Ресурсы

Основы

Ресурсы - один из основных компонентов, с которыми вам придется работать очень часто. В Android принято держать некоторые объекты - изображения, строковые константы, цвета, анимацию, стили и т.п. за пределами исходного кода. Система поддерживает хранение ресурсов в отдельных файлах. Ресурсы легче поддерживать, обновлять, редактировать.

Каждое приложение на Android содержит каталог для ресурсов **res** и каталог для активов **assets**. Реальное различие между ресурсами и активами заключается в следующем:

* информация в каталоге ресурсов будет доступна в приложении через класс **R**, который автоматически генерируется средой разработки. То есть хранение файлов и данных в ресурсах (в каталоге **res**) делает их легкодоступными для использования в коде программы;
* для чтения информации, помещенной в каталог активов **assets** (необработанный формат файла), необходимо использовать **AssetManager** для чтения файла как потока байтов.

Android умеет динамически выбирать данные из дерева ресурсов, содержащие разные значения для разных конфигураций, языков и регионов. При запуске Android автоматически загрузит нужный ресурс, не требуя ни одной строчки кода.

Ресурсы в Android являются декларативными. В основном ресурсы хранятся в виде XML-файлов в каталоге **res** с подкаталогами **values**, **drawable-ldpi**, **drawable-mdpi**, **drawable-hdpi**, **layout**, но также бывают и другие типы ресурсов.

Для удобства система создает идентификаторы ресурсов и использует их в файле **R.java** (класс **R**, который содержит ссылки на все ресурсы проекта), что позволяет ссылаться на ресурсы внутри кода программы. Статический класс **R**генерируется на основе ваших заданных ресурсов и создается во время компиляции проекта. При создании класс содержит статические подклассы для всех типов ресурсов, для которых был описан хотя бы один экземпляр. Подробнее о классе **R** [рассказано ниже](http://developer.alexanderklimov.ru/android/theory/resources.php#using).

Так как файл **R** генерируется автоматически, то не смысла его редактировать вручную, потому что все изменения будут утеряны при повторной генерации.

В общем виде ресурсы представляют собой файл (например, изображение) или значение (например, заголовок программы), связанные с создаваемым приложением. Удобств использования ресурсов заключается в том, что их можно изменять без повторной компиляции или новой разработки приложения. Имена файлов для ресурсов должны состоять исключительно из букв в нижнем регистре, чисел и символов подчеркивания.

Самыми распространенными ресурсами являются, пожалуй, строки (string), цвета (color) и графические рисунки (bitmap). В приложении не рекомендуется применять жестко написанные строки кода - вместо них следует использовать соответствующие идентификаторы, что позволяет изменять текст строкового ресурса, не изменяя исходного кода.

Начинающие программисты не всегда до конца правильно понимают процесс создания ресурсов. В Android используются два подхода - первый подход заключается в том, что ресурсы задаются в файле, при этом имя файла значения не имеет. Второй подход - ресурс задается в виде самого файла, и тогда имя файла уже имеет значение (при этом нужно учитывать определенные нюансы).

Общая структура каталогов, содержащих ресурсы выглядит следующим образом:

/res/values/strings.xml

/colors.xml

/dimens.xml

/attrs.xml

/styles.xml

/drawable/\*.png

/\*.jpg

/\*.gif

/\*.9.png

/anim/\*.xml

/layout/\*.xml

/raw/\*.\*

/xml/\*.xml

/assets/\*.\*/\*.\*

Только в **assets** может располагаться любой набор подкаталогов разной вложенности. Файлы, находящиеся в любом другом каталоге, размещаются именно на уровне этого каталога и не глубже.

Перечисление основных ресурсов Android

|  |  |  |
| --- | --- | --- |
| **Тип ресурса** | **Размещение** | **Описание** |
| Цвета | res/values/имя\_файла | Идентификатор цвета, указывающий на цветовой код. ID таких ресурсов выражаются в **R.java** как **R.color.\***. XML-узел: /resources/color |
| Строки | res/values/имя\_файла | Строковые ресурсы. В их число также входят строки в формате java и html. ID таких ресурсов выражаются в **R.java** как **R.string.\***. XML-узел: resources/string. Можно использовать дополнительное форматирование при помощи стандартных html-тегов <b>, <i> и <u>. Методы, которые будут обрабатывать строковые ресурсы с HTML-форматированием, должны уметь обрабатывать эти теги. |
|  |  |  |
| Меню | /res/menu/имя\_файла | Меню в приложении можно задать как XML-ресурсы. |
| Параметры | /res/values/имя\_файла | Представляет собой параметры или размеры различных элементов. Поддерживает пикселы, дюймы, миллиметры, не зависящие от плотности экрана пикселы (dip) и пикселы, не зависящие от масштаба. ID таких ресурсов выражаются в **R.java** как **R.dimen.\***. XML-узел: resources/dimen |
| Изображения | /res/drawable/ваши\_файлы | Ресурсы-изображения. Поддерживает форматы JPG, GIF, PNG (самый предпочтительный) и др. Каждое изображение является отдельным файлом и получает собственный идентификатор, который формируется по имени файла без расширения. Такие ID ресурсов представлены в файле R.java как R.drawable.\*. Система также поддерживает так называемые растягиваемые изображения (stretchable image), в которых можно менять масштаб отдельных элементов, а другие элементы оставлять без изменений. |
| Отрисовываемые цвета | /res/values/ваш\_файл или /res/drawable/ваши\_файлы | Представляет цветные прямоугольники, которые используются в качетве фона основных отрисовываемых объектов, например точечных рисунков. Поддержка такой функции обеспечивается тегом значения drawable, находящимся в подкаталоге значений. Такие id ресурсов выражаются в файле R.java как R.drawable.\*. ХМL-узел для такого файла: /resources/drawable. В Android при помощи специальных ХМL-файлов, расположенных в /res/drawable, также подерживаются скругленные и градиентные прямоугольники. Корневым ХМL-тегом для drawable является <shape>. Идентификаторы таких ресурсов выражаются в файле R.java как R.drawable.\*. В таком случае, каждое имя файла преобразуется в уникальный id отрисовываемого объекта |
| Анимация | /res/anim/ваш\_файл | Android может выполнить простую анимацию на графике или на серии графических изображений. Анимация включает вращения, постепенное изменение, перемещение и протяжение. |
| Произвольные XML-файлы | /res/xml/\*.xml | В Android в качестве ресурсов могут использоваться произвольные XML-файлы. Они компилируются в aapt. Идентификаторы таких ресурсов также выражаются в файле R.java как R.xml.\* |
| Произвольные необработанные ресурсы | /res/raw/\*.\* | Любые нескомпилированные двоичные или текстовые файлы, например, видео. Каждый файл получает уникальный id ресурса. Идентификаторы таких ресурсов выражаются в файле R.java как R.raw.\* |
| Произвольные необработанные активы | /assets/\*.\*/\*.\* | Можно использовать произвольные файлы в произвольно названных каталогах, которые находятся в подкаталоге /assets. Это не ресурсы, а просто необработанные файлы. В этом каталоге, в отличие от /res, подкаталоги могут располагаться на любой глубине. Для таких файлов не создаются идентификаторы ресурсов. При работе с ними нужно использовать относительное имя пути, начиная с /assets, но не указываея этого каталгоа в имени пути |

Идентификаторы

Этот тип ресурсов формируется, как правило, автоматически, и программисты даже не обращают на него внимания. Когда вы размещаете новый элемент на форме, с которым будете взаимодействовать в программе, то ему нужно присвоить идентификатор. Как правило, это происходит в виде **@+id/editText** (часто это происходит автоматически). Знак плюса обозначает, что если идентификатора не существует, то его нужно создать в классе **R**. В программе вы можете обращаться к элементу **R.id.editText**.

Но можно заранее создать ресурс типа **item** для задания **id**, не связанного ни с каким конкретным ресурсом:

<resources>

<item type="id" name="text"/>

</resources>

Здесь **type** описывает тип ресурса, в данном случае **id**. Когда **id** будет установлен, будет работать и следующее определение **View**:

<TextView android:id="@id/textView">

</TextView>

Обычно идентификаторы размещают в отдельном файле **res/values/ids.xml**.

Строковые ресурсы

Строковые ресурсы помогают упростить процесс создания локализованных версий. Строковые ресурсы обозначаются тегом **<string>.**,

При [разработке первого приложения](http://developer.alexanderklimov.ru/android/android2.php) вы видели, что система создала файл **strings.xml**, в котором хранились строки для заголовка приложения и выводимого сообщения. Вы можете редактировать данный файл, добавляя новые строковые ресурсы. А также вы можете создать новые файлы, которые будут содержать строковые ресурсы, например, **strings2.xml**, **catnames.xml** и т.д. Все эти файлы должны находиться в подкаталоге **/res/values**. Запомните, имена файлов и их число не важно. Но в большинстве случаев программисты используют для строковых ресурсов стандартное имя **strings.xml**. Типичный файл выглядит следующим образом.

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="hello">Здравствуй, Мир!</string>

<string name="app\_name">Hello, World</string>

</resources>

При создании или обновлении файла со строковыми ресурсами среда разработки автоматически создает или обновляет класс **R.java**, сообщая уникальные ID для определенных в файле строковых ресурсов (Независимо от количества файлов ресурсов, в проекте содержится только один файл **R.java**). Если открыть данный файл, то можно найти там наши ресурсы в следующем виде:

public static final int hello = 0x7f040000;

public static final int app\_name = 0x7f040001;

В принципе достаточно запомнить, что **R.java** создает внутренний статический класс как пространство имен, в котором содержатся ID строковых ресурсов. Два **static final int**, которые используются в переменных **hello** и **app\_name**, являются идентификаторами ресурсов и соответствуют соответствующим строковым ресурсам. Вы можете использовать данные идентификаторы в своем исходном коде, используя следующий формат - **R.string.hello**.

Обратите внимание, что сгенерированные ID указывают на **int**, а не на **String**. Android при необходимости самостоятельно подставляет вместо **int** нужные строки.

Обычно принято хранить строковые ресурсы в файле **strings.xml**, но вы можете использовать несколько файлов. Главное, чтобы XML-файл имел необходимую структуру и находился в подкаталоге **res/values**.

Структура файла для строковых ресурсов довольно проста. Имеется корневой узел <resources>, за которым следуют один или несколько дочерних элементов <string>. Каждый элемент <string> в свою очередь имеет свойство **name**, которое в файле **R.java** представляет собой атрибут **id**.

Если вы создаете несколько файлов с ресурсами, то следите за уникальностью создаваемых имен. Не выйдет ничего хорошего, если в двух файлах будет одна и та же переменная **app\_name**.

Запомните, что пробелы в начале и в конце строк обрезаются. Если вам так нужны пробелы, то разместите строку в кавычках и строка будет выводиться как есть. Также можно попробовать использовать код **\u0020** вместо пробела.

Такая же проблема и с несколькими пробелами внутри строки - будет выводиться только один пробел (напоминает поведение в html-документе).

Продвинутые приемы работы со строковыми ресурсами

Кроме стандартного использования строковых ресурсов, можно использовать более сложные приёмы. Посмотрим, как определять и использовать строки, написанные на HTML, а также узнаем, как происходит подстановка переменных в строковых ресурсах.

Начнем с того, как определять в ХМL-файлах ресурсов следующие виды строк:

* обычные строки
* строки в кавычках
* НТМL-строки
* заменяемые строки

<resources>

<string name="simple\_string">Простая строка</string>

<string name="quoted\_ string">"Строка в кавычках"</string>

<string name="double\_quoted\_string">\"Строка в двойных кавычках\"</string>

<string name="java\_format\_string">Привет %2$s. Здравствуй %1$s</string>

<string name="tagged\_string"><b>Рыжик</b> - <i>мой любимый кот</i></string>

</resources>

Этот ХМL-файл строковых ресурсов должен находиться в подкаталоге **res/values**. Имя файла выбирается произвольно.

Обратите внимание - строки, находящиеся в кавычках, необходимо либо пропускать, либо помещать еще в одни кавычки. При определении строк также можно использовать стандартные последовательности Jаvа, предназначенные для форматирования строк.

Нельзя использовать символы '&', '<'. Задать эти символы можно используя специальную последовательность < — &lt; или & — &amp;. Если текст содержит html-теги и в нем встречается неразрывный пробел &nbsp;, то его надо заменить на &#160;.

Также можно использовать простые элементы HTML, предназначенные для форматирования, используя теги <b> (полужирный шрифт), <i> (наклонный шрифт), <u> (подчеркнутый шрифт). Вы можете использовать такую НТМL-строку для оформления текста, перед тем как выводить текст на экран.

Каждый вариант использования проиллюстрирован в листинге на примере кода Jаvа.

// Считывание обычной строки и помещение ее в текстовый вид

String simpleString = activity.getString (R.string.simple\_string);

textView.setText(simpleString);

// Считывание строки в кавычках и помещение ее в текстовый вид

String quotedString = activity.getString(R.string.quotedstring);

textView.setText(quotedString);

// Считывание строки в двойных кавычках и помещение ее в текстовый вид

String doubleQuotedString = activity.getString(R.string.double\_quoted\_string);

textView.setText(doubleQuotedString);

// Считывание строки форматирования Java

String javaFormatString = activity.getString(R.string.java\_format\_string);

// Преобразование отформатированной строки при помощи данных

// передаваемых в аргументах

String substitutedString = String.format(javaFormatString,

"Рыжик", "Барсик" );

// помещение вывода в текстовый вид

textView.setText(substitutedString);

// Считывание строки html\_string из ресурса и помещение ее в текстовый вид

String htmlTaggedString = activity.getString(R.string.tagged\_string);

// Преобразование строки во фрагмент текста,

// который может быть помещен в текстовом виде

// Класс android.text.Html допускает рисование строк "html" (не все теги)

Spanned textSpan = android.text.Html.fromHtml(htmlTaggedString);

// Поместить информацию в текстовый вид

textView.setText(textSpan);

После того, как вы определите строки в виде ресурсов, вы можете вставить их прямо в вид. Например, воспользуемся строкой на HTML в элементе **TextView**:

<TextView

android:layout\_width="match\_ parent"

...

android:text="@string/tagged string" />

**TextView** автоматически определяет, что эта строка написана на HTML, и соответствующим образом обрабатывает форматирование этой строки. Данный пример использовался в статье о [создании нового окна](http://developer.alexanderklimov.ru/android/activity.php).

Можно использовать строковые ресурсы в качестве входящих параметров для метода **String.format**. Однако данный метод не поддерживает стилизацию текста через теги <b>, <i>, <u>. Что выйти из этого положения, экранируйте HTML-теги следующим образом:

<string name="my\_message">&lt;b>Жирный кот&lt;/b></string>

А в коде используйте метод **Html.fromHtml** для преобразования строки в нужном виде:

String rString = getString(R.string.my\_message);

String fString = String.format(rString, "Bla-bla-bla");

CharSequence styledString = Html.fromHtml(fString);

Сложный текст с проблемными символами можно загнать в особый контейнер **CDATA**, который знаком веб-мастерам.

<string name="long\_message">

<![CDATA[

<html>

<body>

<h1>Котик: {{birthDate}}</h1>

<p>Здесь текст о котиках:</p>

<p>{{message}}</p>

<p>Если хотите поговорить об этом, то обращайтесь к <b>Котовскому</b>.</p>

</body>

</html>

]]>

</string>

Существует особый вид ресурсов для строк - [Строковые массивы](http://developer.alexanderklimov.ru/android/theory/resources.php#array)

Есть еще один вид ресурсов, который редко используется, но может пригодиться - **plurals**, позволяющий задавать строки для разных ситуаций. Например, в английском языке мужчина в единственном числе - man, а во множественном - men.

<plurals name="NumberOfMan">

<item quantity="one">man</item>

<item quantity="other">men</item>

</plurals>

<plurals name="cats">

<item quantity="one">One Cat</item>

<item quantity="other">%d Cats</item>

</plurals>

Два варианта представлены в виде двух отдельных строк одного числительного. Теперь можно использовать Java-код для применения этого множественного числа при выводе строки с указанием количества.

Resources res = getResources();

String s1 = res.getQuantityString(R.plurals.NumberOfMan, 0, 0);

Первый параметр в методе **getQuantityString()** - это идентификатор ресурса множественного числа, второй указывает на нужную строку, если значение равно 1, то строка берётся без изменений. Третий параметр - строка, которая подставляется, если второй параметр не равен 1.

В английском языке можно использовать только два значения: **one** и **other**. В русском возможны три варианта: one (для 1), few (для 2-4), other (в остальных случаях).

Более подробно об этом типе ресурсов читайте в отдельной статье [Ресурсы plurals. Множественное число](http://developer.alexanderklimov.ru/android/theory/plurals.php).

Системные строковые ресурсы

Во многих случаях можно задействовать системные строковые ресурсы - это строки типа OK, Cancel и др. В таких ситуациях используется схожий формат (добавляется слово android):

android:text="@android:string/cancel"

xliff

В строковых ресурсах может применяться особый способ с использованием специального формата **xliff**. На практике мне не встречался, но вдруг вам пригодится.

Создадим ресурсы следующим образом.

<?xml version="1.0" encoding="utf-8"?>

<resources xmlns:xliff="rn:oasis:names:tc:xliff:document:1.2">

...

<string name="title">Ваше имя: <xliff:g id="name" example="Александр" >%2$s</xliff:g>,

Имя кота: <xliff:g id="cat\_name" example="Рыжик" >%1$s</xliff:g>

</string>

</resources>

Обратите внимание на пространство имён, определяющий тип документа.

Ресурс **title** содержит несколько вставок, обрамленных в тег **xliff**, которые будут заполняться динамически. Каждый тег **xliff** включает в себя атрибуты **id** с уникальным именем тега и **example** с примером его содержимого. Далее заполняем участки с **xliff**:

TextView textView = findViewById(R.id.textView);

textView.setText(getString(R.string.title, "Васька", "Толя"));

С помощью метода **getString()** достаем нужный ресурс, где первый параметр — это **id** ресурса, второй параметр «Васька» заменит плэйсхолдер **%1$s**, а третий — «Толя» заменит **%2$s**.

Вы можете использовать любой порядок для вывода текста, который будет зависеть от чисел **%1**, **%2** и т.д. В строке можно использовать несколько раз один и тот же плэйсхолдер, например, %1$s, и он везде будет заполнен одним и тем же параметром — «Васька».

Булевы ресурсы

Можно также хранить в ресурсах булевы значения **true** или **false** в файле с произвольным именем в папке **res/values**.

В файле с корневым элементом <resources> вы определяете элемент **bool** с нужным значением. У элемента есть атрибут **name** - строка, определяющая имя булевого ресурса.

<?xml version="1.0" encoding="utf-8"?>

<resources>

<bool name="autostart">true</bool>

<bool name="sound">no</bool>

</resources>

Получить значение через код:

Resources res = getResources();

boolean autostartSetting = res.getBoolean(R.bool.autostart);

В макете для булевых атрибутов

<ImageView

android:layout\_height="fill\_parent"

android:layout\_width="fill\_parent"

android:adjustViewBounds="@bool/adjust\_view\_bounds" />

Числовые ресурсы

В ресурсах можно хранить числа типа **Integer**. Хранить можно в произвольном имени XML-файла в папке **res/values/** в корневом элементе <resources>

У элемента <integer> есть атрибут **name**, определяющий имя числового значения.

<?xml version="1.0" encoding="utf-8"?>

<resources>

<integer name="max\_speed">75</integer>

<integer name="min\_speed">5</integer>

</resources>

Для работы в коде:

Resources res = getResources();

int maxSpeed = res.getInteger(R.integer.max\_speed);

Таким образом, для работы с типами **boolean** и **int** следует применять код (в общем виде):

Resources resources = context.getResources();

boolean myBooleanResource = resources.getBoolean(R.bool.my\_boolean\_resource);

int myIntegerResource = resources.getInteger(R.integer.my\_integer\_resource);

Нет необходимости писать избыточный код (хотя он и будет работать) типа такого:

boolean myBooleanResource = Boolean.parseBoolean(context.getString(R.string.my\_boolean\_resource));

int myIntegerResource = Integer.parseInt(context.getString(R.string.my\_integer\_resource));

// или так (совсем никуда не годится)

if ("false".equals(context.getString(R.string.my\_boolean\_resource))){

// ...

}

Ресурсы меню

Не создавайте меню в коде приложения, а используйте отдельные ресурсы для меню в формате XML. Можно использовать как для описания обычного и контекстного меню. Меню, описанное в формате XML, загружается в виде программного объекта с помощью метода **inflate**, принадлежащего сервису **MenuInflater**. Как правило, это происходит внутри обработчика **onCreateOptionsMenu** (смотри урок [Меню](http://developer.alexanderklimov.ru/android/menu.php)).

Описание меню хранится в отдельном файле в каталоге **res/menu**. Имена файлов без расширения автоматически становятся идентификаторами ресурсов.

В ХМL-файле меню есть три элемента:

* <menu> — корневой элемент файла меню;
* <group> — контейнерный элемент, определяющий группу меню;
* <item> — элемент, определяющий пункт меню

Элементы <item> и <group> могут быть дочерними элементами <group>.

Элемент **item** может иметь несколько атрибутов:

**id**

Идентификатор пункта меню, по которому приложение может распознать при выделении пункта меню пользователем

**title**

Текст, который будет выводиться в меню

**icon**

Значок для пункта меню. Можно использовать графический ресурс

Ресурсы разметки

Еще один из важных видов ресурсов - ресурсы разметки, которые отвечают за внешний вид приложения. Данные ресурсы представлены в формате XML. Ресурс разметки формы (layout resource) - это ключевой тип ресурсов, применяемый при программировании пользовательских интерфейсов в Android. Каждый ресурс, описывающий разметку, хранится в отдельном файле каталога **res/layout**. Имя файла без расширения выступает как идентификатор ресурса.

Ниже представлен фрагмент исходного кода для разметки:

setContentView(R.layout.main);

Строка **setContentView(R.layout.main);** указывает на то, что у нас имеется статический класс, называемый **R.layout**, в котором есть константа **main** (некое число), указывающая на **View**, определяемый в ХМL-файле ресурса разметки формы. ХМL-файл будет иметь имя main.xml, и он должен быть размещен в подкаталоге ресурсов res/layout и содержать необходимое определение разметки формы.

Содержимое самого файла **main.xml** может быть таким:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

>

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/hello"

/>

</LinearLayout>

В данном файле разметки формы имеется корневой узел <LinearLayout>, в котором содержится дочерний элемент <TextView>.

Для каждого варианта разметки требуется специальный файл. Если вы рисуете два экрана, вам понадобится два файла разметки, например, **res/layout/screenl\_layout.xml** и **res/layout/screen2\_layout.xml**. Каждый файл в подкаталоге **res/layout** генерирует уникальную константу на основе имени файла (без расширения). При работе с ресурсами разметки здесь важно количество файлов, а при работе со строковыми ресурсами важно количество отдельных строковых ресурсов внутри файлов. Например, если в подкаталоге **res/layout** у нас есть два файла **filel.xml** и **filе2.xml**, то в файле **R.java** будут содержаться следующие записи:

public static final int file1=0x7f030000;

public static final int file2=0x7f030001;

Элементы управления, которые используются в данных файла компоновки, например, **TextView**, будут доступны в коде через ID, генерируемым в **R.java**:

TextView tvInfo = this.findViewByld(R.id.textView);

В данном примере мы находим класс **TextView** при помощи метода **findViewByid()** класса **Activity**. Константа **R.id.textView** соответствует ID, заданному в **TextView**. Идентификатор для **TextView** в файле разметки выглядит следующим образом:

<TextView android:id="@+id/textView" />

Значение атрибута **id** указывает, что константа **textView** будет использоваться для идентификации этого вида среди других, используемых данным окном. Знак **+** в **@+id/textView** означает, что будет создан **ID** с именем **textView**, если он еще не существует.

Синтаксис ссылок на ресурсы

Все ресурсы Android идентифицируются по их **id**, содержащемуся в исходном коде Jаvа. Синтаксис, используемый при связывания id с ресурсом в файле XML, называется синтаксис ссылок на ресурс (resource-reference syntax). Синтаксис атрибута **id** в предыдущем примере **@+id/textl** имеет следующую структуру:

@[package:]type/name

Параметр **tуре** соответствует одному из пространств имен:

* R.drawable
* R.id
* R.layout
* R.string
* R.attr
* и др.

Параметр **name** - это имя ресурса. Оно также представлено в виде константы **int** в файле **R.java**.

Если не указывать пакет (package), то разрешение пары **type/name** будет производиться на основе локальных ресурсов и локального пакета **R.java**. Если указать **android:type/name**, то связывание ID ссылки будет производиться с применением пакета Android, в частности с использованием файла **android.R.java**. Вы можете использовать имя любого java-пакета вместо подстановочного слова **package**, чтобы использовать файл **R.java**, подходящий для связывания ссылки.

Рассмотрим несколько примеров

TextView android:id="text" - Ошибка компиляции, так как id не принимает необработанные текстовые строки.

TextView android:id="@text" - Неправильный синтаксис. Не хватает названия типа. Вы получите сообщение об ошибке "No Resource type specified"

TextView android:id="@id/text - Ошибка: не найдено ни одного ресурса, соответствующего id "text". Возможно, вы не задали "text" как один из видов ID

TextView android:id="@android:id/text" - Ошибка: Ресурс не является общедоступным". Означает, что такой id отсутствует в android.R.id. Чтобы такая запись была действительной, необходимо в файле Android.R.java задать id с таким именем

TextView android:id=="@+id/text - Успешно: создает id с названием "text" в файле R.java локального пакета.

Определение собственных идентификационных номеров ресурсов для последующего использования

Общий принцип присвоения **id** предполагает либо создание нового id для ресурса, либо использование одного из id, созданных в пакете Android. Однако **id** можно создавать и заранее, а потом использовать их в собственных пакетах.

Строка <TextView android:id="@+id/text"> в предыдущем фрагменте кода указывает, что **id** с названием **text**, будет использоваться в том случае, если этот **id** уже создан. Если **id** еще не существует, нужно создать новый идентификатор. В связи с этим возникает вопрос: может ли **id**, например, **text** уже существовать в файле **R.java**, чтобы такой идентификатор можно было использовать многократно?

Можно предположить, что такую задачу могла бы выполнять константа, например **R.id.text**, находящаяся в файле **R.java**, но **R.java** не поддается редактированию. Даже, если бы было возможно внести такие изменения, файл приходилось бы заново генерировать после добавления, изменения или удаления любой информации из подкаталога **res/\***.

Решение проблемы заключается в использовании тега ресурса под названием **item** для задания **id**, не связанного ни с каким конкретным ресурсом. Ниже приведен соответствующий пример:

<resources>

<item type="id" name="text "/>

</ resources>

Здесь **type** описывает тип ресурса, в данном случае **id**. Когда **id** будет установлен, будет работать и следующее определение **View**:

<TextView android:id="@id/text"/>

Цветовые ресурсы

Для работы со цветом используется тег **<color>**, а цвет указывается в специальных значениях.

* #RGB;
* #RRGGBB;
* #ARGB;
* #AARRGGBB;

Также существуют предопределенные названия цветов. Такие ID доступны в пространстве имен **android.R.соlor**. Посмотреть цветовые значения цветов можно в документации <http://developer.android.com/reference/android/R.color.html>.

Например, там есть оранжевый цвет **holo\_orange\_dark**, а также оранжевый-светлый, синий-темный, синий-светлый и т.д.

Обычно для цветовых ресурсов используют файл **colors.xml** в подкаталоге **res/values**. Но можно использовать любое произвольное имя файла, или даже вставить их в файл вместе со строковыми ресурсами **strings.xml**. Android прочтет все файлы, а затем обработает их, присвоив им нужные ID.

<?xml version="1.0" encoding="utf-8"?>

<resources>

<color name="red">#f00</color>

<color name="yellow">#FFFF00</color>

<color name="green">#FF00FF00</color>

</resources>

Для программного использования цветовых ресурсов можно использовать следующий код:

int myRedColor = activity.getResourses.getColor(R.color.red); // получаем значение красного цвета

linearLayout.setBackgroundResource(R.color.yellow); // устанавливаем фон в желтый цвет

linearLayout.setBackgroundResource(android.R.color.holo\_orange\_dark); // устанавливаем фон в оранжевый цвет

Более длинный вариант без склейки методов.

Resources res = getResources(); // получим объект для работы с ресурсами

int title\_color = res.getColor(R.color.title\_red); // получим значение цвета, заданного в ресурсах

При использовании в xml-файлах (например, файл разметки) используется следующий формат:

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="@string/anytext"

android:color="@color/red" />

Обратите внимание на использование префикса **@** для того, чтобы ввести ссылку ресурса — текст после этого префикса — имя ресурса. В этом случае мы не должны были указывать пакет, потому что мы ссылаемся на ресурс в нашем собственном пакете. Для ссылки на системный ресурс мы должны записать: **android:textColor="@android:color/black"** (чёрный цвет).

Ресурсы размеров

В Android используются следующие единицы измерения: пикселы, дюймы, точки. Все они могут входит в состав ХМL-шаблонов и кода Jаvа. Данные единицы измерения также можно использовать в качестве ресурсов при помощи тега <dimen> (обычно используют файл **dimens.xml**):

<resources>

<dimen name="in\_pixels">1px</dimen>

<dimen name="in\_dp">5dp</dimen>

<dimen name="in\_sp">100sp</dimen>

</resources>

Список разрешенных единиц измерения можно прочитать в отдельной статье [Android:: Единицы измерения](http://developer.alexanderklimov.ru/android/theory/scales.php)

Метод Java использует в названии целое слово **Dimension**, а в коде и в XML используется сокращенная версия **dimen**

Вы должны иметь доступ к каждому экземпляру объекта **Resources**, чтобы найти значения его параметров. Это можно сделать, применив метод **getResources()** к объекту **Activity**. Когда у вас будет объект **Resources**, его можно запросить по **id**, чтобы узнать значение этого параметра.

float dimen = activity.getResources().getDimension(R.dimen.in\_pixels);

В XML-файлах используется следующий синтаксис

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:textSize="@dimen/in\_dp"/>

Ресурсы визуальных стилей и тем

Ресурсы со стилями позволяют поддерживать единство внешнего вида приложения. Чаше всего визуальные стили и темы используются для хранения цветовых значений и шрифтов. Обычно используют **файл styles.xml**.

Чтобы создать стиль, используйте тег <style>, включающий атрибут name, а также один или несколько вложенных узлов item. Каждый тег **item** в свою очередь также должен иметь атрибут **name**, содержащий тип описываемого значения (например, размер шрифта или цвет). Внутри тега должно находиться само значение.

<style name="StyleName">

<item name="attributeName">value</item>

</style>

Тег <style> поддерживает наследование с помощью атрибута **parent**, поэтому вы можете свободно варьировать стили.

res/drawable

В папке **res/drawable** могут содержаться ресурсы изображений, фигур, градиентов.

Примеры с [с ресурсами подобного типа приведены в отдельной статье](http://developer.alexanderklimov.ru/android/theory/drawable.php).

Ресурсы отрисовываемых цветов

Кроме обычных изображений в Android поддерживается и другой тип отрисовываемых ресурсов, которые называются отрисовываемыми цветами (color-drawable). По сути, это просто цветные прямоугольники.

Чтобы задать цветной прямоугольник, нужно указать тег <drawable> в имени узла XML-файла, находящегося в подкаталоге **res/values/**.

<resources>

<drawable name="black\_rectangle">#000000</drawable>

<drawable name="white\_rectangle">#ffffff</drawable>

</resources>

В XML-шаблонах ресурсы используются следующим образом

<TextView

android:layout\_width="fill\_parent"

...

android:background="@drawable/white\_rectangle" />

Программным способом:

// Получение отрисовываемого объекта

ColorDrawable whiteDrawable =

(ColorDrawable)activity.getResources().getDrawable(R.drawable.white\_rectangle);

// установление его в качестве фона для текстового вида

textView.setBackground(whiteDrawable);

Mipmap

С появлением Nexus 6 и Nexus 9, Гугл добавила в Android новые типы ресурсов - **Mipmap**. По сути, это замена ресурсам **Drawable** для значков приложения. Нужно подготовить значки и расположить их в папках **mipmap-mdpi**, **mipmap-hdpi** и т.д.

Ресурсы анимации

Android поддерживает два типа анимации. Первый тип основан на расчете промежуточных кадров и может использоваться для поворота, перемещения, растягивания, затемнения элементов. Второй тип - пошаговая анимация, т.е. последовательный вывод заранее подготовленных изображений.

При использовании анимации промежуточных кадров каждый экземпляр анимации хранится в отдельном XML-файле внутри каталога **res/anim**. Имена файлов без расширения являются идентификаторами для ресурсов.

Анимацию можно задать в виде изменений параметров **alpha** (затемнение), **scale** (масштабирование), **translate**(перемещение) или **rotate** (вращение).

Вы можете комбинировать различные экземпляры анимации, используя тег **set** с атрибутами:

* duration - продолжительность анимации в секундах
* startOffset - миллисекундная задержка перед началом анимации
* fillBefore - при значении true преобразование происходит перед началом анимации
* fillAfter - при значении true преобразование происходит после завершения анимации
* interpolator - описывает изменения в скорости эффекта

Если не использовать атрибут **startOffset**, все анимационные эффекты внутри набора происходят одновременно.

Пошаговая анимация подразумевает создание последовательности объектов **Drawable**, каждый из которых будет отображаться в качестве фона на протяжении указанного промежутка времени. Пошаговая анимация хранится в виде ресурсов **Drawable** в каталоге **res/drawable** (см. выше). Имена файлов без расширения используются в качестве идентификаторов.

Ресурсы массивов

Существует еще один тип ресурсов для хранения значений массивов. Принято хранить данные ресурсы в файле **arrays.xml** папки **res/values**. Вот как может выглядеть файл:

<?xml version="1.0" encoding="utf-8"?>

<resources>

<array name="choose">

<item>@string/red\_pill\_label</item>

<item>@string/blue\_pill\_label</item>

</array>

<string-array name="catnames">

<item>Рыжик</item>

<item>Барсик</item>

<item>Мурзик</item>

</string-array>

<integer-array name="years">

<item>2009</item>

<item>2010</item>

<item>2011</item>

</integer-array>

</resources>

Как видите, есть типы ресурсов **array**, **string-array**, **integer-array**.

Программно получить доступ к ресурсам массива можно так:

// загрузка массива строк из res/values/arrays.xml в текстовое поле textStrings

String[] names = getResources().getStringArray(R.array.names);

for(int i = 0; i < names.length; i++) {

textStrings.append("Name[" + i + "]: "+ names[i] + "\n");

}

// загрузка массива целых чисел из res/values/arrays.xml в текстовое поле textDigits

int[] digits = getResources().getIntArray(R.array.digits);

for(int i = 0; i < digits.length; i++) {

textDigits.append("Digit[" + i + "]: "+ digits[i] + "\n");

}

Ещё один вариант, который может вам встретиться. Цвета заданы в массиве и доступ к ним через особый объект **TypedArray**

<array name="rainbow">

<item>#ff0000</item>

<item>#ff9900</item>

<item>#ffff00</item>

<item>#00ff00</item>

<item>#0000ff</item>

<item>#0099ff</item>

</array>

TypedArray arrayColors = getResources().obtainTypedArray(R.array.rainbow);

ImageView imageView = findViewById(R.id.imageView);

imageView.setBackgroundColor(arrayColors.getColor(1, 0xFF000000));

arrayColors.recycle();

Скомпилированные и нескомпилированные ресурсы Android

Большинство ресурсов компилируются в двоичные файлы, но некоторые используются без дополнительной обработки. В Android поддержка ресурсов осуществляется преимущественно при помощи файлов двух типов - XML и RAW-файлов (к последним относятся изображения, аудио и видео). При работе с ХМL-файлами мы видели, что в большинстве случаев ресурсы определяются как значения внутри файла XML (это касается, например, строк), а иногда весь ХМL-файл является ресурсом (например, файл ресурса разметки).

Файлы, созданные в XML, также подразделяются на два типа: первые компилируются в двоичный формат, а вторые копируются на устройство без изменений. Например, XML-файлы строковых ресурсов и ресурсов разметки компилируются в двоичный формат. Эти ХМL-файлы имеют заданный формат, в котором узлы XML преобразуются в ID.

XML-документы

Вы также можете специально выбрать некоторые ХМL-файлы и задать для них структуру формата, отличающуюся от принятой по умолчанию, - чтобы эти файлы не интерпретировались и для них не генерировались идентификаторы ресурсов. Однако в данном случае нам как раз нужно, чтобы они компилировались в двоичный формат и чтобы их было удобно локализовывать. Для достижения этой цели файлы XML можно поместить в подкаталог **/res/xml** - тогда они будут скомпилированы в двоичный формат. В таком случае вы можете воспользоваться поставляемыми вместе с Android инструментами для считывания XML, которые могут считывать ХМL-узлы.

Ниже приведен пример ХМL-файла в подкаталоге **res/xml/test.xml**:

<rootelement>

<subelement>

Коты - наше всё!

</subelement>

</rootelement>

Как и при работе с другими ХМL-файлами ресурсов, ААРТ компилирует такой ХМL-файл, перед тем как поместить его в пакет прикладных программ. Для синтаксического разбора подобных файлов используйте экземпляр **XmlPullParser**:

Resources res = activity.getResources();

XmlResourceParser хрр = res.getXml(R.xml.test);

Возвращенный **XmlResourceParser** является экземпляром **XmlPullParser**, а также реализует **java.util.AttributeSet**.

Пример работы с XML-документами описан в статье [Работаем с XML-документами](http://developer.alexanderklimov.ru/android/theory/xmlresourceparser.php)

Использование необработанных ресурсов RAW

Если разместить файлы, в том числе написанные на XML, в каталоге **res/raw**, они не будут скомпилированы в двоичном формате, а попадают в пакет прикладных программ как есть. Для считывания таких фалов нужно использовать явные API с поддержкой потоков. К категории **raw** относятся аудио- и видеофайлы.

Каждый такой файл, помещенный в папку **res/raw**, имеет свой идентификатор, генерируемый в **R.java**. Чтобы получить доступ к ресурсам, предназначенным только для чтения, вызовите метод **openRawResource()**, принадлежащий объекту **Resource** приложения. Таким образом, вы получите объект **InputStream**, основанный на указанном файле. В качестве имени переменной, принадлежащей **R.raw**, задайте имя файла (без расширения). Если бы вы поместили текстовый файл в **res/raw/test.txt**, то его можно было бы прочитать при помощи следующего кода, используя идентификатор **test**:

public void onClick(View view) {

TextView infoTextView = findViewById(R.id.textViewInfo);

infoTextView.setText(getStringFromRawFile(MainActivity.this));

}

private String getStringFromRawFile(Activity activity) {

Resources r = activity.getResources();

InputStream is = r.openRawResource(R.raw.test);

String myText = null;

try {

myText = convertStreamToString(is);

} catch (IOException e) {

e.printStackTrace();

}

try {

is.close();

} catch (IOException e) {

e.printStackTrace();

}

return myText;

}

private String convertStreamToString(InputStream is) throws IOException {

ByteArrayOutputStream baos = new ByteArrayOutputStream();

int i = is.read();

while( i != -1)

{

baos.write(i);

i = is.read();

}

return baos.toString();

}

Мы уже рассматривали структуру папок в каталоге **res**. Компилятор ресурсов, входящий в состав инструмента Android Asset Packaging Tool (ААРТ), собирает все ресурсы, кроме **raw**, и помещает все их в итоговый файл АРК. Этот файл, содержащий код и ресурсы приложения Android, аналогичен файлу JAR, который применяется в Java (АРК означает Android package - пакет Android). Именно файл АРК устанавливается на устройство.

Хотя инструмент синтаксического разбора (parser) ресурсов XML допускает такие имена ресурсов, как hello-string, в файле R.java во время компиляции такое название вызовет ошибку. Для ее устранения можно переименовать ресурс в hello\_string, заменив дефис нижним подчеркиванием.

Имена файлов, в которых дублируются базовые имена, вызывают ошибку компиляции (build error). Так происходит со всеми идентификаторами ресурсов, которые сгенерированы для ресурсов, созданных на основе файлов.

Использование ресурсов в коде программы

Подведем воедино информацию об использовании ресурсов в коде программы.

Во время компиляции генерируется статический класс **R** на основе ваших ресурсов и содержит идентификаторы всех ресурсов в программе. Класс **R** имеет несколько вложенных классов, один для каждого типа ресурса, поддерживаемого системой Android, и для которого в проекте существует файл ресурса. Класс **R** может содержать следующие вложенные классы:

* R.anim — идентификаторы для файлов из каталога res/anim/ (анимация);
* R.array — идентификаторы для файлов из каталога res/values/ (массивы);
* R.bool — идентификаторы для битовых массивов в файлах arrays.xml из каталога res/values/;
* R.integer — идентификаторы для целочисленных массивов в файлах arrays.xml из каталога res/values/;
* R.color — идентификаторы для файлов colors.xml из каталога res/values/ (цвета);
* R.dimen — идентификаторы для файлов dimens.xml из каталога res/values/ (размеры);
* R.drawable — идентификаторы для файлов из каталога res/drawable/ (изображения);
* R.id — идентификаторы представлений и групп представлений для файлов ХМL-разметки из каталога res/layout/;
* R.layout — идентификаторы для файлов разметки из каталога res/layout/;
* R.raw — идентификаторы для файлов из каталога res/raw/;
* R.string — идентификаторы для файлов strings.xml из каталога res/values/ (строки);
* R.style — идентификаторы для файлов styles.xml из каталога res/values/ (стили);
* R.xml — идентификаторы для файлов из каталога res/xml/.

Синтаксис для обращения к ресурсу:

R.resource\_type.resource\_name

При использовании системных ресурсов используется класс **android.R**.

android.R.drawable.sym\_def\_app\_icon; // стандартный значок приложения

android.R.style.Theme\_Black; // загрузить стандартный стиль:

Если в коде вам понадобится идентификатор ресурса для конструктора или метода, то можете использовать данные свойства:

setContentView(R.layout.activity\_main); // загрузка ресурса разметки

// Строковый ресурс используется при выводе Toast-сообщения

Toast.makeText(this, R.string.mytext, Toast.LENGTH\_LONG).show();

Если вам нужен не идентификатор, а сам экземпляр ресурса, то используйте метод **getResources** для доступа к экземпляру класса **Resources**:

Resources myResources = getResources();

Принцип работы класса **Resources** заключается в передаче идентификатора ресурса, чей экземпляр вам нужен для работы. Вот несколько примеров получения экземпляров ресурсов:

Resources myResources = getResources();

CharSequence myText = myResources.getText(R.string.hello\_app); // получим строку

Drawable myIcon = myResources.getDrawable(R.drawable.app\_icon); // получим значок

int myColor = myResources.getColor(R.color.color\_blue); // получим значение цвета

float myWidth = myResources.getDimension(R.dimen.width\_border); // получим размер

// Получим массив строк

String[] stringArray;

stringArray = myResources.getStringArray(R.array.string\_array);

// Ресурс, содержащий пошаговую анимацию, возвращается в виде объекта AnimationResources

// Также можно вернуть значение с помощью метода getDrawable и привести к нужному типу

AnimationDrawable cat;

cat = (AnimationDrawable)myResources.getDrawable(R.drawable.frame\_cat);

Итак, к графическим ресурсам можно обратиться через **R.drawable.cat** (файл cat.png), а к музыкальным трекам через **R.raw.meow** (файл meow.mp3) и по аналогии с другими типами ресурсов.

Иногда требуется большей гибкости при использовании файлов из ресурсов. Например, в приложении нужно использовать имена ресурсов. В этом случае есть два подхода для получения информации о ресурсе.

Первый способ:

android.resource://[package]/[res type]/[res name]

Uri.parse("android.resource://ru.alexanderklimov.test/raw/filename");

Второй способ с использованием идентификатора (не особо нужен):

android.resource://[package]/[resource\_id]

Uri.parse("android.resource://ru.alexanderklimov.test/" + R.raw.filename);

Допустим, у вас файлы имеют схожие имена **meow1.mp3**, **meow2.mp3**, **meow3.mp3**. По приведённой выше схеме не составит труда создать переменную типа **String fileName = "meow" + n**. Такой подход может пригодиться в циклах, когда счётчик можно сопоставить с именем файла.

Можете заменить имя пакета на программное извлечение с помощью метода **Context.getPackageName()**.

Ещё один полезный способ, который может пригодиться. Получить идентификатор по имени файла (без расширения) при помощи метода **getIdentifier()**:

int resourceID = this.getResources().getIdentifier("filename",

"raw", this.getPackageName());

mediaPlayer = MediaPlayer.create(getApplicationContext(),

resourceID);

Пример для графического ресурса.

String mDrawableName = "cat1"; // файл cat1.jpg в папке drawable

int resID = getResources().getIdentifier(mDrawableName, "drawable", getPackageName());

Иногда нужно получить не сам идентификатор, а его имя, чтобы сохранить его, скажем, в базе данных. Воспользуйтесь следующим приёмом, используя метод**getResourceEntryName()**:

// вернёт cat1

Log.i("id", this.getResources().getResourceEntryName(R.drawable.cat1));

Или так

// вернет ru.alexanderklimov.test:drawable/cat1

Log.i("id", getResources().getResourceName(R.drawable.cat1));

// вернёт res/drawable/cat1.jpg

Log.i("id", getResources().getString(R.drawable.cat1));

// вернёт тот же результат

Log.i("getText()", this.getResources().getText(R.drawable.cat1).toString());

Вложенные ресурсы

Можно использовать ссылки на ресурсы в качестве значений для атрибутов внутри других ресурсов (разметка, стили), что позволяет создавать специальные варианты визуальных тем, локализованных строк и графических объектов. Чтобы сослаться на один ресурс внутри другого, используйте символ @ в следующем виде:

attribute = "@[packagename:]resourcetype/resourceID"

Полное имя **packagename** нужно указывать только при использовании сторонних пакетов, при использовании собственных ресурсов данный параметр можно опустить, так как Android предполагает, что вы используете ресурсы пакета со своим приложением.

...

android:text="@string/hello\_app"

android:textColor="@color/color\_blue"

...

Использование системных ресурсов

Сама система имеет собственные ресурсы (строки, изображения, анимация, разметки, стили), которые используются стандартными приложениями, входящими в состав Android. Вы также можете использовать данные системные ресурсы в своих приложениях, добиваясь единообразного стиля и дизайна.

Получение доступа к системным ресурсам внутри кода программы ничем не отличается от приведенных выше примеров. Единственно, в чем состоит отличие, это использование не вашего класса R, а системного класса **android.R**. Например, для получения строки, которая хранит сообщение об ошибке, используется следующий код:

CharSequence errorMessage = getString(android.R.string.httpErrorBadUrl);

Чтобы получить доступ к системным ресурсам внутри XML-файла, используйте значение **android** следующим образом:

...

android:text="@android:string/httpErrorBadUrl"

android:textColor="@android:color/darker\_gray"

...

Использование ссылок на визуальные стили в текущей теме

Использование визуальных тем позволяет обеспечить целостность пользовательского интерфейса приложения. Вместо описания каждого стиля, вы можете использовать ссылки, предоставляемые Android, с помощью которых вы можете использовать стили из текущей темы.

Чтобы задать ссылку на ресурс, который нужно применить, вместо символа @ укажите префикс **?**. Например, предыдущий пример можно переделать следующим образом:

android:textColor="?android:textColor"

android:background="?colorPrimary"

Такой подход позволяет создавать визуальные стили, которые смогут меняться по мере преобразований в текущей теме, и избавляет от необходимости редактировать каждый отдельный ресурс.

Создание ресурсов для локализации и аппаратных конфигураций

Применение ресурсов позволяет использовать механизм динамического выбора нужного ресурса в программе. Можно задать определенную структуру каталогов в проекте, чтобы создавать ресурсы для конкретных языков, регионов и аппаратных конфигураций. Во время выполнения программы Android выберет нужные значения для конкретного телефона пользователя.

Вы можете указывать альтернативные значения, создавая собственные структуры каталогов внутри каталога **res** при помощи дефиса (-). Например, мы хотим создать дополнительные строковые ресурсы для французского языка, франкоканадского региона и для русского языка. Тогда структура каталогов в проекте будет выглядять следующим образом:

Project/

res/

values/

strings.xml

values-fr/

strings.xml

values-fr-rCA

strings.xml

values-ru/

strings.xml

Как видите, мы создали несколько файлов strings.xml, которые содержат текст на французском и русском языках и раскидали их по нужным каталогам. Далее приводится список возможных идентификаторов, которые можно использовать для создания альтернативных значений в ресурсах.

Регионы и язык можно указывать без всякой связи друг с другом. Так в папке values-ru-rJP будут храниться русские тексты для жителей Японии.

* **Мобильный код страны и код мобильного оператора (MCC/MNC)** - содержит информацию о стране и опционально о мобильной сети, которая привязана к SIM-карте. MCC (Mobile Country Code) состоит из символов mcc, за которыми следует трехзначный код страны. Также можно добавить MNC (Mobile Network Code), используя символы mnc и двухзначный код мобильной сети, например, **mcc250-mnc99** (Россия-Билайн). Список кодов MCC/MNC можно посмотреть в Википедии.
* **Язык и регион** - указывает на язык при помощи языкового кода в формате ISO 639-1. Состоит из двух символов в нижнем регистре. В случае необходимости можно добавить обозначение региона в виде символа **r** и двухсимвольного кода в формате ISO 3166-1-alpha-2, записанного в верхнем регистре, например, en-rUS, en-rGB, fr-rCA.
* **Размер экрана** - может иметь одно из следующих значений:
  + small - меньше, чем HVGA, 3.2 дюйма
  + medium - HVGA или меньше, чем HVGA, типичный размер
  + large - VGA или больше, планшет или нетбук
* **Высота и ширина экрана** - можно использовать значения
  + long — для экранов, которые в альбомном режиме значительно шире, чем на стандартных смартфонах (таких как G1);
  + notlong — для экранов с обычным соотношением сторон
* **Ориентация экрана** - возможны значения port (Портретный), land (Альбомный), square (Квадратные экраны).
* **Плотность пикселов на экране** - возможны значения:
  + ldpi — предназначен для хранения ресурсов, рассчитанных на экраны с низкой плотностью пикселей (100–140 dpi);
  + mdpi — для экранов со средней плотностью пикселей (140–180 dpi);
  + hdpi — для экранов с высокой плотностью пикселей (190–250 dpi);
  + xdpi - новый тип для очень высокой плотности для планшетов
  + nodpi - Вы можете использовать значение nodpi для растровых ресурсов, которые не должны масштабироваться. В этом случае система не требует точного совпадения.

Подбирая подходящий каталог, Android выберет тот спецификатор, который наиболее точно описывает плотность пикселей экрана устройства и откорректирует масштаб объекта Drawable. По умолчанию, в новом проекте создаются папки drawable-ldpi, drawable-mdpi, drawable-hdpi, drawable-xhdpi, в которых содержатся значки для приложения.

* **Тип сенсорного экрана** - возможны варианты: notouch, stylus, finger.
* **Наличие клавиатуры** - возможны варианты: keysexposed, keyshidden, keysoft.
* **Тип ввода** - возможны варианты: nokeys, qwerty, 12key.
* **Способ навигации** - возможны варианты: nonav, dpad, trackball, wheel.

Вы можете комбинировать указанные спецификаторы, разделяя их дефисами. Поддерживаются любые сочетания, однако они должны идти в том порядке, как указано в списке, т.е. язык всегда указывается раньше ориентации экрана и т.п. В одном спецификаторе может применяться не более одного значения.

Корректные варианты

drawable-en-rUS

drawable-en-keyshidden

drawable-long-land-notouch-nokeys

Некорректные варианты

drawable-rUS-en (нарушен порядок)

drawable-nonav-dpad (несколько значениф для одного спецификатора)

Если на устройстве не будет обнаружено соответствующих ресурсов, при попытке получения доступа к ним ваша программа сгенерирует исключение. Чтобы избежать подобной ситуации, необходимо всегда добавлять значения по умолчанию для всех типов ресурсов в каталог без спецификаторов.

Пример с локализацией программы можно посмотреть в статье [Локализация приложений](http://developer.alexanderklimov.ru/android/locale.php).

Псевдонимы (alias)

Чтобы избежать дублирования ресурсов, можно использовать псевдонимы, которые будут ссылаться на один и тот же ресурс. Предположим вы создали два файла **res/layout-land/activity\_main.xml** и **res/layout-large/activity\_main.xml** с одинаковым содержанием для разметки с альбомной ориентацией для смартфонов и планшетов. Создайте теперь ещё один файл с таким же содержанием, например под именем **res/layout/activity\_main\_horizontal.xml**. Теперь два одинаковых файла можете удалить. Вместо них создайте два файла **res/values-land/refs.xml** и **res/values-large/refs.xml**.

<resources>

<item type="layout" name="activity\_main">@layout/activity\_main\_horizontal</item>

</resources>

Такой приём часто применяется при использовании фрагментов.

Получение идентификаторов ресурсов приложения

Врядли вам пригодится этот код в жизни. Для общего развития.

public void getAllResourceId() {

final R.id idResources = new R.id();

final Class<R.id> c = R.id.class;

final java.lang.reflect.Field[] fields = c.getDeclaredFields();

for (Field field : fields) {

final int resourceId;

try {

resourceId = field.getInt(idResources);

Log.i("ID", "resourceId: " + resourceId + " Name:" + field.getName());

} catch (Exception e) {

Log.e("Exception", e.getLocalizedMessage());

}

}

}