대학생을 위한 안드로이드 애플리케이션 교육 자료 – 11

# 5,6강 복습 / Native Development Kit

www.kandroid.org 운영자 : 양정수 (yangjeongsoo@gmail.com), 닉네임:들풀

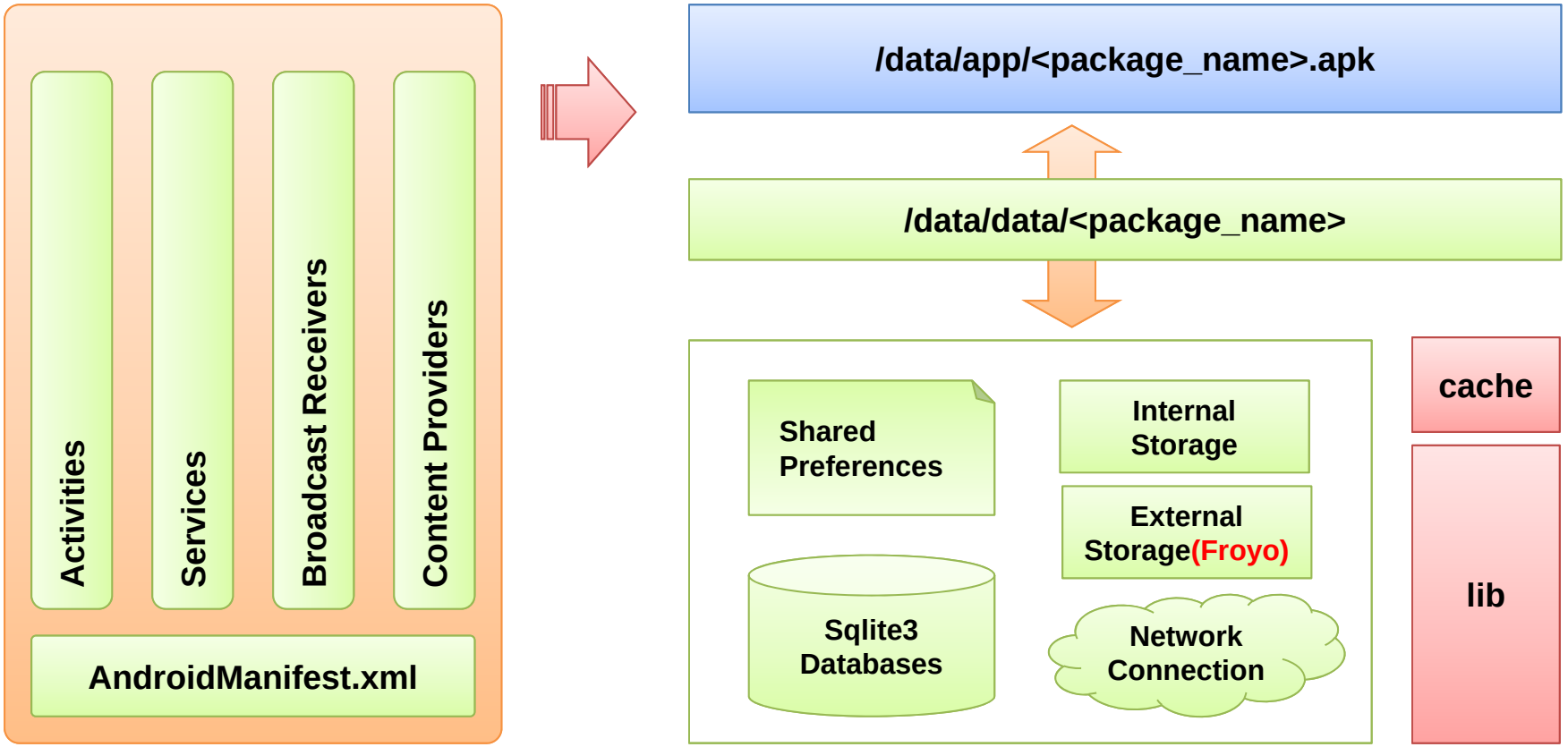
1. Data Storage

2. ContentProviders

# Data Storage : Overview

**Android File System Exploring**

1. > adb.exe shell
   > ls /data/app
2. eclipse > DDMS > File Explorer

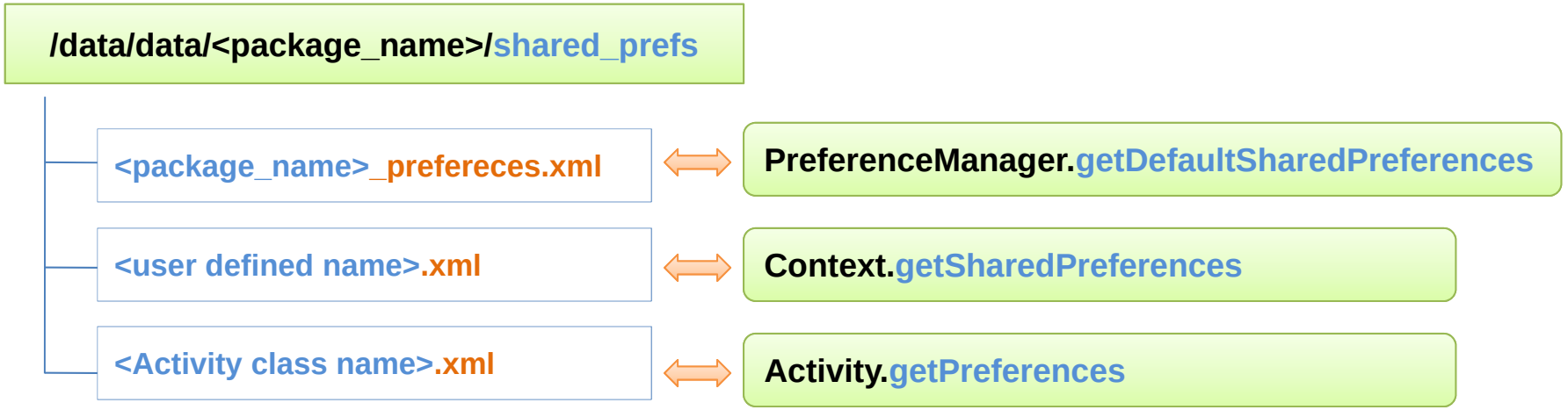


## Using Shared Preferences

**Preferences is a lightweight mechanism to store and retrieve key-value pairs of primitive data types. It is typically used to store application preferences, such as a default greeting or a text font to be loaded whenever the application is started. Call Context.getSharedPreferences() to read and write values.**

**Assign a name to your set of preferences if you want to share them with other components in the same application,**
**or**
**use Activity.getPreferences() with no name to keep them private to the calling activity.**

**You cannot share preferences across applications (except by using a content provider).**



## Using Shared Preferences

```
@Override
public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SharedPreferences prefs=
                PreferenceManager.getDefaultSharedPreferences(this);

        // Write Preference File
        SharedPreferences.Editor editor = prefs.edit();
        editor.putString(“key_1", "value_1");
        editor. putBoolean(“key_2", true);
        editor.commit();

        // Read Preference File
        Boolean prefs_bool = prefs.getBoolean(“key_1", false);
        String prefs_str = prefs.getString(“key_2", "<unset>");

}
```

## Using the Internal Storage

**You can store files directly on the mobile device or on a removable storage medium. By default, other applications cannot access these files.**

**To read data from a file, call Context.openFileInput() and pass it the local name and path of the file. It returns a standard Java FileInputStream object.**

**To write to a file, call Context.openFileOutput() with the name and path. It returns a FileOutputStream object. Calling these methods with name and path strings from another application will not work; you can only access local files.**

**If you have a static file to package with your application at compile time, you can save the file in your project in res/raw/myDataFile, and then open it with Resources.openRawResource (R.raw.myDataFile). It returns an InputStream object that you can use to read from the file.**

## Using the Internal Storage

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

        try {
            InputStream is = getAssets().open("sample.txt");
            int size = is.available();
            byte[] buffer = new byte[size];
            is.read(buffer);
            is.close();

            FileOutputStream fos = openFileOutput("sample.txt", 0);
            fos.write(buffer);
            fos.close();

        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

## Using the External Storage

Every Android-compatible device supports a shared "**external storage**" that you can use to save files. This can be a **removable storage media (such as an SD card)** or an **internal (non-removable) storage**. Files saved to the **external storage are world-readable** and **can be modified by the user** when they enable USB mass storage to transfer files on a computer.

> **Caution**: External files can disappear if the user mounts the external storage on a computer or removes the media, and there's no security enforced upon files you save to the external storage. All applications can read and write files placed on the external storage and the user can remove them.

- **Checking media availability**
- **Accessing files on external storage**
- **Saving files that should be shared**
- **Saving cache files**

## Databases

The Android API contains support for creating and using SQLite databases. Each database is private to the application that creates it.

To create the database, call SQLiteDatabase.create() and also subclass SQLiteOpenHelper.

As part of its support for the SQLite database system, Android exposes database management functions that let you store complex collections of data wrapped into useful objects. For example, Android defines a data type for contact information; it consists of many fields including a first and last name (strings), an address and phone numbers (also strings), a photo (bitmap image), and much other information describing a person.

Android ships with the sqlite3 database tool, which enables you to browse table contents, run SQL commands, and perform other useful functions on SQLite databases. See Examine databases (sqlite3) to learn how to run this program.

All databases, SQLite and others, are stored on the device in /data/data/package_name/databases.

We do recommend including an autoincrement value key field that can be used as a unique ID to quickly find a record. This is not required for private data, but if you implement a content provider, you must include such a unique ID field.

# Data Storage : Databases

```
@Override
public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        mOpenHelper = new DatabaseHelper(this);
        SQLiteDatabase db = mOpenHelper.getReadableDatabase();
        Cursor c = db.query(MY_DATABASE_TABLE,
                        new String[] {"lastname","firstname","country","age"}, null, null, null, null, null);
        try {
                int lastNameColumn = c.getColumnIndex("lastname");
                int firstNameColumn = c.getColumnIndex("firstname");
                int countryColumn = c.getColumnIndex("country");
                int ageColumn = c.getColumnIndex("age");
                int i = 0;
                if (c.moveToFirst()) {
                        do {
                                i++;
                                String firstName = c.getString(firstNameColumn);
                                String lastName = c.getString(lastNameColumn);
                                String country = c.getString(countryColumn);
                                int age = c.getInt(ageColumn);
                                results.add("" + i + ":" + firstName + lastName  + "(" + country + ":" +age + ")");
                        } while (c.moveToNext());
                }
        } finally {   if (c != null) c.close();       }   // /data/data/package_name/databases/sample.db
```

## Network

You can also use the network to store and retrieve data (when it's available). To do network operations, use the classes in the following packages:

- java.net.*
- android.net.*

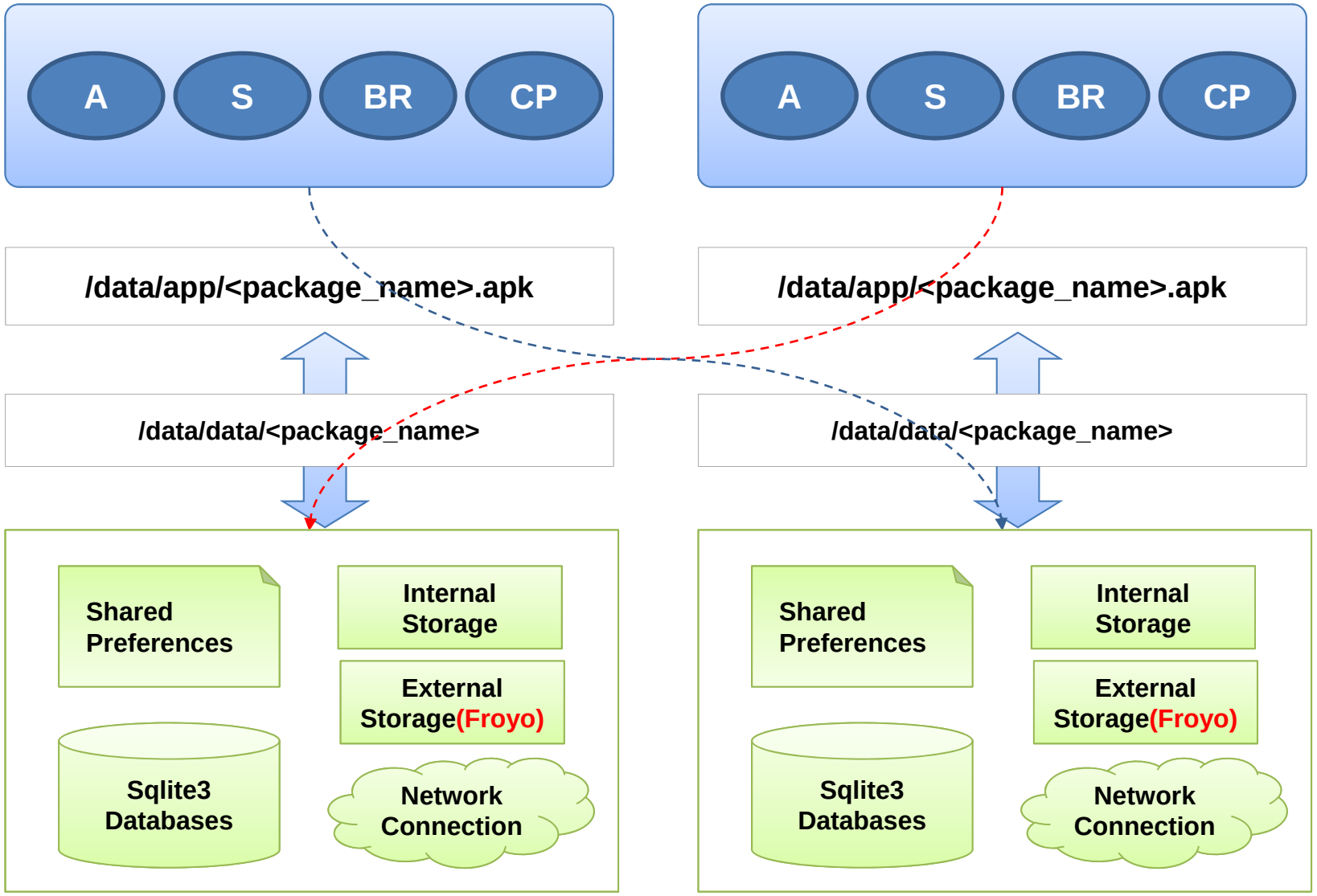
# Data Storage : Network

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;

public class SocTcpServer implements Runnable{
    public static final int SERVERPORT = 8002;

    public void run() {
        try {
            ServerSocket serverSocket = new ServerSocket(SERVERPORT);
            while (true) {
               Socket client = serverSocket.accept();
               try {
                   BufferedReader in =
                        new BufferedReader(new  InputStreamReader(client.getInputStream()));
                   String str = in.readLine();
                  } catch(Exception e) {
                  } finally {
                      client.close();
                  }
            }
        } catch (Exception e) {
        }
} }
```

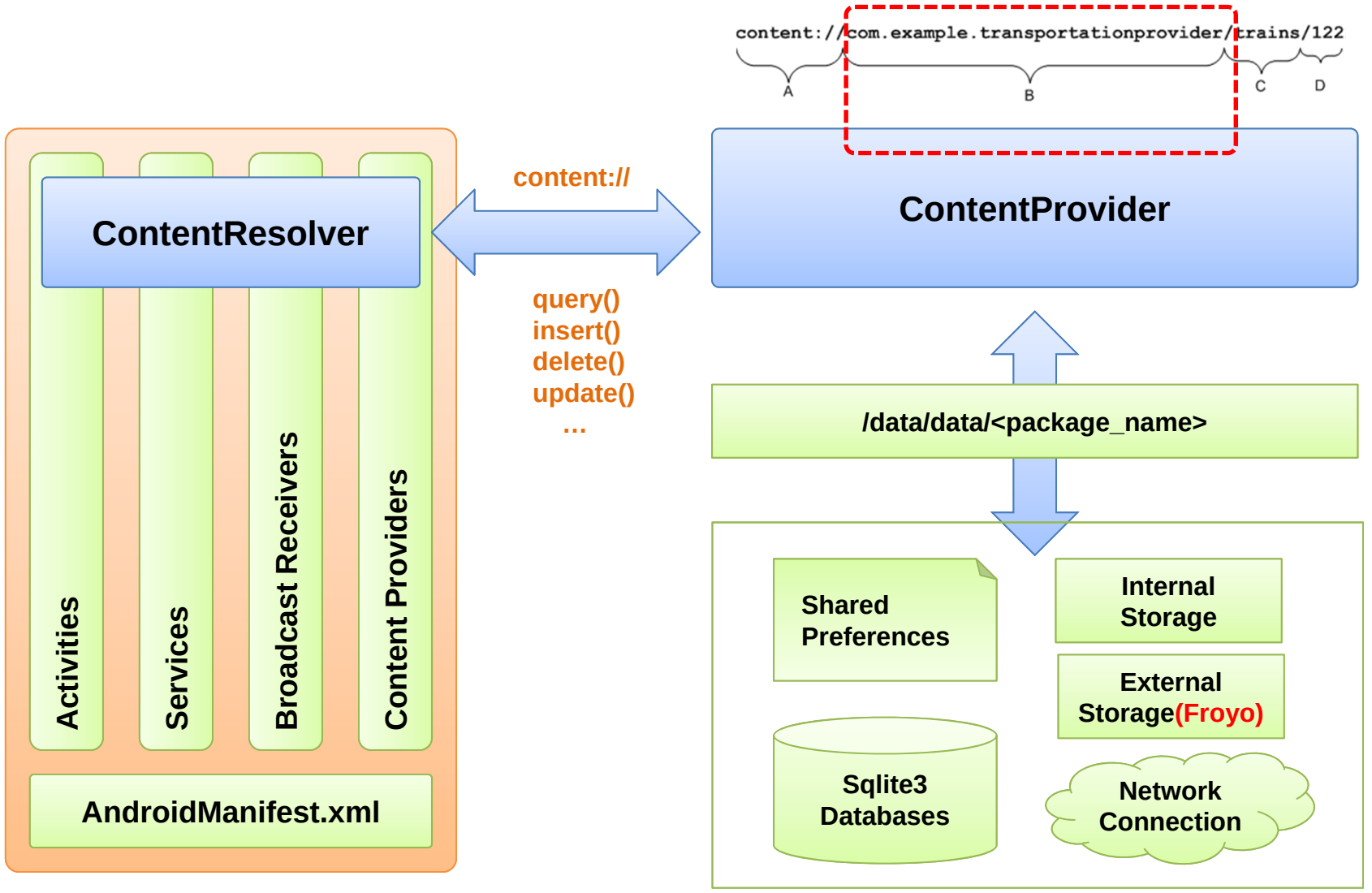
1. Data Storage

2. ContentProviders

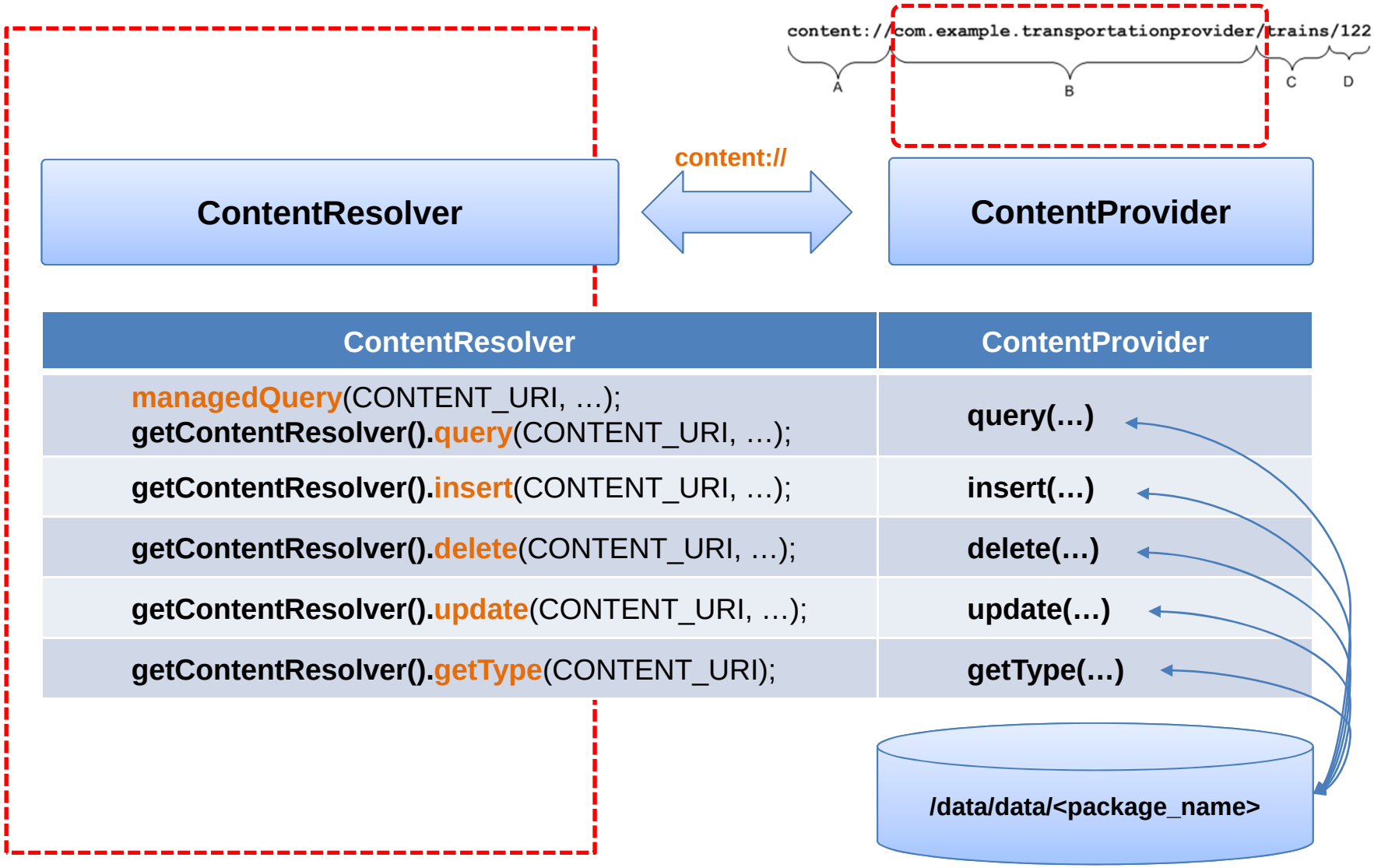
# Review : Component, Process, Apk, UID/GID, Storage



# Content Providers : Overview



# Content Providers : Overview




| ContentResolver | ContentProvider |
|---|---|
| managedQuery(CONTENT_URI, …);<br>getContentResolver().query(CONTENT_URI, …); | query(…) |
| getContentResolver().insert(CONTENT_URI, …); | insert(…) |
| getContentResolver().delete(CONTENT_URI, …); | delete(…) |
| getContentResolver().update(CONTENT_URI, …); | update(…) |
| getContentResolver().getType(CONTENT_URI); | getType(…) |

## Creating a Content Provider : Extending the ContentProvider class

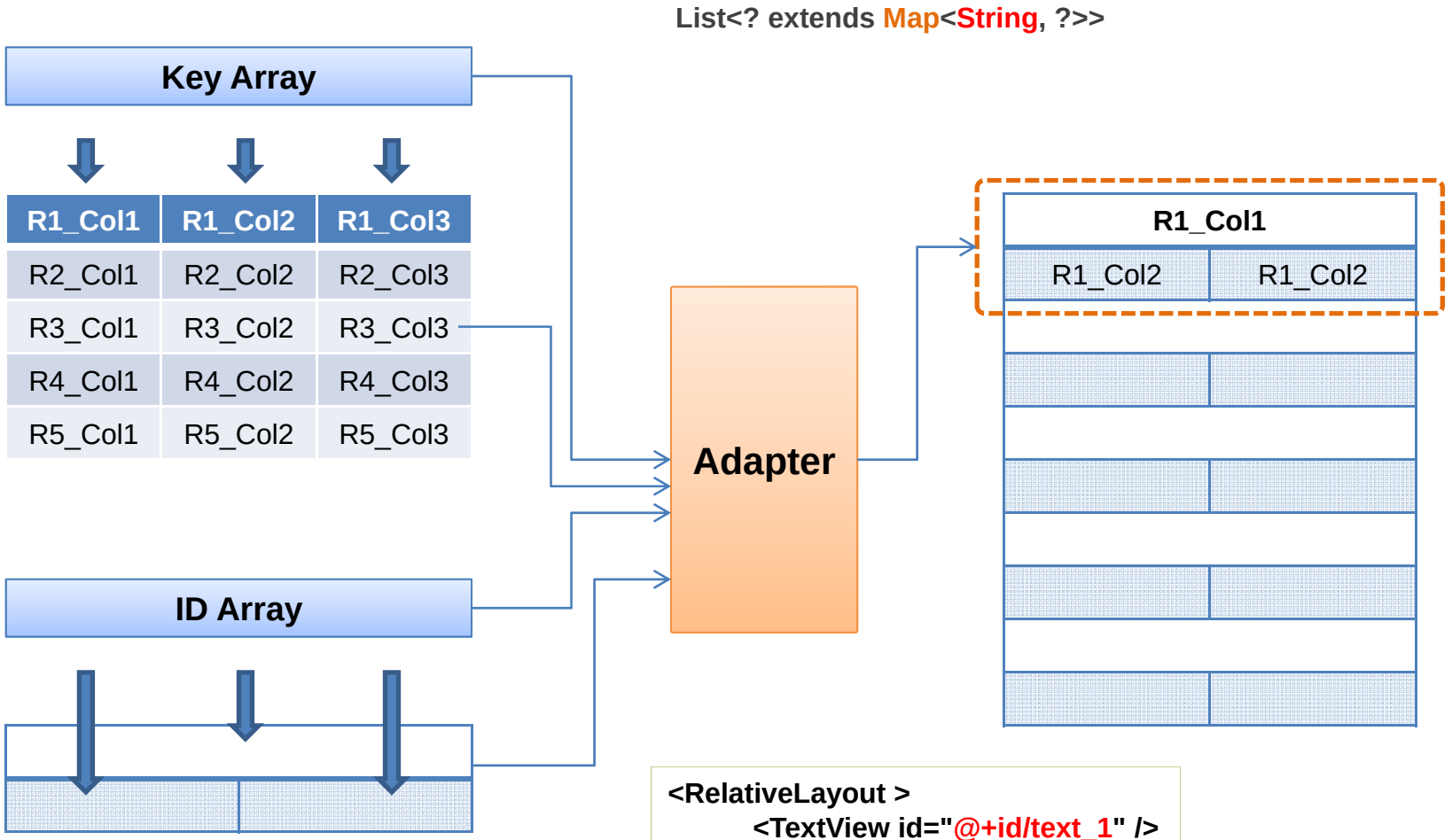
You define a ContentProvider subclass to expose your data to others using the conventions expected by ContentResolver and Cursor objects. Principally, this means implementing six abstract methods declared in the ContentProvider class:

```
public class MyProvider extends ContentProvider {

    Cursor query() { }

    Uri insert() { }

    int update() { }

    int delete() { }

    String getType() { }

    boolean onCreate() { }

}
```

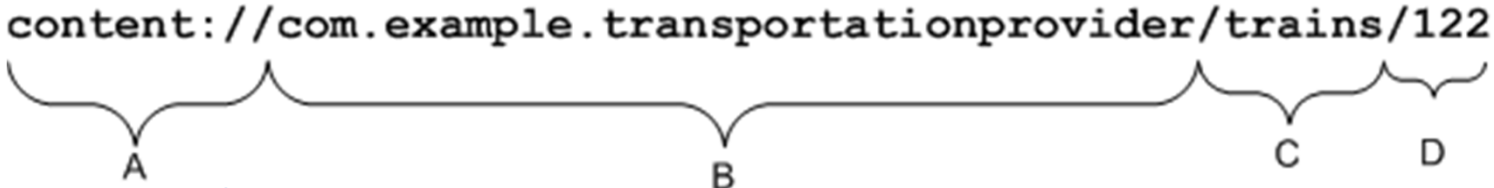
# Content Providers : Content Resolver Usage

```
// ArrayList<RowItem> budgetCategoryItems = new ArrayList<RowItem>();

final Uri CONTENT_URI =Uri.parse("content://org.kandroid.sample.Provider/namecard");

Cursor  namecardCursor= getContentResolver()
                        .query(CONTENT_URI,PROJECTION, null, null, null);

ListView budget_list = (ListView) view.findViewById(R.id.budget_list);

/*
ListAdapter mAdapter = new SimpleAdapter(
        this,budgetCategoryItems,R.layout.budget_row,
        new String[] {RowItem.ROW_TEXT_1, RowItem.ROW_TEXT_2, RowItem.ROW_TEXT_3},
        new int[]{ R.id.text_1, R.id.text_2, R.id.text_3}); */

ListAdapter mAdapter = new SimpleCursorAdapter(this, R.layout.row, namecardCursor,
                                new String[] {"title","value"},new int[] {R.id.title, R.id.value});

budget_list.setAdapter(mAdapter);
```

# AdapterView Review



## Content URI Summary




A. Standard prefix indicating that the data is controlled by a content provider. It's never modified.

B. The authority part of the URI; it identifies the content provider. For third-party applications, this should be a fully-qualified class name (reduced to lowercase) to ensure uniqueness.

```
authorities="com.example.transportationprovider"
```

C. The path that the content provider uses to determine what kind of data is being requested. This can be zero or more segments long. If the content provider exposes only one type of data (only trains, for example), it can be absent. If the provider exposes several types, including subtypes, it can be several segments long - for example, "land/bus", "land/train", "sea/ship", and "sea/submarine" to give four possibilities.The ID of the specific record being requested, if any.

D. This is the _ID value of the requested record. If the request is not limited to a single record, this segment and the trailing slash are omitted:

```
content://com.example.transportationprovider/trains
```

# Advanced Topic

1. Data Storage
    - PreferenceActivity and LayoutInflation
    - Storage Performance Issue
2. ContentProvider
    - Binary Data Transmission
    - Permission Granting and Per-URI Permission Usage
    - ContentObserver

# 1. Security and Permissions

## Overview

Android is a multi-process system, in which each application (and parts of the system) runs in its own process.

Most security between applications and the system is enforced at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications.

Additional finer-grained security features are provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform, and per-URI permissions for granting ad-hoc access to specific pieces of data.

## Security Architecture

A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or e-mails), reading or writing another application's files, performing network access, keeping the device awake, etc.

An application's process is a secure sandbox. It can't disrupt other applications, except by explicitly declaring the permissions it needs for additional capabilities not provided by the basic sandbox.

These permissions it requests can be handled by the operating in various ways, typically by automatically allowing or disallowing based on certificates or by prompting the user.

The permissions required by an application are declared statically in that application, so they can be known up-front at install time and will not change after that.

## Application Signing

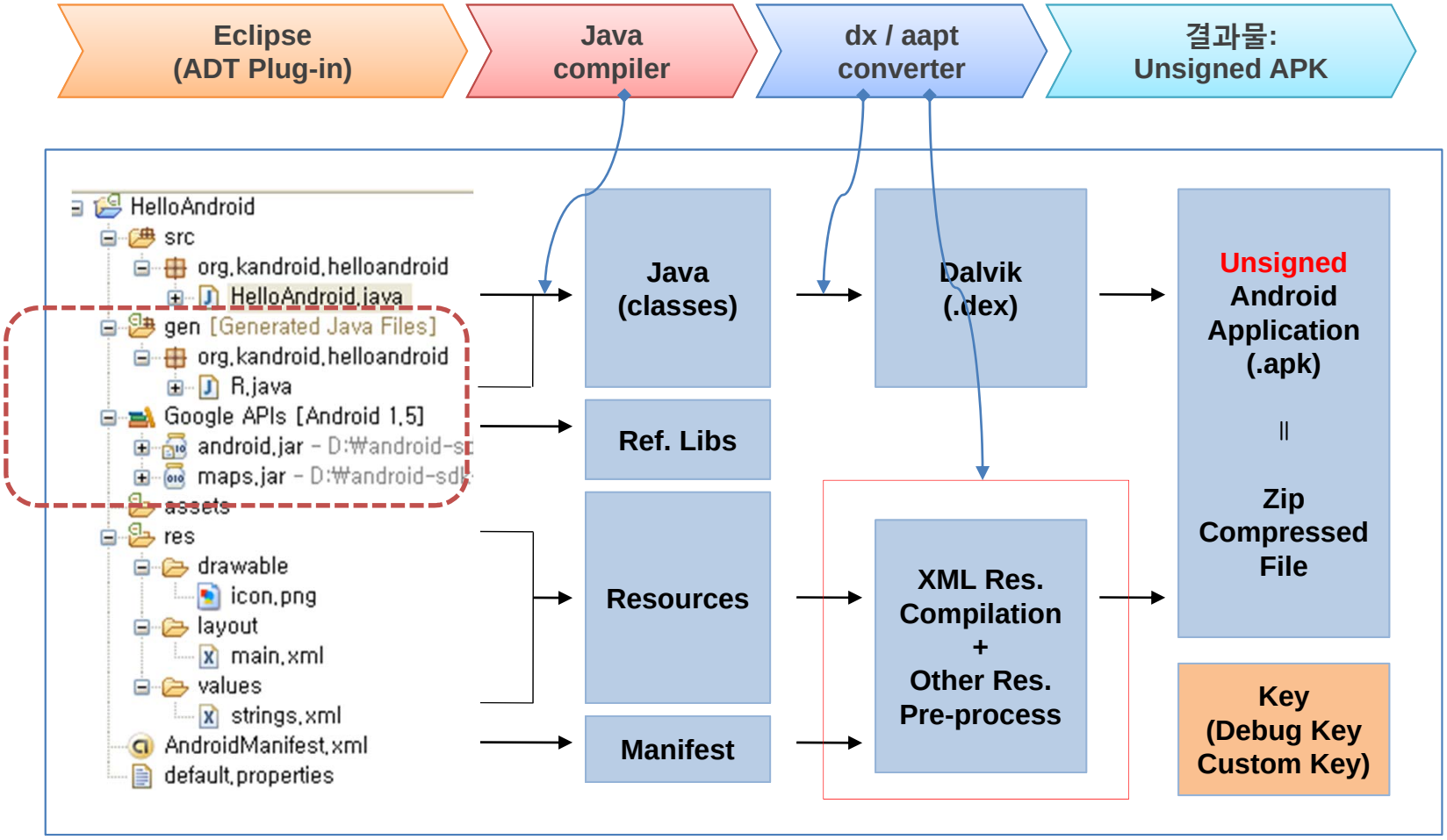
All Android applications (.apk files) must be signed with a certificate whose private key is held by their developer. This certificate identifies the author of the application.

The certificate does not need to be signed by a certificate authority:
it is perfectly allowable, and typical, for Android applications to use self-signed certificates.

The certificate is used only to establish trust relationships between applications,
not for wholesale control over whether an application can be installed.

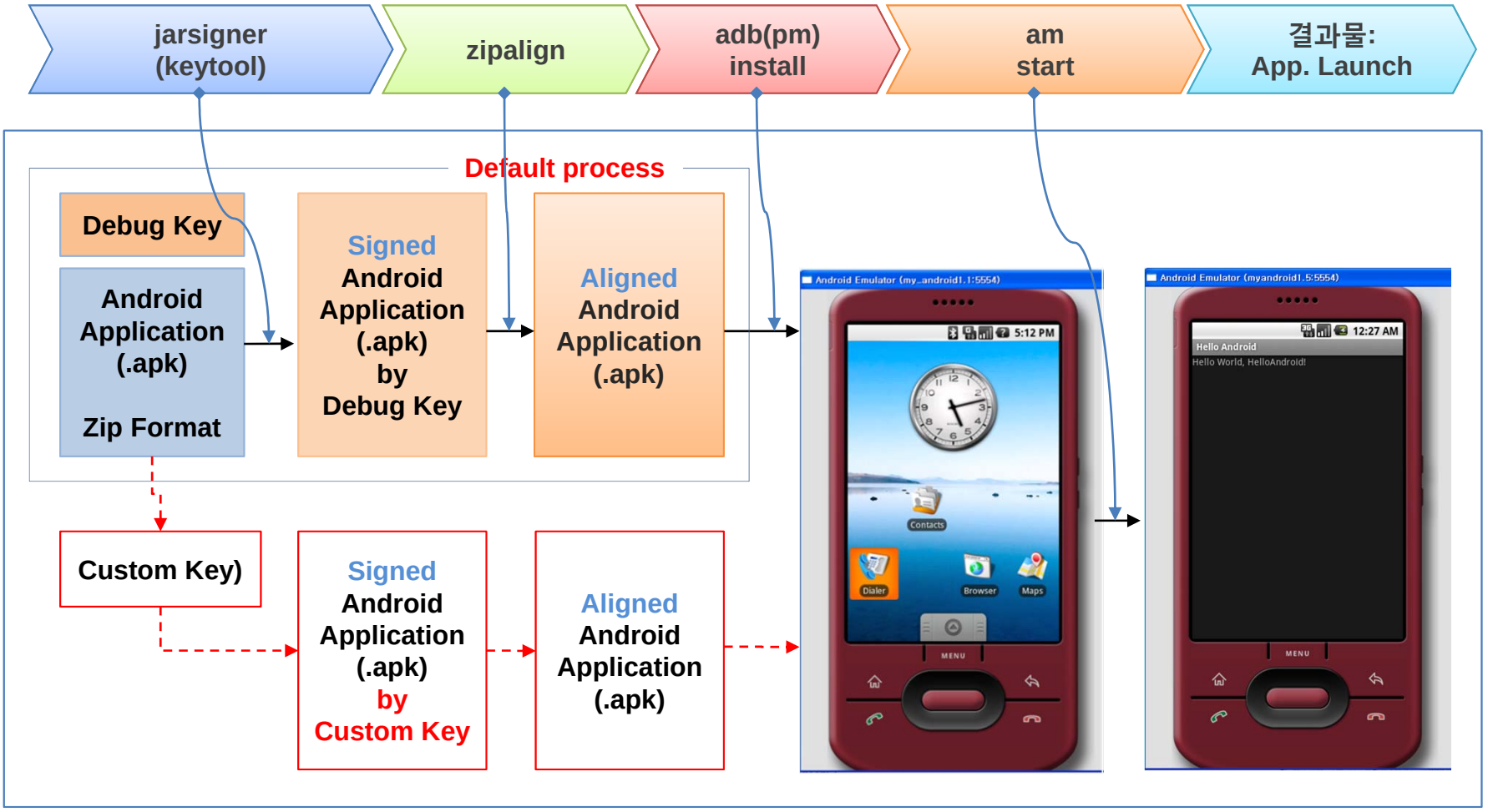
The most significant ways that signatures impact security is by determining
who can access signature-based permissions and who can share user IDs.

## 안드로이드 애플리케이션 빌드 프로세스(1)



# Security and Permissions : Application Signing

## 안드로이드 애플리케이션 빌드 프로세스(2)



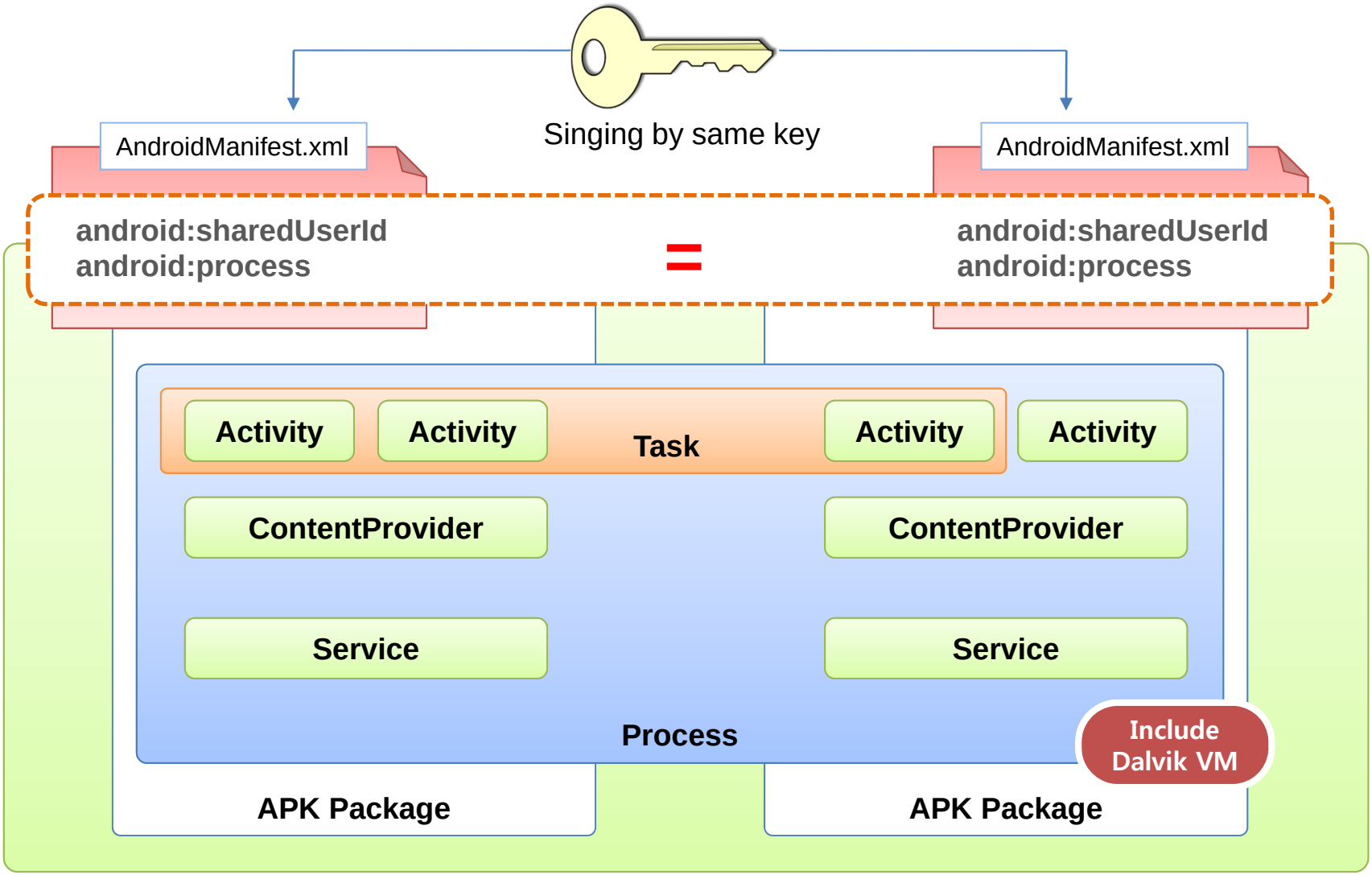
## User IDs and File Access

Each Android package (.apk) file installed on the device is given its own unique Linux user ID, creating a sandbox for it and preventing it from touching other applications (or other applications from touching it). This user ID is assigned to it when the application is installed on the device, and remains constant for the duration of its life on that device.

Because security enforcement happens at the process level, the code of any two packages can not normally run in the same process, since they need to run as different Linux users. You can use the sharedUserId attribute in the AndroidManifest.xml's manifest tag of each package to have them assigned the same user ID. By doing this, for purposes of security the two packages are then treated as being the same application, with the same user ID and file permissions. Note that in order to retain security, only two applications signed with the same signature (and requesting the same sharedUserId) will be given the same user ID.

Any data stored by an application will be assigned that application's user ID, and not normally accessible to other packages. When creating a new file with getSharedPreferences(String, int), openFileOutput(String, int), or openOrCreateDatabase(String, int, SQLiteDatabase.CursorFactory), you can use the MODE_WORLD_READABLE and/or MODE_WORLD_WRITEABLE flags to allow any other package to read/write the file. When setting these flags, the file is still owned by your application, but its global read and/or write permissions have been set appropriately so any other application can see it.

# Security and Permissions : shardUserId, Signature, Process



## Using Permissions

A basic Android application has no permissions associated with it, meaning it can not do anything that would adversely impact the user experience or any data on the device. To make use of protected features of the device, you must include in your AndroidManifest.xml one or more <uses-permission> tags declaring the permissions that your application needs.

For example, an application that needs to monitor incoming SMS messages would specify:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
</manifest>
```

At application install time, permissions requested by the application are granted to it by the package installer, based on checks against the signatures of the applications declaring those permissions and/or interaction with the user. No checks with the user are done while an application is running: it either was granted a particular permission when installed, and can use that feature as desired, or the permission was not granted and any attempt to use the feature will fail without prompting the user.

## Using Permissions (cont.)

The permissions provided by the Android system can be found at Manifest.permission. Any application may also define and enforce its own permissions, so this is not a comprehensive list of all possible permissions.

A particular permission may be enforced at a number of places during your program's operation:

- At the time of a call into the system, to prevent an application from executing certain functions.
- When starting an activity, to prevent applications from launching activities of other applications.
- Both sending and receiving broadcasts, to control who can receive your broadcast or who can send a broadcast to you.
- When accessing and operating on a content provider.
- Binding or starting a service.

# Security and Permissions : Permission

## Define Permission

```
<permission android:name="string“ … />
```

## Use Permission

```
<uses-permission
        android:name="string" />
```

## Enforce Permission

```
<activity android:permission="string“ />
<activity-alias android:permission="string“ />
<receiver android:permission="string“ />
<service android:permission="string” />
<provider android:grantUriPermissions=["true" | "false"]
                android:permission="string“
                android:readPermission="string“
                android:writePermission="string" >
        <grant-uri-permission android:path="string"
                        android:pathPattern="string"
                        android:pathPrefix="string" />
        <path-permission android:path="string"
                                android:pathPrefix="string"
                                android:pathPattern="string"
                                android:permission="string"
                                android:readPermission="string"
                                android:writePermission="string“ />
</provider>
```

## Declaring and Enforcing Permissions

To enforce your own permissions, you must first declare them in your AndroidManifest.xml using one or more <permission> tags.

For example, an application that wants to control who can start one of its activities could declare a permission for this operation as follows:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.app.myapp" >

    <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permlab_deadlyActivity"
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />

</manifest>
```

## Declaring and Enforcing Permissions - protectionLevel

| Constant | V | Description |
|---|---|---|
| normal | 0 | A lower-risk permission that gives an application access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, with out asking for the user's explicit approval (though the user always has the option to review these permissions before installing). |
| dangerous | 1 | A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system may not automatically grant it to the requesting application. For example, any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities. |
| signature | 2 | A permission that the system is to grant only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval. |
| Signature OrSystem | 3 | A permission that the system is to grant only to packages in the Android system image *or* that are signed with the same certificates. Please avoid using this option, as the signature protection level should be sufficient for most needs and works regardless of exactly where applications are installed. This permission is used for certain special situations where multiple vendors have applications built in to a system image which need to share specific features explicitly because they are being built together. |

## Declaring and Enforcing Permissions in AndroidManifest.xml

High-level permissions restricting access to entire components of the system or application can be applied through your AndroidManifest.xml. All that this requires is including an android:permission attribute on the desired component, naming the permission that will be used to control access to it.

Activity permissions (applied to the <activity> tag) restrict who can start the associated activity. The permission is checked during Context.startActivity() and Activity.startActivityForResult(); if the caller does not have the required permission then SecurityException is thrown from the call.

Service permissions (applied to the <service> tag) restrict who can start or bind to the associated service. The permission is checked during Context.startService(), Context.stopService() and Context.bindService(); if the caller does not have the required permission then SecurityException is thrown from the call.

BroadcastReceiver permissions (applied to the <receiver> tag) restrict who can send broadcasts to the associated receiver. The permission is checked after Context.sendBroadcast() returns, as the system tries to deliver the submitted broadcast to the given receiver. As a result, a permission failure will not result in an exception being thrown back to the caller; it will just not deliver the intent. In the same way, a permission can be supplied to Context.registerReceiver() to control who can broadcast to a programmatically registered receiver. Going the other way, a permission can be supplied when calling Context.sendBroadcast() to restrict which BroadcastReceiver objects are allowed to receive the broadcast (see below).

## Declaring and Enforcing Permissions in AndroidManifest.xml (cont.)

**ContentProvider permissions (applied to the <provider> tag) restrict who can access the data in a ContentProvider. (Content providers have an important additional security facility available to them called URI permissions which is described later.) Unlike the other components, there are two separate permission attributes you can set: android:readPermission restricts who can read from the provider, and android:writePermission restricts who can write to it. Note that if a provider is protected with both a read and write permission, holding only the write permission does not mean you can read from a provider. The permissions are checked when you first retrieve a provider (if you don't have either permission, a SecurityException will be thrown), and as you perform operations on the provider. Using ContentResolver.query() requires holding the read permission; using ContentResolver.insert(), ContentResolver.update(), ContentResolver.delete() requires the write permission. In all of these cases, not holding the required permission results in a SecurityException being thrown from the call.**

## Enforcing Permissions when Sending Broadcasts

In addition to the permission enforcing who can send Intents to a registered BroadcastReceiver ( as described above), you can also specify a required permission when sending a broadcast. By calling Context.sendBroadcast() with a permission string, you require that a receiver's application must hold that permission in order to receive your broadcast.

Note that both a receiver and a broadcaster can require a permission. When this happens, both permission checks must pass for the Intent to be delivered to the associated target.

## Other Permissions Enforcement

Arbitrarily fine-grained permissions can be enforced at any call into a service. This is accomplished with the Context.checkCallingPermission() method. Call with a desired permission string and it will return an integer indicating whether that permission has been granted to the current calling process. Note that this can only be used when you are executing a call coming in from another process, usually through an IDL interface published from a service or in some other way given to another process.

There are a number of other useful ways to check permissions. If you have the pid of another process, you can use the Context method Context.checkPermission(String, int, int) to check a permission against that pid. If you have the package name of another application, you can use the direct PackageManager method PackageManager.checkPermission(String, String) to find out whether that particular package has been granted a specific permission.

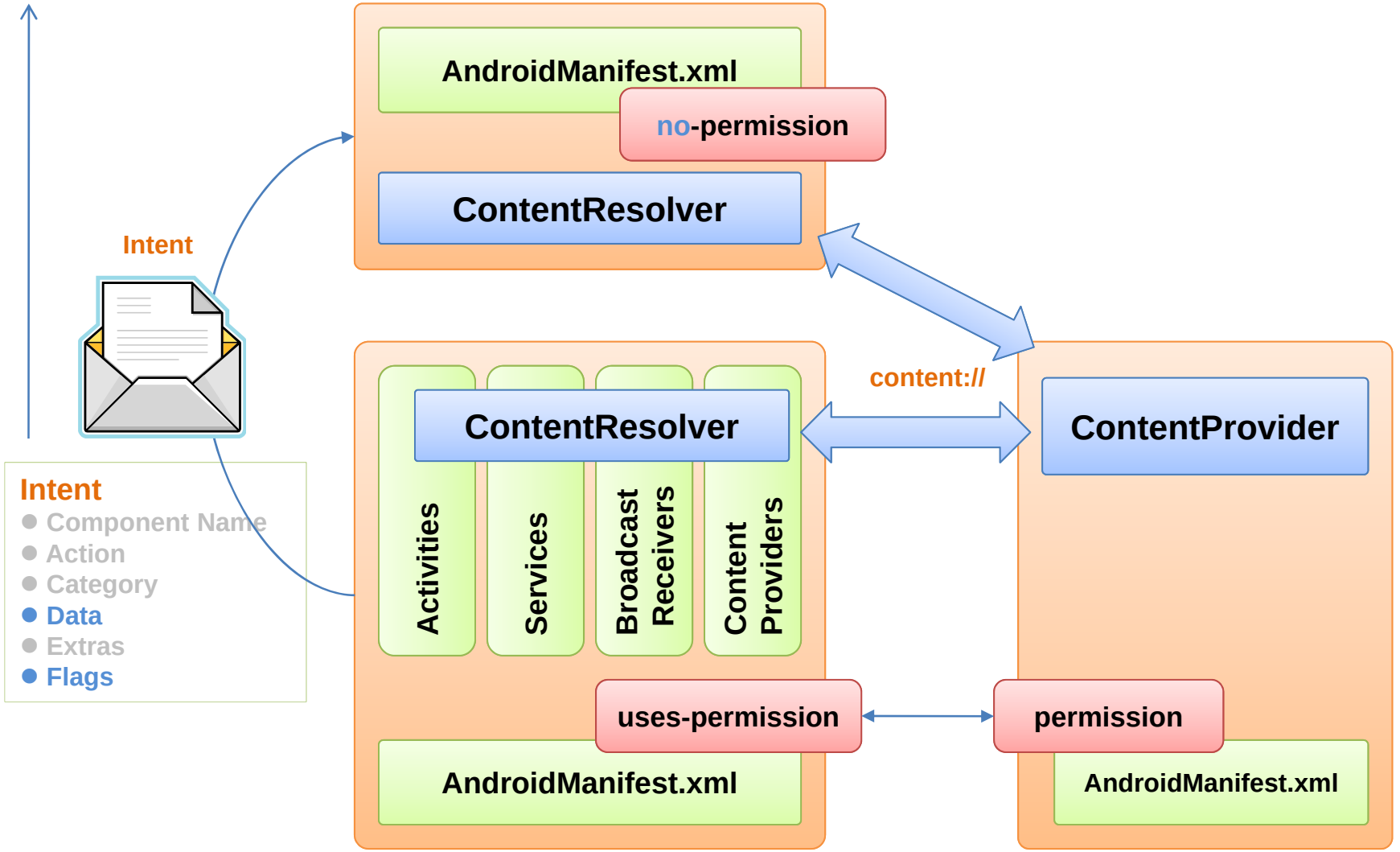
## URI Permissions

The standard permission system described so far is often not sufficient when used with content providers. A content provider may want to protect itself with read and write permissions, while its direct clients also need to hand specific URIs to other applications for them to operate on. A typical example is attachments in a mail application. Access to the mail should be protected by permissions, since this is sensitive user data. However, if a URI to an image attachment is given to an image viewer, that image viewer will not have permission to open the attachment since it has no reason to hold a permission to access all e-mail.

The solution to this problem is per-URI permissions: when starting an activity or returning a result to an activity, the caller can set Intent.FLAG_GRANT_READ_URI_PERMISSION and/or Intent.FLAG_GRANT_WRITE_URI_PERMISSION. This grants the receiving activity permission access the specific data URI in the Intent, regardless of whether it has any permission to access data in the content provider corresponding to the Intent.

This mechanism allows a common capability-style model where user interaction (opening an attachment, selecting a contact from a list, etc) drives ad-hoc granting of fine-grained permission. This can be a key facility for reducing the permissions needed by applications to only those directly related to their behavior.
The granting of fine-grained URI permissions does, however, require some cooperation with the content provider holding those URIs. It is strongly recommended that content providers implement this facility, and declare that they support it through the android:grantUriPermissions attribute or <grant-uri-permissions> tag.

```
i.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
i.addFlags(Intent.FLAG_GRANT_WRITE_URI_PERMISSION);
i.setData(CONTENT_URI);
```

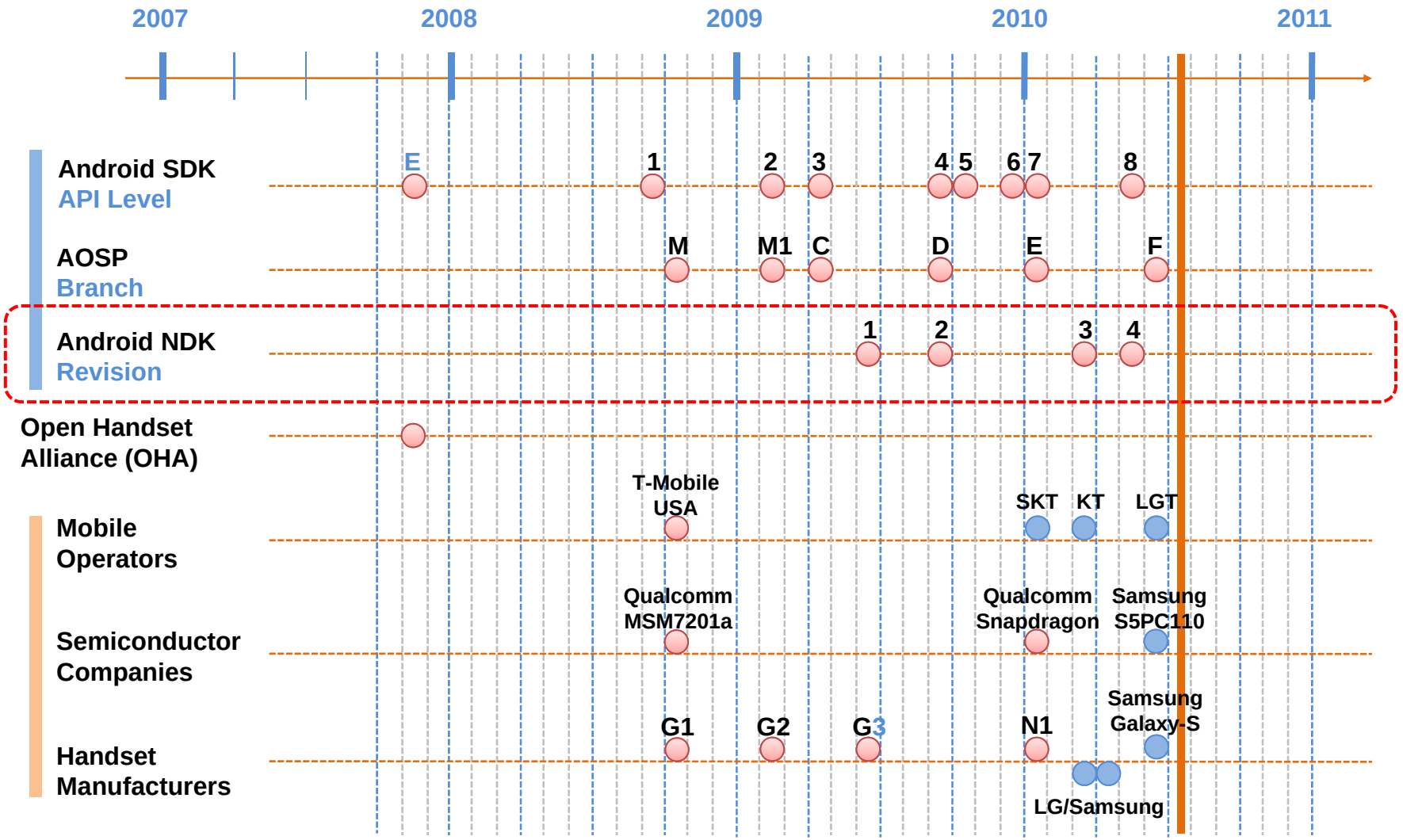


# Advanced Topic

1. Signing
    - platform / media / shared / testkey
2. Permission
    - /system/etc/permissions
3. Security
    - MITM (Man in the middle) Attack

# NDK(Native Development Kit)

# NDK(Native Development Kit)



## 11. NDK는 C 또는 C++ stand-alone 애플리케이션 개발 도구가 아니다

최초의 안드로이드 SDK 릴리즈 이후, 안드로이드 NDK(Native Development Kit)가 두 번에 걸쳐서 릴리즈되었다. 안드로이드 SDK가 최초로 릴리즈된 시점에는 안드로이드가 Java 프로그래밍 언어만을 지원한다고 공식 발표되었다. C 또는 C++ 애플리케이션 개발 환경은 지원하지 않는다는 것이 공식입장이었다. 하지만 안드로이드 SDK 1.5(Cupcake)가 릴리즈된 시점에서 NDK 1.5가, 안드로이드 SDK 1.6(Donut)이 릴리즈된 시점에는 NDK 1.6이 릴리즈되었다. 흔히들, 안드로이드 NDK는 C 또는 C++ 네이티브 개발을 지원한다고 생각한다. 하지만 이러한 생각은 부분적으로만 옳다. 정확하게 말하면, 안드로이드 NDK는 C 또는 C++ stand-alone 애플리케이션 개발을 위한 것이 아니라, 안드로이드 SDK상에서 개발 가능한 애플리이션에서 JNI 인터페이스를 통해 호출 가능한 C 또는 C++ 네이티비 라이브러리를 개발하기 위한 환경을 제공함이 목적이다. 또한 안드로이드 NDK에서 사용가능한 C 또는 C++ API는 안드로이드 플랫폼상에 존재하는 모든 네이티브 라이브러리가 아닌, Bionic 계열의 라이브러리에 한정된다. 최초의 NDK 1.5는 Bionic 계열의 네이티브 라이브러리에 한정되었으나, NDK 1.6에서는 OpenGL ES 1.1 3D 그래픽 라이브러리가 포함되었다.

안드로이드 플랫폼상의 네이티브 라이브러리는 크게 Bionic, Functional, Server, HAL로 구분된다. 이것들 중 NDK가 Bionic과 OpenGL ES 1.1에 한정된 API만 지원하는 이유는 무엇일까? 그것은 미래에 릴리즈될 안드로이드 플랫폼에서의 호환성 유지의 범위를 그것으로 국한한다는 의미일 것이다.

# NDK(Native Development Kit)

http://www.kandroid.org/board/board.php?board=androidsource&command=body&no=48

| Subject | Description |
|---|---|
| Goals | • The Android VM allows your application's source code to call methods implemented in native code through the JNI.<br>• The Android NDK is a complement to the Android SDK that helps you to:<br>- Generate JNI-compatible shared libraries<br>- Copy the generated shared libraries to a proper location<br>- Provide tools that help debug your native code (in later revisions)<br>• Moreover, the Android NDK provides:<br>- A set of cross-toolchains<br>- A set of system headers<br>- A build system |
| Non-Goals | • The NDK is *not* a good way to write generic native code that runs on Android devices.<br>• A good understanding of JNI is highly recommended<br>• The NDK only provides system headers for a very limited set of native APIs and libraries |
| In practice | • Configuring the NDK<br>• Placing C and C++ sources<br>• Writing an Android.mk build script<br>• Writing an Application.mk build file<br>• Invoke the NDK build system |

# NDK(Native Development Kit) : Setup Dev. Environment

**http://www.kandroid.org/board/board.php?board=androidsource&command=body&no=48**

**1.** NDK Download : http://developer.android.com/sdk/ndk/index.html

**2.** Cygwin Install

- http://www.cygwin.com/setup.exe 을 다운로드 후, 실행한다.
  - Cygwin Net Release Setup Program : 다음(N)
  - Choose A Download Source : Install From Intenet (check) : 다음(N)
  - Select Root Install Directory : Root Directory C:cygwin : 다음(N)
  - Select Local Package Directory : 다음(N)
  - Select Your Internet Connection : Direct Connection(check) : 다음(N)
  - Choose A Download Site : http://ftp.daum.net (예) : 다음(N)
  - Select Package : make & gcc 선택할 것 : 다음(N)
    - Devel :
      - gcc-core (Skip->3.4.4-999) 바꿀 것 (마우스로 클릭하면 됨)
      - gcc-g++ (Skip->3.4.4-999) 바꿀 것 (마우스 클릭하면 됨)
      - make (Skip -> 3.81-1) 바꿀 것 (마우스 클릭하면 됨)
  - Create Icons : Add icon to Start Menu(check) : 마침(N)

**3.** Windows 시작메뉴 > 모든 프로그램 > Cygwin > Cygwin Bash Shell

**4.** 1에서 Download한 NDK 을 C:\cygwin\home\<your_computer> 로 옮긴 후,
해당 디렉토리에서 압축을 푼다.

# NDK(Native Development Kit) : Usage

Windows 시작메뉴 > 모든 프로그램 > Cygwin > Cygwin Bash Shell 상에서 다음과 같이 실행한다.

**$ cd android-ndk-r4**

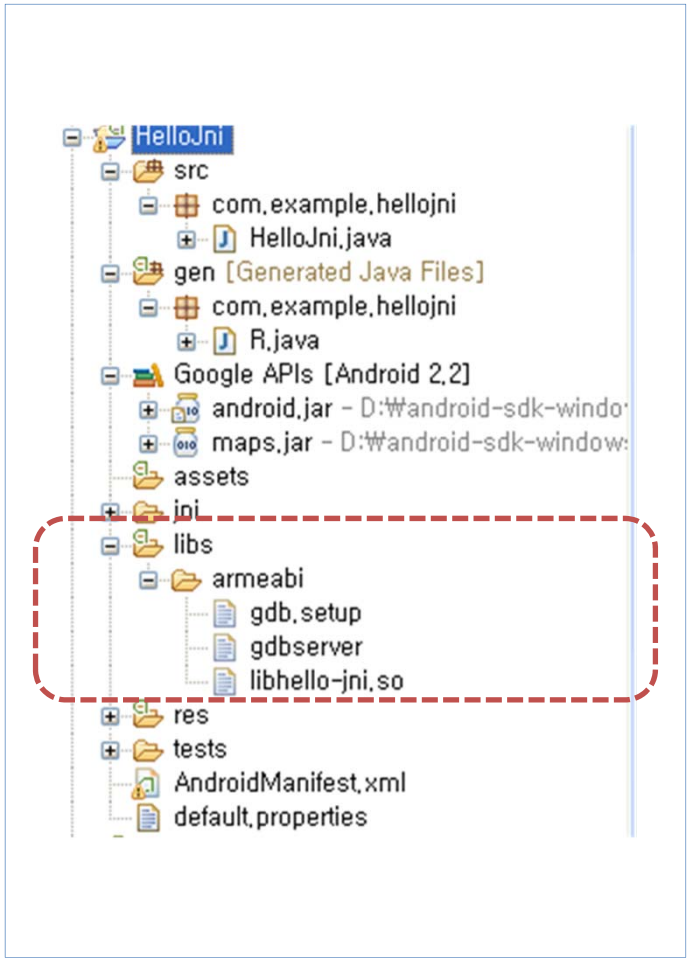
**$ ls**
GNUmakefile README.TXT build docs kandroid **ndk-build** ndk-gdb **samples** source

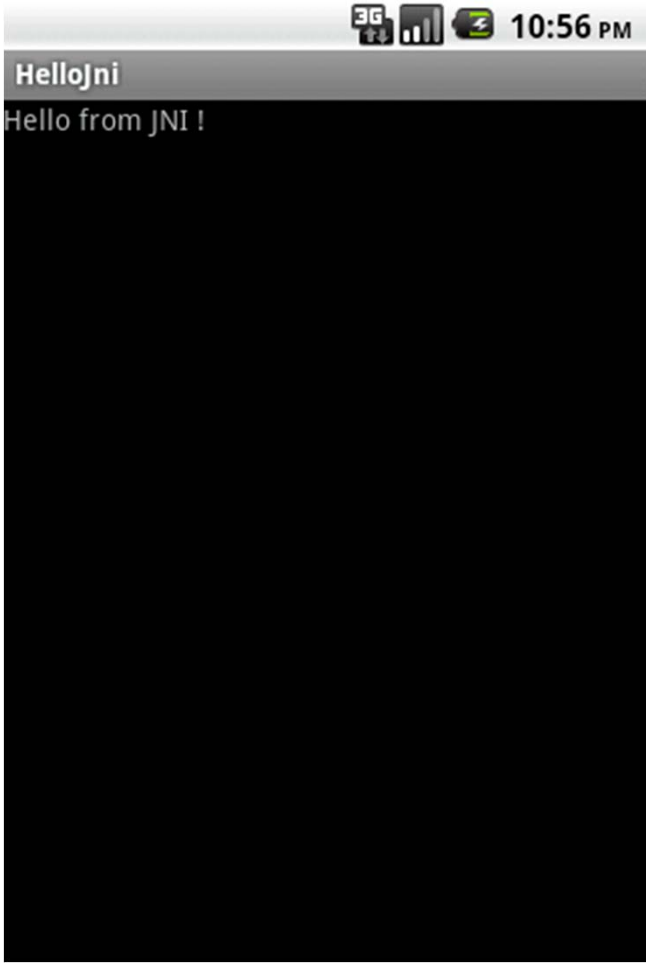
**$ cd samples/hello-jni**

**$ ../../ndk-build**
Gdbserver : [arm-eabi-4.4.0] /home/kandroid/android-ndk-r4/samples/hello-jni/libs/armeabi/gdbserver
Gdbsetup : /home/kandroid/android-ndk-r4/samples/hello-jni/libs/armeabi/gdb.setup
Gdbsetup : + source directory /home/kandroid/android-ndk-r4/samples/hello-jni/jni
Compile thumb : hello-jni <= /home/kandroid/android-ndk-r4/samples/hello-jni/jni/hello-jni.c
SharedLibrary : **libhello-jni.so**
Install : libhello-jni.so => /home/kandroid/android-ndk-r4/samples/hello-jni/libs/armeabi

**이후, Eclipse 상에서 해당 Project를 Load 한다.**

# NDK(Native Development Kit) : Usage



# NDK(Native Development Kit) : Usage

```
public class HelloJni extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        TextView  tv = new TextView(this);
        tv.setText( stringFromJNI() );

        setContentView(tv);
    }

    public native String  stringFromJNI();

    /* /data/data/<package_name>/lib/libhello-jni.so */

    static {
        System.loadLibrary("hello-jni");
    }
}
```

# NDK(Native Development Kit) : Usage

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c

include $(BUILD_SHARED_LIBRARY)
```

```
#include <string.h>
#include <jni.h>

jstring
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env, jobject thiz )
{
    return (*env)->NewStringUTF(env, "Hello from JNI !");
}
```

# NDK(Native Development Kit) : Android JNI Tips

- **What's JNI?**
- **JavaVM and JNIEnv**
- **jclass, jmethodID, and jfieldID**
- **Local vs. Global References**
- **UTF-8 and UTF-16 Strings**
- **Primitive Arrays**
- **Region Calls**
- **Exceptions**
- **Extended Checking**
- **Native Libraries**
- **64-bit Considerations**
- **Unsupported Features**

**JNI** is **the Java Native Interface**. It defines **a way for code written in the Java programming language to interact with native code, e.g. functions written in C/C++**. **It's VM-neutral, has support for loading code from dynamic shared libraries, and while cumbersome at times is reasonably efficient**. You really should read through the **JNI spec for J2SE 1.6** to get a sense for how JNI works and what features are available. Some aspects of the interface aren't immediately obvious on first reading, so you may find the next few sections handy. The more detailed *JNI Programmer's Guide and Specification* can be found here.

https://android.git.kernel.org/?p=platform/dalvik.git;a=blob_plain;f=docs/jni-tips.html;h=e85434b525688f083104cdb46e33226c38d6e839;hb=HEAD

# 질의 및 응답

Q & A