкласса android.view.animation.Animation:

android: detachWallpaper – если true, тогда обои не анимируются вместе с окном.

android: duration – продолжительность анимации в миллисекундах.

android: fillAfter – если true, тогда преобразование применяется после окончания анимации.

android: fillBefore – если true, тогда преобразование применяется перед началом анимации.

android: fillEnabled – если true, тогда значение fillBefore учитывается.

android: interpolator – указывает объект android.view.animation.Interpolator, отвечающий за определение скорости анимации.

android: repeatCount – количество циклов анимации.

android: repeatMode – режим повторения анимации, возможные значения repeat и reverse.

android: startOffset – задержка анимации в миллисекундах.

android: zAdjustment – определяет поведение компонента по оси Z, возможные значения normal (позиция в стеке сохраняется), top (компонент в течение анимации находится на вершине стека), bottom (компонент в течение анимации находится внизу стека).

Тэг <alpha> представляет класс AlphaAnimation и описывает анимацию значения прозрачности в течение определенного промежутка времени, используя атрибуты:

android: fromAlpha – начальное значение прозрачности.

android: toAlpha – конечное значение прозрачности.

Тэг <rotate> представляет класс RotateAnimation и описывает вращение вокруг оси, используя атрибуты:

android: fromDegrees – начальный угол вращения.

android: toDegrees – конечный угол вращения.

android: pivotX и android: pivotY – координаты оси вращения от левого края компонента в пикселях или процентах.

Тэг <scale> представляет класс ScaleAnimation и описывает масштабирование компонента, используя атрибуты:

android: fromXScale – начальный коэффициент масштабирования по оси X.

android: toXScale – конечный коэффициент масштабирования по оси X.

android: fromYScale – начальный коэффициент масштабирования по оси Y.

android: toYScale – конечный коэффициент масштабирования по оси Y.

android: pivotX и android: pivotY – координаты центра масштабирования.

Тэг <translate> представляет класс TranslateAnimation и описывает движение компонента, используя атрибуты:

# Конец ознакомительного фрагмента.