Работа со строковыми ресурсами Android

Понимание, что используемый в программе текст это отдельный ресурс, такой же как изображения и звук, приходит на сразу. Но стоит несколько раз поменять имя программы в паре десятков файлов или заняться исправлением однотипной синтаксической ошибки в пяти, шести разных местах и необходимость хранить строки отдельно от кода становиться очевидной.  
В Android работа со строковыми ресурсами сделана очень удобна и не вызывает поначалу никаких сложностей. В официальной документации она описана в статье [String Resources](http://developer.android.com/guide/topics/resources/string-resource.html). В файле *project\res\values\strings.xml* задаем строку и ее имя после чего в Activity загружаем строку по этому имени.  
*strings.xml*

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

<resources>

<string name="appName">Project Name</string>

</resources>

*ProjectActivity.java*

public class LoadingActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

String applicationName = getString(R.string.appName);

}

*//...*

}

Эту же строку можно использовать в xml файле разметки формы (layout resource). Как это реализовано можно посмотреть, создав новый проект и открыв файл *main.xml*.  
  
*main.xml*

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

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

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent" >

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

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

</LinearLayout>

Описание ресурсов

Теперь несколько более интересных вещей, которые тоже описаны в документации, но используют которые уже не так широко.  
Файлов со строками (впрочем как и с остальными ресурсами) может быть несколько. Главное, чтобы они находились в папке *project\res\values\*, имели расширение *xml*, имя состояло из строчных английских букв, цифр и не содержало пробелов. Внутреняя структура должна повторять структуру файла *strings.xml*. Для чего это может использоваться? Я в одном файле храню все строки, которые надо будет переводить, во втором строки, которые не требуют перевода, в третьем храню константы, которые использую для составления запросов к веб серверу.  
XML файл имеет небольшие ограничения на формат хранимых данных. В открытом виде в нем нельзя использовать символы '&', '<'. Задать эти символы можно используя специальную последовательность  
< — &lt;  
& — &amp;  
Дополнительные ограничения накладываются на работу с одинарными и двойными апострофами: использовать непарный апостроф просто так нельзя. Есть несколько методов решения этой проблемы, самым простым из них является — добавление обратного слеша перед знаком апострофа:  
' — \' или &apos;  
" — \" или &quot;  
Если текст содержит html теги и в нем встречается неразрывный пробел &nbsp;, то его надо заменить на &#160;.

Загрузка строк

Если надо задать текст из ресурсов одному из элементов интерфейса, то нет необходимости заранее его загружать. Вторая и третья строка в приведенном примере работают совершенно одинаково.

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

sectionHeader.setText(getString(R.string.sectionPhone));

sectionHeader.setText(R.string.sectionPhone);

Метод *getString* удобно использовать, когда в строку из ресурсов нужно внести дополнительные данные перед дальнейшим использованием. В этом случае в ресурсы помещается форматированная строка и дополнительные параметры для нее указываются прямо в *getString*, без дополнительного обращения к методу *String.format*. Примеры разделенные чертой приводят к одним и тем же результатам:

int sectionId = 10;

String header = getString(R.id.headerSection);

sectionHeader.setText(String.format(header, sectionId));

*//-----------------*

sectionHeader.setText(getString(R.id.headerSection, sectionId));

Для формата строки используется следующая конструкция: %X$F.  
X$ — номер подставляемого параметра. В основном тексте они обычно идут по порядку 1$, 2$, но в локализованных ресурсах могут меняться местами. Также позволяется использовать один парметр в строке несколько раз.  
F — обычный идентификатор формата, такой 's', 'd'. Их полный список, включая форматрирование даты, описан в документации класса [Formatter](http://developer.android.com/reference/java/util/Formatter.html)  
Если в строке используется только один параметр, то X$ можно опустить. Если считать параметры неудобно, а проблем с локализацией не предвидится, то можно вернуться к стандартной схеме формата строки — для этого в описание элемента нужно добавить атрибут *formatted* со значением *false*. Следующие две строки форматируют текст одинаковым образом:

<string name="score\_correct">%1$d - %2$d</string>

<string name="score\_simple" formatted="false">%d - %d</string>

Загрузить строку по ее имени можно также как и любой другой ресурс. Для этого сначала нужно с помощью метода *getIdentifier* по имени строки найти ее *id*, а с этим номером уже работать обычным сопособом. Следующий пример загружает строку с именем «score\_correct».

int strId = getResources().getIdentifier("score\_correct", "string", getPackageName());

String strValue = getString(strId);

Android позволяет хранить в ресурсах массивы строк. Для этого используется тег *string-array*, который содержит внутри элементы *item* с конкретными строками. Вот сокращенный пример из документации Android, который иллюстрирует задание массива.  
*strings.xml*

<string-array name="planets\_array">

<item>Earth</item>

<item>Mars</item>

</string-array>

Кроме очевидного применения — удобная загрузка данных, *string-array* часто используют для иницализации UI элементов с выпадающим списком значений: *Spiner* и *ListPreference*. В этом случае обычно требуется использовать одну из строк массива, как значение по умолчанию. Сослаться в ресурсах на конкретный элемент массива мы не можем, но из ситуации можно выйти задействовав [псевдонимы для ресурсов](http://developer.android.com/guide/topics/resources/providing-resources.html#AliasResources).  
Элементы массива инициализируются, как обычные строки, а элементы *item* содержат только ссылку на них. Такая инициализация на самом деле черезвычайно удобна и я использую ее даже в тех случаях, когда обращение к конкретной строке не планируется — это гарантирует, что при локазации все массивы будут одинакового размера, даже если часть строк в них не будет переведена. Само описание массива при этом удобно вынести в отдельный ресурсный файл.  
Переписанный с использованием псевдонимов пример выглядит так:  
  
*strings.xml*

<string name="earth">Earth</string>

<string name="mars">Mars</string>

<string-array name="planets\_array">

<item>@string/earth</item>

<item>@string/mars</item>

</string-array>

Полезное замечание: до версии Android 2.3 в реализации загрузки string-array была ошибка, которая позволяла загружать максимум только 512 элементов.

Системные строки

Android дает доступ к нескольким строкам, хранящимся в системных ресурсах. Часть из них довольно специализированы: заголовок сообщения об ошибке проигрывания видео. Но такие строки как «Ok» или «Cancel» удобно использовать практически в каждом проекте. Использовать системные строки почти также легко, как и свои собственные ресурсы — надо перед идентификатором ресурса *string*через двоеточие указать имя *android*. Следующий пример описывает кнопку с надписью «Cancel» с использованием строки из системных ресурсов.  
  
*main.xml*

<Button

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

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

Локализация строк

Базовый механизм локализации ресурсов в Android позволяет легко решить практически все задачи с локализацией строк. Полное описание в документации Android можно прочесть [здесь](http://developer.android.com/guide/topics/resources/localization.html#strategies). В ресурсах вам необходимо создать новую папку с именем *values-xx* и поместить туда файлы со строковыми ресурсами из базовой папки *values*. *xx* — это двухсимвольный идентификатор языка (список поддерживаемых значений приведен в конце текста). После этого необходимые строки нужно перевести, остальные удалить.  
Для более тонкой локализации можно задействовать механизм задания региона. К примеру, у вас в программе есть соглашение пользователя, переведнное на французкий язык, но в нем есть отличия между версией для Франции и версией для Канады. Чтобы реализовать эти отличия надо к папке *values-fr* добавить название региона в формате *rYY*, где *YY* — это двухсимвольное название региона. В данном примере получатся папки *values-fr-rFR* и *values-fr-rCA*. В них следует поместить ресурсный файл с необходимой версией соглашения пользователя, а все остальные строки на французском языке оставить в папке *values-fr*.  
Регионы и язык можно указывать без всякой связи друг с другом. Так в папке *values-ru-rJP* будут храниться русские тексты для жителей Японии.  
  
Так как регионов определенно немного, то в большинстве случаев вместо них удобнее указывать точный идентификатор страны через свойство *mcc* (мобильный код страны). Список кодов доступен в [Википедии](http://ru.wikipedia.org/wiki/Mobile_Country_Code) и единственный недостаток данного метода — это то, что некоторые страны имеют несколько кодов (США используют 7 номеров, Япония — 2 номера). Переделав пример на использование *mcc* получаем папки *values-mcc208-fr* для Франции и *values-mcc302-fr* для Канады.  
Сложности с локализацией начинаются в тот момент, когда вам нужно локализовать приложение на тот язык, который не поддеживается системой. Попасть в такую ситуацию легко, потому, что большинство языков и регионов были добавлены только в 2.3 версии платформы. При этом некоторые производители добавляли поддержку других языков в своих устройствах, но использовали при этом разные коды. В итоге Норвежский язык в Android 2.1 и 2.2 распознается на многих устройствах по коду *no*, на отдельных не распознается, а начиная с Android 2.3 распознается по коду *nb*. Такая же ситуация, но меньшего масштаба, с Ивритом, который может встречаться под кодом *he* и Индонезийским языком — код *id*.  
В этом случае, чтобы не хранить два набора одинаковых строк в разных файлах, можно задействовать псевдонимы для ресурсов, которые уже упоминались раньше.  
В этом случае реальный перевод строк храниться в папке *values*, при этом для их названия используются рабочие имена. В папках *values-no* и *values-nb* хранятся ссылки на строки и тут уже используются настоящие названия ресурсов.  
  
*values/strings.xml*

<resources>

*<!-- Текст по умолчанию -->*

<string name="hello">Hello World</string>

*<!-- Переведенный текст -->*

<string name="hello\_translate">Hallo Verden</string>

</resources>

*values-no/strings.xml*

<resources>

<string name="hello">@string/hello\_translate</string>

</resources>

*values-nb/strings.xml*

<resources>

<string name="hello">@string/hello\_translate</string>

</resources>

Вы можете создавать ресурсы с какими угодно кодами для языка и для региона. Если операционная система не найдет нужного региона, то она возмет значения для текущего языка. Если не сможет найти язык, то возмет значения по умолчанию из папки *values*.  
  
Можно загрузить находящиеся в ресурсах строку для языка и региона отличных от установленных на устройстве. Для этого надо создать новый ресурс и задать ему необходимую локаль. Следующий пример загружает строку для французкого языка.

Resources baseResources = getResources();

Configuration config = new Configuration(baseResources.getConfiguration());

config.locale = Locale.FRANCE;

Resources localResources = new Resources(baseResources.getAssets(), baseResources.getDisplayMetrics(), config);

String strFranceValue = localResources.getString(R.string.score\_correct);

Вы также можете программно задать произвольный язык для всего приложения, но рассмотрение этой задачи выходит за рамки рассматриваемой темы.