

Mobile Programming

Android 6: Intents, App-Widgets, & Verschiedenes



Nicola Keller



Inhalt

- Intents
 - Filter & Auflösung
- App-Widgets
 - "Mini-App-UI auf Home-Screen"
- App-Design
- Usability & Prototyping
- Android JetPack (Android X)
 - AppCompatActivity (ehem. Support Library)
- Publizieren von Apps
 - Packaging, Signing & Release



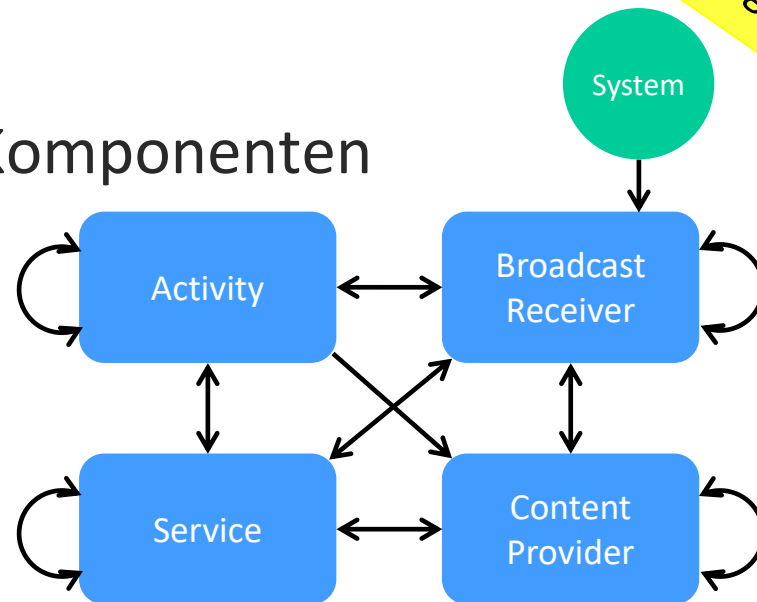
<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Intent Filters

Intents: Aufruf anderer Komponenten

■ Kommunikation zwischen Komponenten

- Activity <-> Activity
- Activity <-> Service
- Service <-> Service
- ...



■ Intents sind der "Leim" zwischen den Android Systemkomponenten

- Analogie: Postpaket mit Nachricht (=Daten)
 - *Genaue Adresse oder Empfängerbeschreibung*

Explizite
Intents

Implizite
Intents

Kontrollübergabe mittels Intents

- Android benutzt Intents, um Komponenten zu benachrichtigen oder um Kontrolle zu übergeben
- **Zwei Arten von Intents:**
 - **Explizite Intents** adressieren Komponente direkt
 - **Implizite Intents** beschreiben geeigneten Empfänger
- Implizite Intents kann man sich als Verb (und Objekt) vorstellen: Kurzbeschreibung, was getan werden soll
 - z.B.: *view image, take photo, call contact, play movie, ...*
- Das System übergibt dann der am besten passenden Komponente

Aber wie?

...Intent Auswahl? - Intent-Filter!

■ Implizite Intents

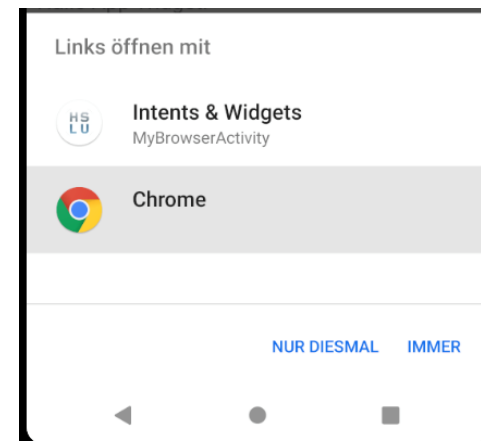
- Empfänger steht nicht im Vornherein fest
- Nachricht mit "Anforderungsprofil" für Empfänger

■ System eruiert mögliche(n) Empfänger

- Drei mögliche Fälle:

1. Genau ein Empfänger gefunden: Direkte Zustellung
2. Mehrere Empfänger: Auswahl durch Benutzer (Dialog), siehe rechts:
3. Kein Empfänger: Wurf von `ActivityNotFoundException` o.ä.

= Intent Auflösung (EN: Resolution)



Intent-Filter

Oder u.U. auch im Code; siehe Broadcast Receivers vom letzten Mal!

- (Potentielle) Intent-Empfänger deklarieren im Manifest entsprechende Intent-Filter
- System vergleicht implizite Intents mit deklarierten Filtern und liefert passende Komponenten zurück
 - Lässt Benutzer auswählen, falls mehrere Filter passen und kein Favorit registriert

```
<activity  
  android:name=".Receiver"  
  android:label="@string/app_name">  
  <intent-filter>  
    <action android:name="ch.hslu.mobpro.actions.SHOW_TEXT" />  
    <category android:name="android.intent.category.LAUNCHER" />  
    <category android:name="android.intent.category.DEFAULT" />  
  </intent-filter>  
</activity>
```

Eine eigene Action!

Soll eine Activity sein, die für App-Start verwendet werden kann (Entry-Point)

Soll Default-Activity sein für diese Action

Demo: Eigene Intent-Action

Activity gestartet durch folgende Intent-ACTION:
'ch.hslu.mobpro.actions.SHOW_TEXT'
Jetzt = Wed Apr 01 21:51:15 GMT+02:00 2020

■ Bsp.: eigene Show-Text-Action

`ch.hslu.mobpro.actions.SHOW_TEXT`

■ Start durch Intent mit entspr. Custom-Action:

- Möglich aus bel. Activity/App (→ lose Koppelung)

```
fun startCustomIntentOnClick(view: View?) {
    val customIntent = Intent()
    customIntent.action = MY_ACTION_SHOW_TEXT
    customIntent.addCategory(Intent.CATEGORY_DEFAULT)
    customIntent.addCategory(Intent.CATEGORY_LAUNCHER)
    val myText = ""Activity gestartet durch folgende Intent-ACTION:
        '$MY_ACTION_SHOW_TEXT'
        Jetzt = ${Date()}"".trimIndent()
    customIntent.putExtra(MY_EXTRA_KEY, myText)
    startActivity(customIntent)
}
```

Kein Expliziter Empfängertyp
= impliziter Intent

Bsp. für einen Kotlin
Multiline-String... ☺

■ In onCreate: testen ob Activity damit gestartet wurde

- Falls ja: text aus Extras darstellen

Siehe rot umrahmten
Text ganz rechts oben

Impliziter Intent: Daten

- Implizite Intents enthalten (optional) folgende Daten
 - **Action:** Der Typ der Aktion, die ausgeführt werden soll, z.B. ACTION_VIEW, ACTION_EDIT, MY_CUSTOM_ACTION, ...
 - **Category:** Kategorie der Komponenten, welche diesen Intent ausführen soll, z.B. DEFAULT, LAUNCHER, BROWSABLE (=eine Komponente welche von einem Browser aufgerufen werden kann)
 - **Data:** Beschreibung der Daten, mit welchen gearbeitet werden soll (URI und Mime Type)
 - **Extras:** Schlüssel/Wert-Paare für Zusatzinformationen

Auflösung impliziter Intents

- Das Android-System löst implizite Intents auf, indem die am besten zu einem Intent passende Komponente gewählt wird
- "Best match" wird festgelegt durch Vergleich von
 - Action: Die Action vom Intent muss im Filter sein
 - Category: Jede Kategorie vom Intent muss im Filter sein
 - Data (URI und Mime type): Alles im Intent unter Data aufgelistete muss zum Filter passen

Siehe <https://developer.android.com/guide/components/intents-filters.html#Resolution>

Implizite Intents: Google-Doku

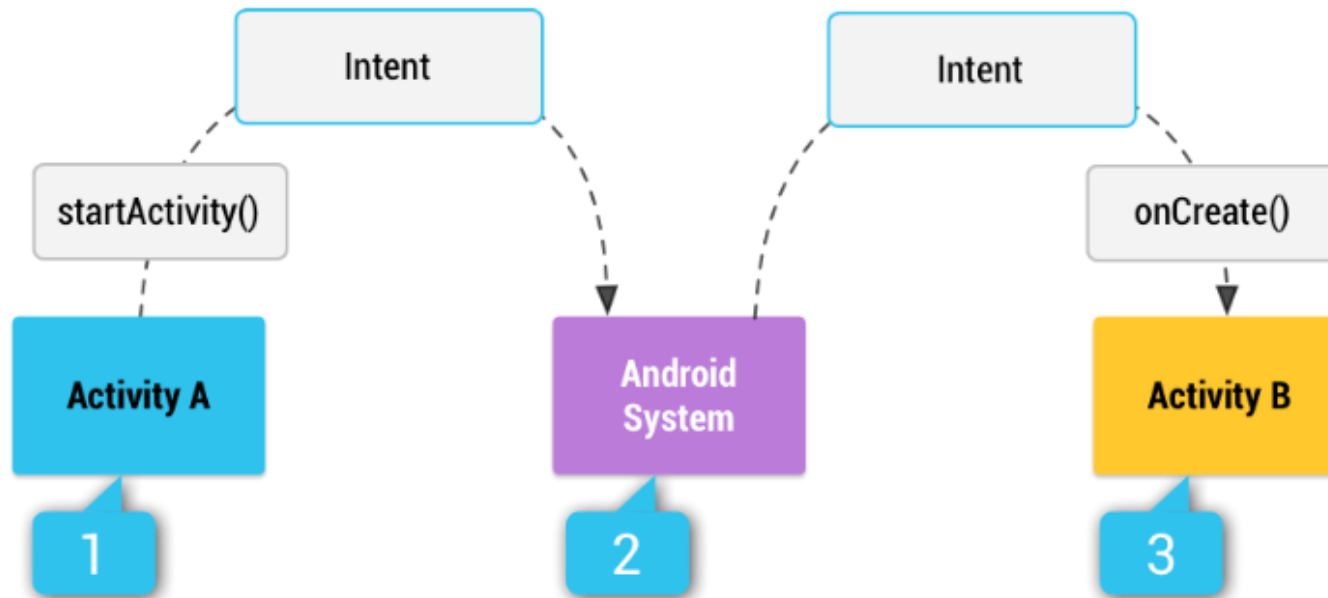


Figure 1. How an implicit intent is delivered through the system to start another activity: **[1]** *Activity A* creates an `Intent` with an action description and passes it to `startActivity()`. **[2]** The Android System searches all apps for an intent filter that matches the intent. When a match is found, **[3]** the system starts the matching activity (*Activity B*) by invoking its `onCreate()` method and passing it the `Intent`.

Beispiel: Filter für Browser-Intent (nicht mehr so einfach seit Android 12)

- Erinnerung: Übung 1 - Browser-Intent:

```
val browserCall = Intent()  
browserCall.action = Intent.ACTION_VIEW  
browserCall.data = Uri.parse("http://www.hslu.ch")  
startActivity(browserCall)
```

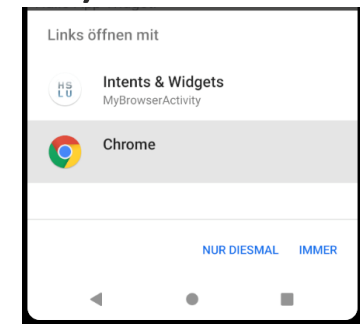
- Jetzt: eigene Activity, welche diesen Browser-Intent "versteht", d.h. inkl. entsprechendem Intent-Filter:

```
<activity  
    android:name=".MyBrowserActivity"  
    android:label="MyBrowserActivity">  
    <intent-filter>  
        <action android:name="android.intent.action.VIEW" />  
        <category android:name="android.intent.category.BROWSABLE" />  
        <category android:name="android.intent.category.DEFAULT" />  
        <data android:scheme="http" />  
        <data android:scheme="https" />  
    </intent-filter>  
</activity>
```

Demo: Eigene Browser Activity

- Standard-Browser und eigene `MyBrowserActivity` "verstehen" jetzt (Dank passendem Intent-Filter) Browser-Intents

☞ System fragt Benutzer! 😊



```
class MyBrowserActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        // Very simple for view creation without layout inflation  
        // from xml. We just set a web view as content view.  
        val webView = WebView(context: this)  
        setContentView(webView)  
  
        // Very simple URL loading  
        webView.loadUrl(url: intent.dataString?: "https://sbb.ch")  
    }  
}
```



Intent-Auflösung abfragen?

- Klasse `PackageManager` liefert diverse Infos zu den aktuell installierten Packages in diesem Android System
 - z.B. kann die Intent-Auflösung (Resolution) abgefragt werden mittels
 - `query...()` -> alles passende
 - `resolve...()` -> best-passende

-> <https://developer.android.com/reference/android/content/pm/PackageManager.html>

Intent-Auflösung braucht Berechtigung...

- Damit alle Packages sichtbar sind, ist ab API 30 eine entsprechende Deklaration im Manifest nötig
 - <https://developer.android.com/training/package-visibility>

Android Developers > Docs > Guides

Package visibility filtering on Android

```
<queries>
  <intent>
    <action android:name="android.intent.action.VIEW" />
    <data android:scheme="*" />
  </intent>
</queries>

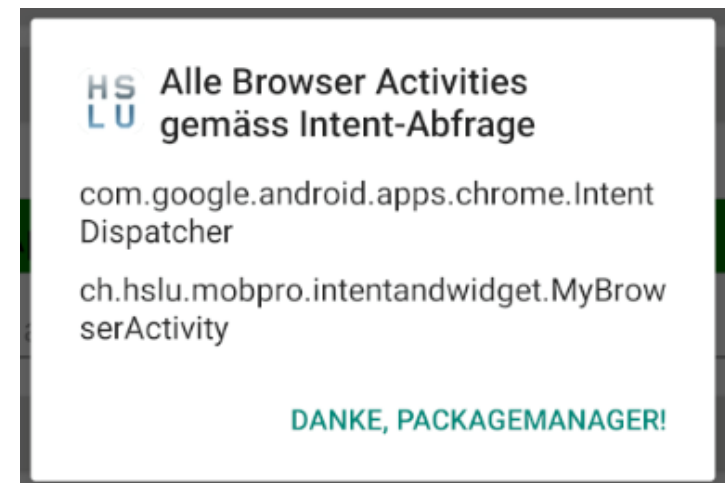
</manifest>
```


Demo: Browser-Activities abfragen & anzeigen

1. Activities für Intent abfragen

```
val resolveList = packageManager  
    .queryIntentActivities(hsluBrowserIntent,  
        PackageManager.MATCH_DEFAULT_ONLY)
```

2. Für alle ResolveInfos das Feld `activityInfo.name` in Dialog anzeigen





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

App-Widgets

App-Widgets

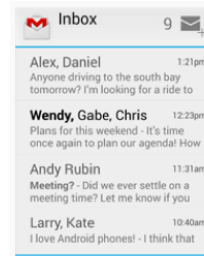


- Ermöglichen individuelle Home-Screen Anpassungen
 - Zeigen wichtigste Daten und Funktionalitäten einer App direkt auf dem Home-Screen an
 - z.B. Wetter, Kalender, News, aktuelle Musik, ...
 - Können auf Home-Screen verschoben werden und die Grösse kann angepasst werden (falls unterstützt)
- ☞ Eine Art "Mini-App" auf dem Home- Screen

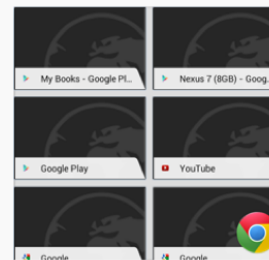
Siehe <https://developer.android.com/guide/topics/appwidgets/>

Typen von App-Widgets

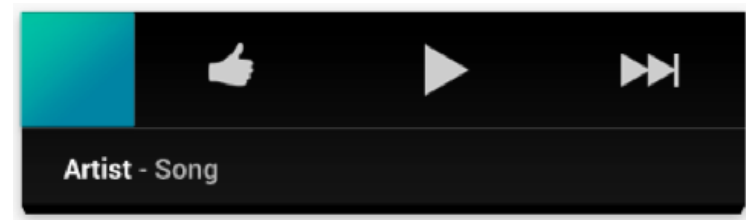
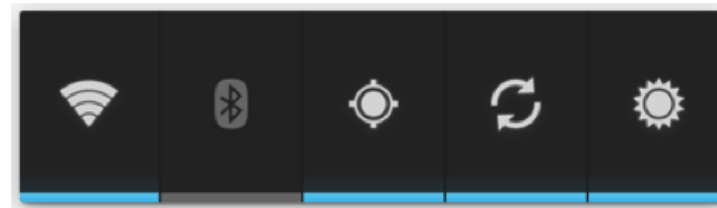
- Information Widgets
- Collection Widgets
- Control Widgets
- Hybrid Widgets
 - Kombination der obigen...



ListView widget



GridView widget



App Widget: Howto...

- Widget muss dem System bekannt gemacht werden
 - Im Android Manifest
 - App-Widgets sind spezielle `BroadcastReceiver`
 - ☞ Deklaration als Receiver-Komponente
- Dazu muss `AppWidgetProviderInfo` angegeben werden, 2 Optionen:
 - Deklarativ: `widget_provider_info.xml` i.A. bevorzugt
 - Programmatisch, Klasse `AppWidgetProviderInfo`
- UI für Widget in `widget.xml`-Datei
- Eigene, von `AppWidgetProvider` abgeleitete Klasse, behandelt Aktualisierungs-Logik usw.

App-Widget: Receiver-Deklaration

■ Im Manifest

```
<receiver android:name=".MyAppWidgetProvider">  
  <intent-filter>  
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />  
  </intent-filter>  
  
  <meta-data  
    android:name="android.appwidget.provider"  
    android:resource="@xml/my_app_widget_provider_info" />  
</receiver>
```

Meine AppWidgetProvider-Klasse, erbt
von AppWidgetProvider

AppWidgetProvider erbt
von BroadcastReceiver

Wir reagieren auf diese Action!

Ein App-Widget ist
eine Receiver-
Komponente

Hier ist die AppWidgetProviderInfo abgelegt
(in diesem Fall also deklarativ in xml-Datei)

my_app_widget_provider_info.xml

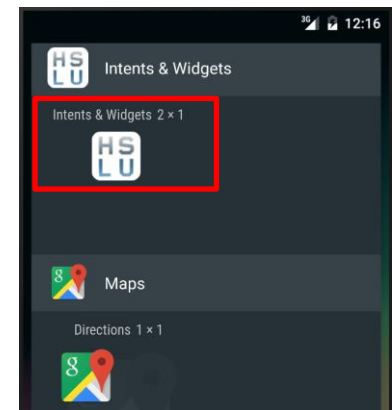
```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialKeyguardLayout="@layout/my_app_widget_provider"
    android:initialLayout="@layout/my_app_widget_provider"
    android:minHeight="40dp"
    android:minWidth="80dp"
    android:previewImage="@drawable/ic_launcher"
    android:resizeMode="horizontal|vertical"
    android:updatePeriodMillis="86400000"
    android:widgetCategory="home_screen" />
```

Hier liegt das
Widget-Layout

Gibt die (initiale)
Grösse auf dem
Home-Screen vor

Dieses Bild erscheint in der
Android-Widget-
Auswahlliste, s.u.

Default ist Home-Screen. Weitere
mögliche Kategorie wäre z.B. Lock-
Screen (keyguard)



my_app_widget_provider.xml

- Mein Bsp: Simple Layout mit einer TextView

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/appwidget_text"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:background="@color/widgetBackground"
        android:gravity="center"
        android:padding="5dp"
        android:text="@string/widget_initialText"
        android:textColor="@color/widgetText"
        android:textSize="20sp" />

</LinearLayout>
```

AppWidgetProvider-Klasse

Wird aufgerufen zur Aktualisierung

```
class MyAppWidgetProvider : AppWidgetProvider() {  
    override fun onUpdate(context: Context,  
                           appWidgetManager: AppWidgetManager,  
                           appWidgetIds: IntArray) {  
        val text = getTextFromPrefs(context)  
        // Perform this loop for each App Widget that belongs to this provider  
        for (appWidgetId in appWidgetIds) {  
            val views = RemoteViews(context.packageName, R.layout.my_app_widget_provider)  
            views.setTextViewText(R.id.appwidget_text, text: "$text \n ${updateCount++}. Akt.")  
            appWidgetManager.updateAppWidget(appWidgetId, views)  
        }  
    }  
}
```

RemoteViews = Prototyp-Objekt. Das Widget wird nicht direkt updated (ist lose gekoppelt). Events aus Widget (z.B. für Buttons) müssen mit "pending event" Deklaration hinterlegt werden

Android System triggert Widget-Updates nur alle 30 Minuten (um Batterie zu sparen). Diese Zeit kann verkürzt werden, indem der entsprechende Broadcast selber verschickt wird (z.B. aus App)

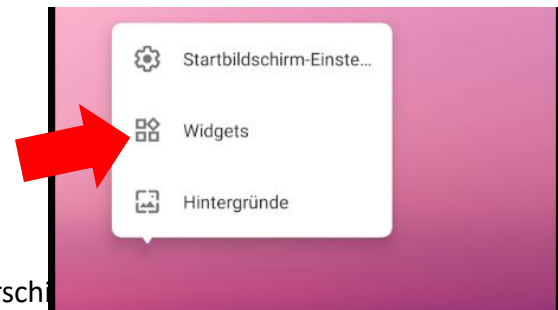
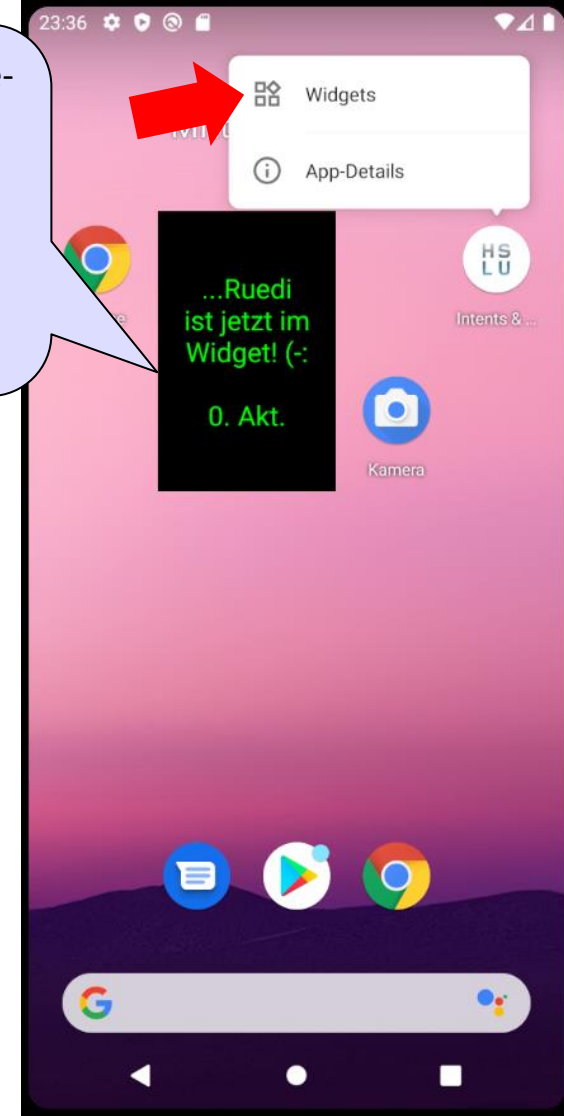
★ **Note:** Updates requested with `updatePeriodMillis` will not be delivered more than once every 30 minutes.

von <http://developer.android.com/reference/android/appwidget/AppWidgetProviderInfo.html#updatePeriodMillis>
siehe auch: <https://developer.android.com/guide/topics/appwidgets#MetaData>

Demo: MyAppWidget

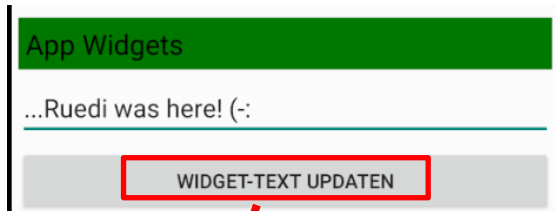
- App-Widget zeigt Text auf MainActivity an
- Info, Provider und Layout wie auf vorangehenden Folien
 - Widget-Inhalt: programmatisch gesetzt in `MyAppWidgetProvider`
- Demo
 - Mein Widget unter "WIDGETS"
 - Zur Android Widget-Liste: Long-Press auf Home-Screen
 - Widget aus Liste auf Home-Screen platzieren

Widget platzieren auf Home-Screen: Long-Click auf Home-Screen & Ziehen aus Widget-Liste ODER: Long-Click auf App-Icon & Click auf Widgets...



Widget: Aktualisierung auslösen (Code inkl. Demo)

- Widgets aktualisiert sich maximal alle 30' via System
- Bei Änderung von Daten (z.B. aus App), ist Update ggf. sofort gewünscht



```
fun requestWidgetUpdate(context: Context) {  
    val widget = ComponentName(context, MyAppWidgetProvider::class.java)  
    val appWidgetIds = AppWidgetManager.getInstance(context).getAppWidgetIds(widget)  
    val updateWidget = Intent(context, MyAppWidgetProvider::class.java)  
    updateWidget.action = AppWidgetManager.ACTION_APPWIDGET_UPDATE  
    updateWidget.putExtra(AppWidgetManager.EXTRA_APPWIDGET_IDS, appWidgetIds)  
    context.sendBroadcast(updateWidget)  
}
```

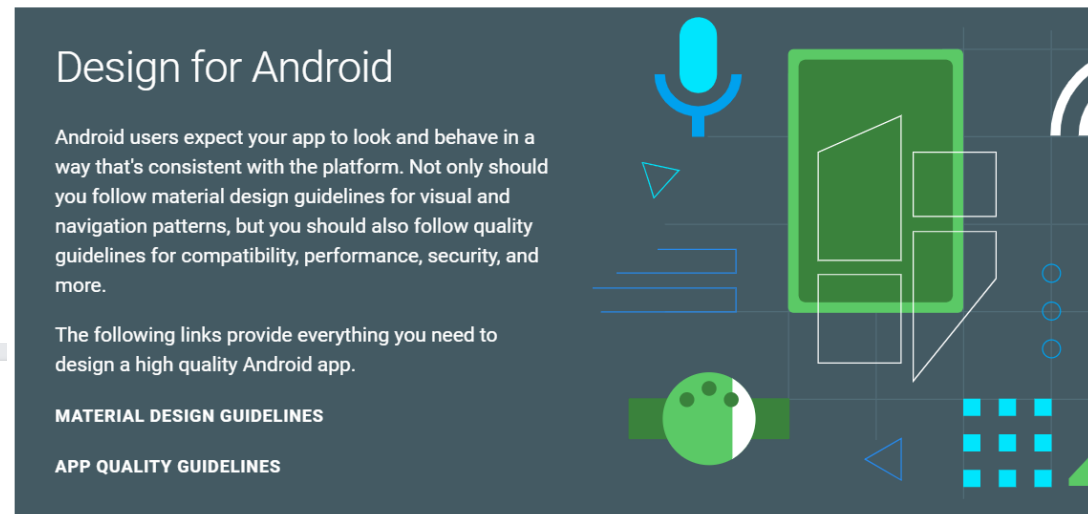


<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

App-Design

Design-Vorgaben

- <http://developer.android.com/design/>



Android Developers > Docs > Design & Quality

Core app quality

Android users expect high-quality apps. App quality directly influences the long-term success of your app—in terms of installs, user rating and reviews, engagement, and user retention.

This page helps you assess the core aspects of quality in your app, through a compact set of quality criteria and associated tests. All Android apps should meet these criteria.

Before publishing your apps, test them against these criteria to ensure that they function well on many devices, meets Android standards for navigation and design, and are prepared for promotional opportunities in the Google Play store. Your testing will go well beyond what's described here—the purpose of this document is to specify the essential quality characteristics all apps should display, so that you can cover them in your test plans.

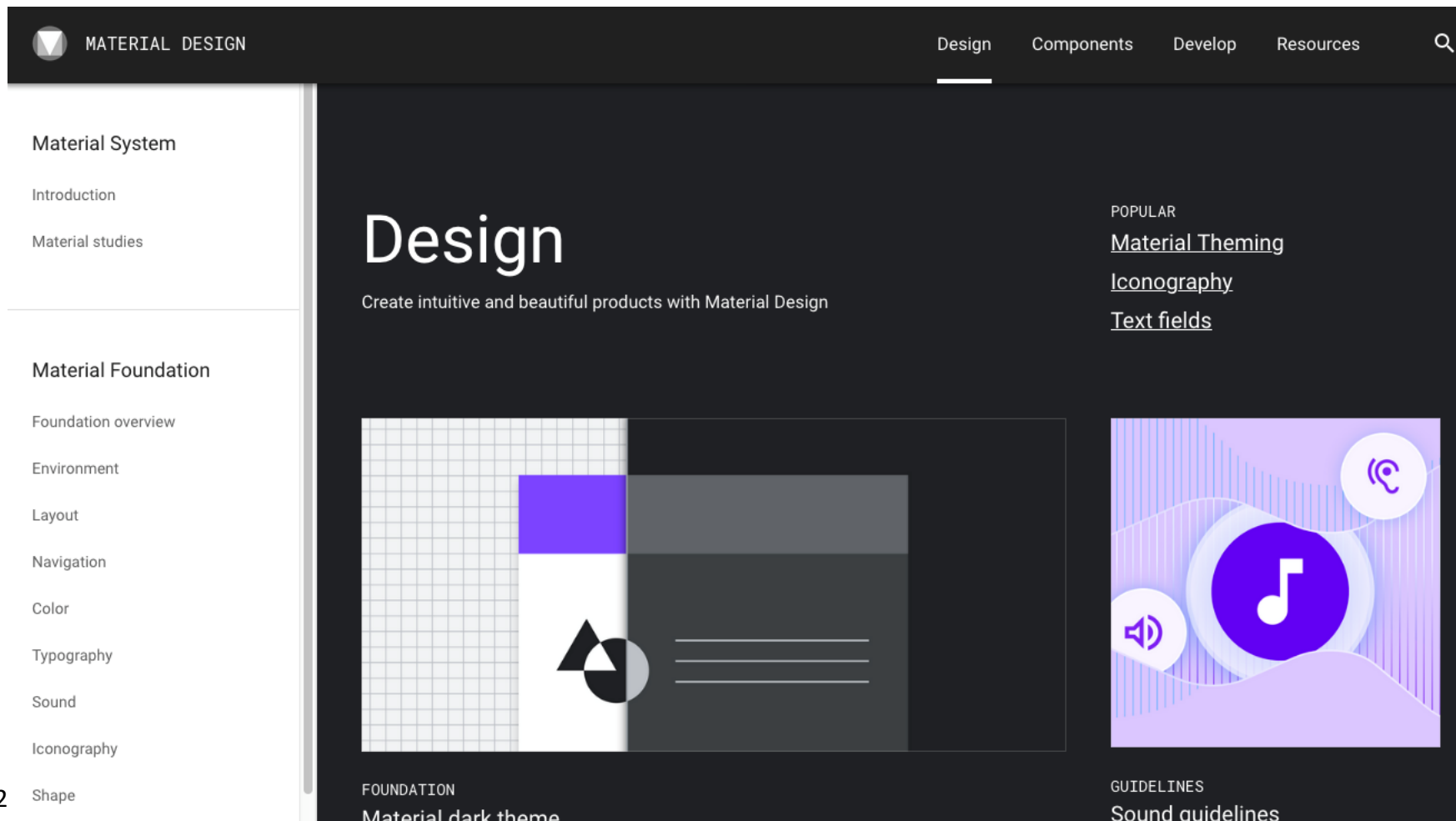
If you're creating apps for other Android devices, such as tablets or TV, there are [additional quality guidelines](#) you should consider.



Google Play
Setting up a test environment
Test procedures
Testing with StrictMode

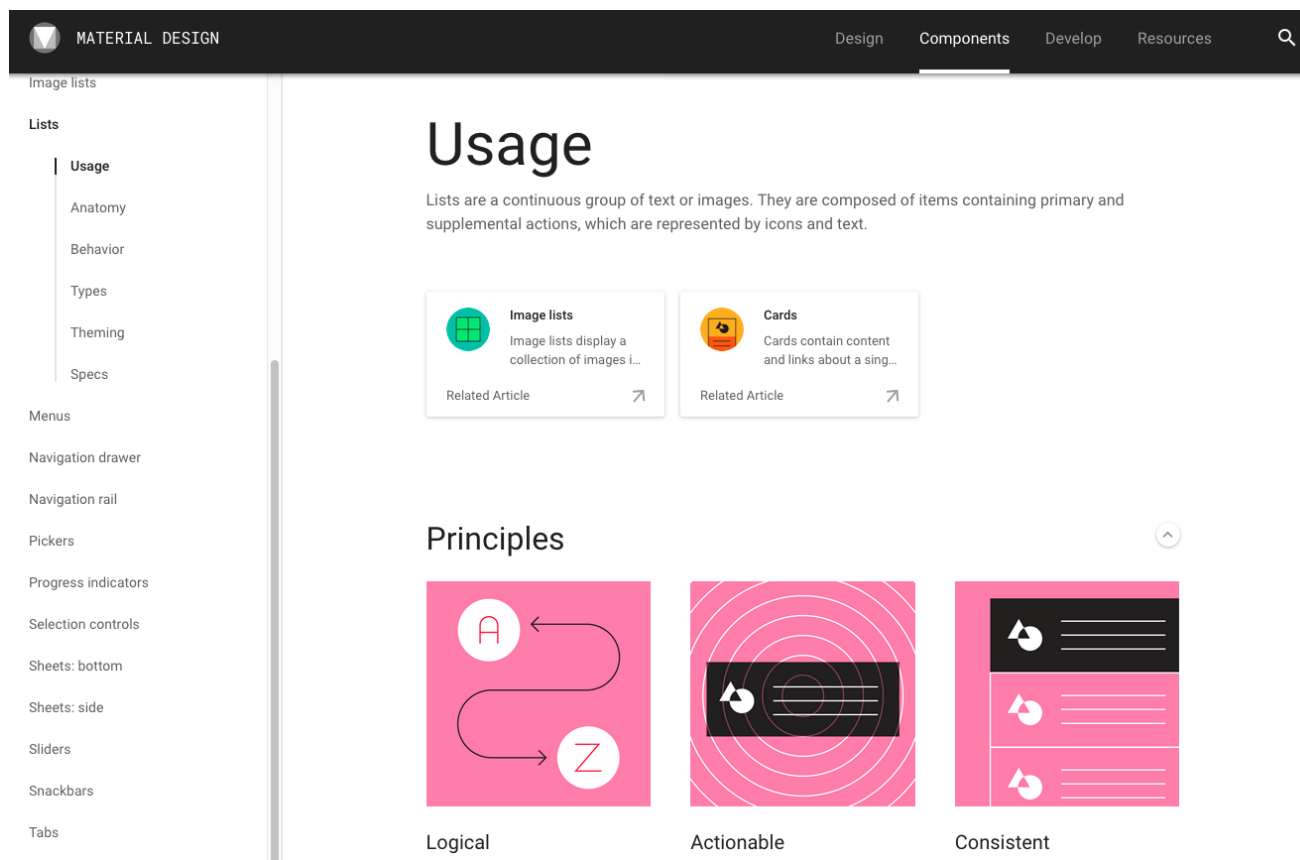
Aktuelles App-Design-Konzept: Material Design

- Design-Vorgaben werden ständig weiter entwickelt
 - Aktuelle Vorgaben: <https://material.io/design/>



Doku zu einzelnen Komponenten

- Android-Doku erläutert, wie Apps designed sein sollen, wie Widgets und Styles aussehen, wie einsetzen usw.
 - z.B. Listen (<https://material.io/design/components/lists.html>)



Wichtiges Element "App Bar"



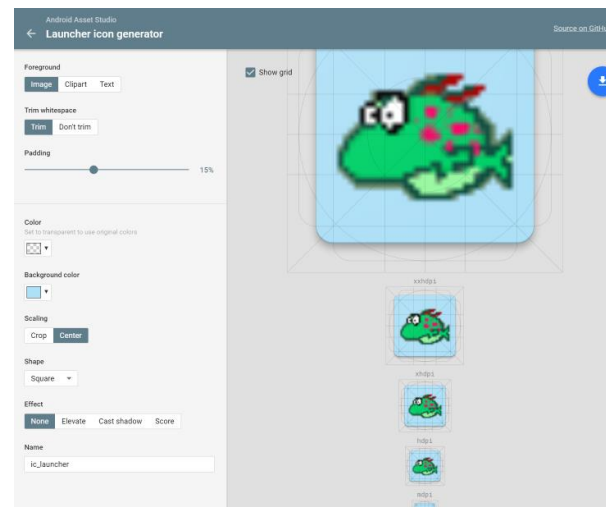
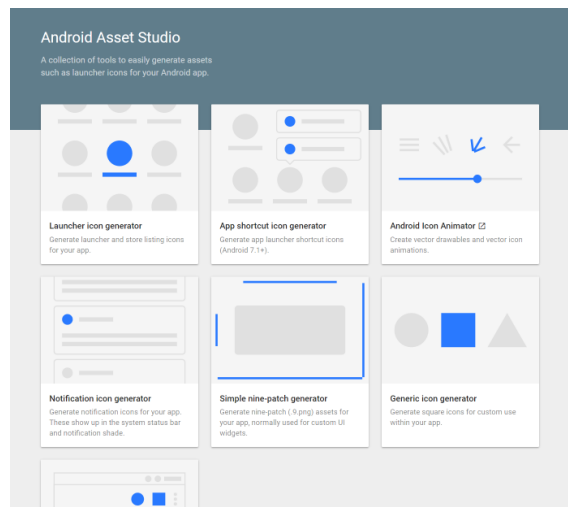
"The app bar, also known as the action bar, is one of the most important design elements in your app's activities, because it provides a visual structure and interactive elements that are familiar to users. Using the app bar makes your app consistent with other Android apps, allowing users to quickly understand how to operate your app and have a great experience. The key functions of the app bar are as follows:

- A dedicated space for giving your app an identity and indicating the user's location in the app.
- Access to important actions in a predictable way, such as search.
- Support for navigation and view switching (with tabs or drop-down lists)."

Von <https://developer.android.com/training/appbar/index.html>

Online Tool: Android Asset Studio

- Hilfsprogramme um custom Icons für Launcher, Notifications, App-Bar, etc. zu erzeugen
- Erstellt Icons in allen Auflösungen > ZIP download



<https://romannurik.github.io/AndroidAssetStudio/index.html>

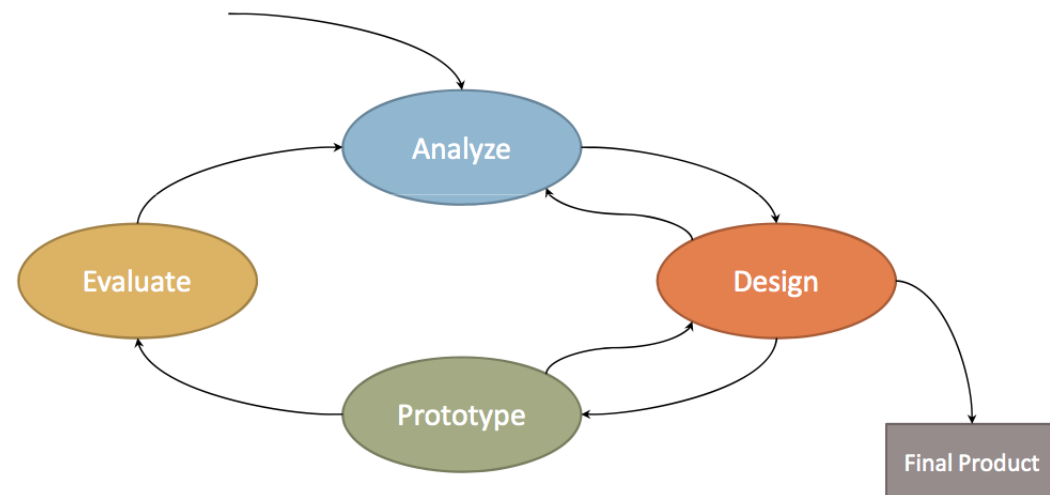


<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Usability & Prototyping

Mini-Exkurs: Usability & Prototypen

- Design = Iterativer Prozess
- Prototypen helfen!
 - Kommunikation mit Kunden: Benutzer sieht etwas
 - Verständnis EntwicklerIn für App/Anwendungsdomäne
- Einfaches iteratives Design-Model:

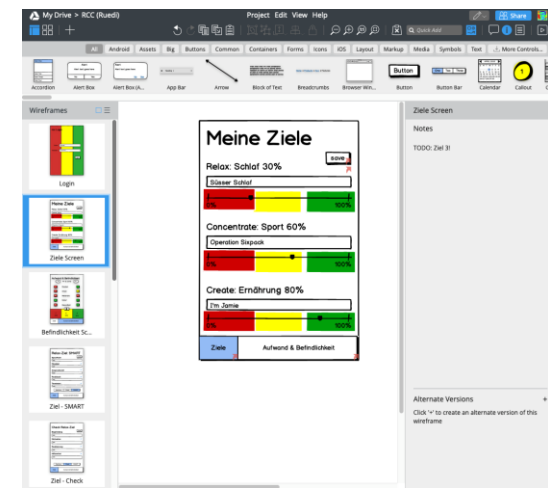


Prototypen erstellen & iterieren!

- (Papier-)Prototypen benutzen!
 - App durchspielen, durchklicken & durchdenken!
 - ☞ Apps werden *besser* und *durchdachter* 😊
 - App auf Papier entwerfen oder mit SW-Tool...
 - z.B. Balsamiq Wireframes for Google Drive (<https://balsamiq-wireframes.appspot.com>)
 - Namen: Wireframes / Mocks / Storyboards / ...

☞ Verwenden wir im Team-Projekt

- (Mehr dazu im USAB oder UCD-Modul)
- Melden Sie sich per Email bei Ruedi Arnold, falls Sie ein Balsamiq-Konto wollen





<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Android Jetpack & AppCompat (Support Library)

Android Jetpack

<https://developer.android.com/jetpack>

Android Jetpack

Jetpack is a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices so that developers can focus on the code they care about.

GET STARTED USING JETPACK



Why use Android Jetpack?



Follow best
practices

Built around modern design



Eliminate
boilerplate
code



Reduce
fragmentation

Reduces complexity with

Android Jetpack



Foundation



Architecture



Behavior



UI

■ Sammlung nützlicher Hilfsbibliotheken 😊

- D.h. viele interessante Bausteine: **AppCompat**, Data Binding, Lifecycles, LiveData, Navigation, Paging, Room, ViewModel, WorkManager, ...
- Verfügbar seit Api 28
- Harmonisiert gut mit Kotlin: erlauben kompakten, aussagenkräftigen Code

Einige kennen wir schon 😊



JetPack "AppCompat" (ehem. Support Library)

"Allows access to new APIs on older API versions of the platform (many using Material Design)."

Von <https://developer.android.com/jetpack/androidx/releases/appcompat>




AppCompat

Degrade gracefully on older versions of Android

Beispiel: Action Bar (App Bar)



- ActionBar gibt's erst seit API-Level 11
- Dank Support Library trotzdem verfügbar ab API-Level 7 (d.h. Android 2.1)
 - Netter Warnhinweis dazu in der Doku der Support Library (<http://developer.android.com/tools/support-library/setup.html>):

 **Caution:** When using classes from the Support Library, be certain you import the class from the appropriate package. For example, when applying the **ActionBar** class:

- `android.support.v7.app.ActionBar` when using the Support Library.
- `android.app.ActionBar` when developing only for API level 11 or higher.



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Publizieren von Android-Apps: Packaging, Signing & Release

The BEST of Monty Python's Flying Circus

App-Veröffentlichung

- Alles gut dokumentiert, primäre Quellen:

- **Developer workflow basics**

<https://developer.android.com/studio/workflow>

- **Publish your app**

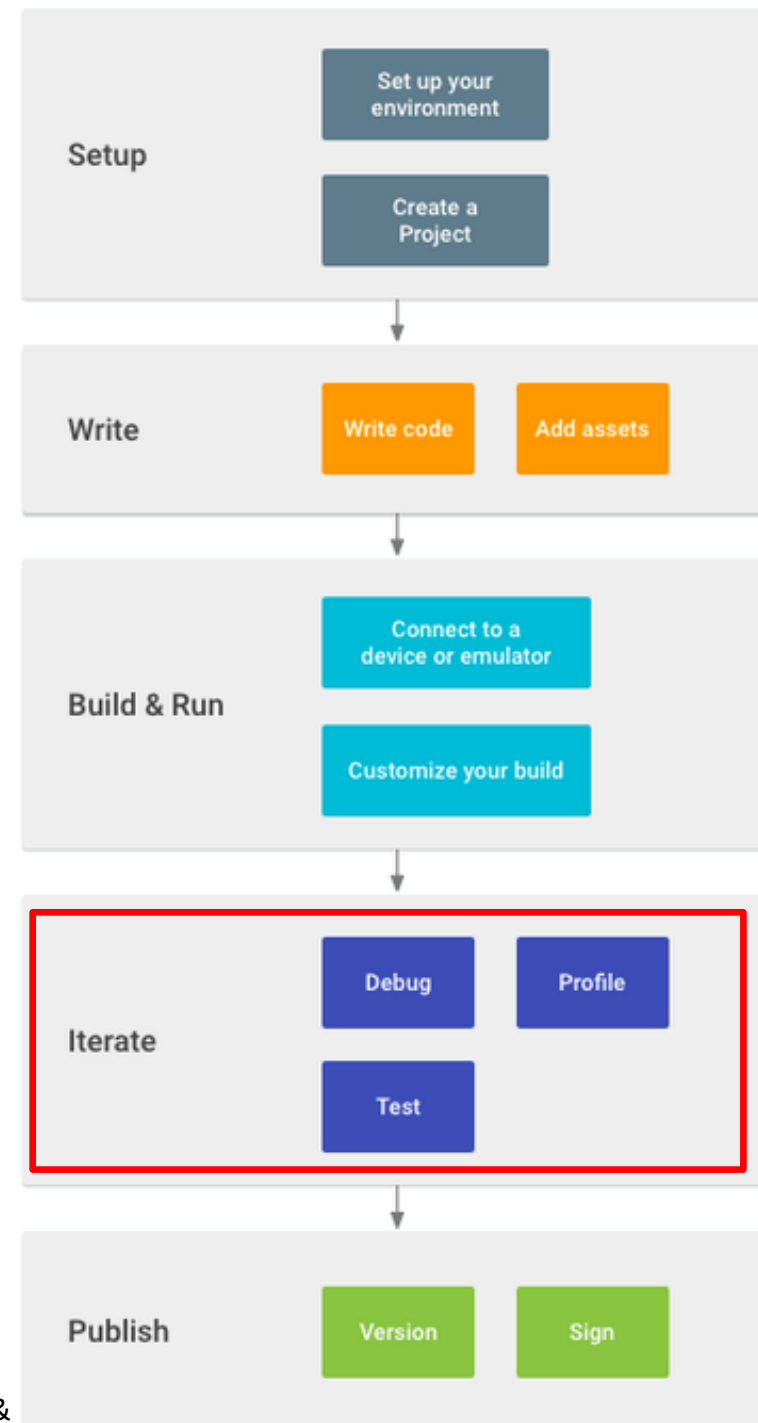
<https://developer.android.com/studio/publish>

- **How to publish, manage, and distribute your app worldwide**

<http://developer.android.com/distribute/googleplay/start.html>

- **Alternative Distribution Options**

<https://developer.android.com/distribute/marketing-tools/alternative-distribution>



Google Play Store



- Googles Market Applikation
 - Vertrieb und Installation von Android Applikationen (+ Music, Books, Movies, ...)
 - Grösster Market für Android mit > 3 Mio Apps
 - Pendant zu Apple App-Store
- Jeder kann Apps im Play Store veröffentlichen

- Voraussetzungen:

- Google Account (gratis)
- Google Play Publisher Account (\$25, einmalig)
- Google Payments Merchant Account
 - braucht's nur für kostenpflichtige Apps

Empfehlung: Neuen Google-Account erstellen

d.h.: Entwickler*in erhält 70%

Gebühr: **30%** für Google bzw. **15%** bei kleiner Firma (d.h. <= 1Mio \$ / Y)

Grob-Vorgehen



1. Code Cleanup, Release vorbereiten

- Versionierung, Internationalisierung, Logging & Debug-Ausgaben entfernen, usw.

2. Release erstellen und signieren

Mit "scharfem" Schlüssel
= Developer Identität (od. Firma)

3. Werbematerial vorbereiten

- Screenshots, Hi-Res Icons, Promo, Video, Website, Google Ads, Whatever, ...

4. Distributionseinstellungen setzen

Digital Content: Game Levels, Zeitungsartikel, Bilder, Videos, etc.

- Content Rating, Verfügbarkeit, Grösse
- Bezahlung: Gratis, Kostenpflichtig, In-App?

Auch möglich in
Gratis-Apps!

5. Release hochladen und veröffentlichen

Die offizielle Launch-Checkliste

1. Understand the Developer Program Policies
2. Prepare your developer account
3. Plan for localization
4. Plan for simultaneous releases
5. Test against the quality guidelines
6. Target a recent API level
7. Build your Android App Bundle
8. Run internal tests
9. Plan your app's Play store listing
10. Build interest in your app or game with pre-registration
11. Upload your Android App Bundle to the closed or an open test track
12. Define your app's device compatibility
13. Check the pre-launch reports
14. Setup your app's price and countries of distribution
15. Opt-in to the right distribution options
16. Set up your in-app products and subscriptions
17. Determine your app's content rating
18. Final checks and publishing
19. Promote your app
20. Browse and reply to user reviews
21. Check your vitals
22. Woohoo, you've launched! What next?

<https://developer.android.com/distribute/best-practices/launch/launch-checklist>

APK = Android Application Package

"Android Package (APK) is the package file format used by the Android operating system for distribution and installation of mobile apps, mobile games and middleware. APK is analogous to other software packages such as APPX in Microsoft Windows or a Debian package in Debian-based operating systems. To make an APK file, a program for Android is first compiled using Android Studio[2], and then all of its parts are packaged into one container file. **An APK file contains all of a program's code (such as .dex files), resources, assets, certificates, and manifest file.** As is the case with many file formats, APK files can have any name needed, provided that the file name ends in the file extension '.apk'."

http://en.wikipedia.org/wiki/Android_application_package

APK = Android Application Package

"Android Package (APK) is the package file format used by the Android operating system for distribution and installation of mobile applications, games and middleware. APK is similar to other file formats such as APPX in Microsoft Windows operating systems. To create an APK, a program must first be compiled using the Android Studio IDE. The compiled code and other resources are then packaged into one container file. **An APK file contains a program's code (such as .dex files), resources, assets, certificates, and manifest file.** As is the case with many file formats, APK files can have any name needed, provided that the file name ends in the file extension '.apk'."

http://en.wikipedia.org/wiki/Android_application_package

Neues Format:

android app bundle



"The Android App Bundle is Android's new, official publishing format that offers a more efficient way to build and release your app. The Android App Bundle lets you more easily deliver a great experience in a smaller app size, which can improve install success and reduce uninstalls. It's easy to switch. You don't need to refactor your code to start benefiting from a smaller app. And once you've switched, you'll benefit from modular app development and customizable feature delivery."

Von <https://developer.android.com/platform/technology/app-bundle/>

Play Store macht APK aus AAB

About Android App Bundles

★ **Important:** From August 2021, new apps will be required to publish with the [Android App Bundle](#) on Google Play. New apps larger than 150 MB are now supported by either [Play Feature Delivery](#) or [Play Asset Delivery](#).

An *Android App Bundle* is a publishing format that includes all your app's compiled code and resources, and defers APK generation and signing to Google Play.

- D.h. APKs bleiben (natürlich), werden jedoch neu von Google Play generiert...

<https://developer.android.com/guide/app-bundle/>

Signieren von Apps



Aber nicht mit (default)
Developer-Key aus IDE

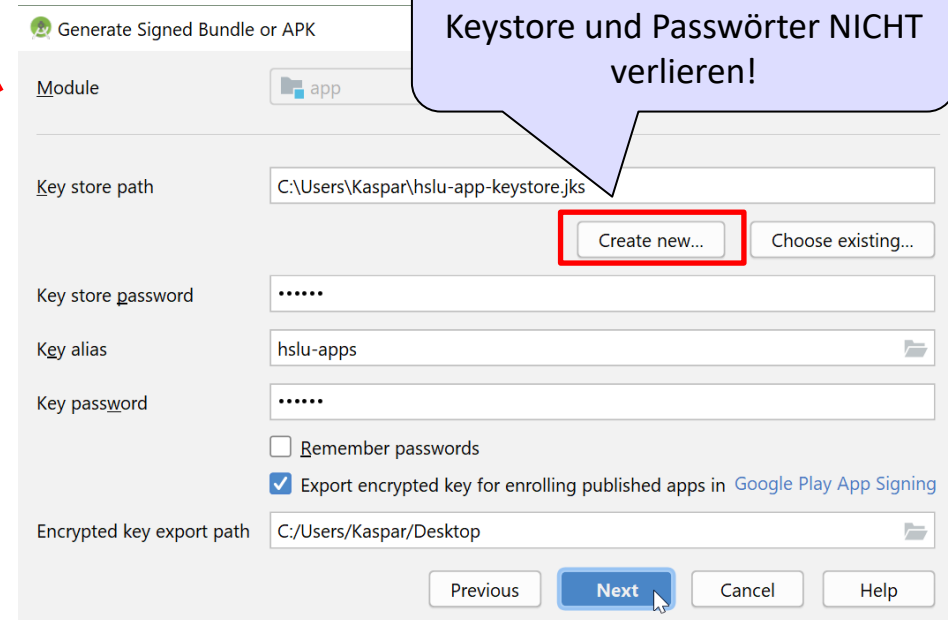
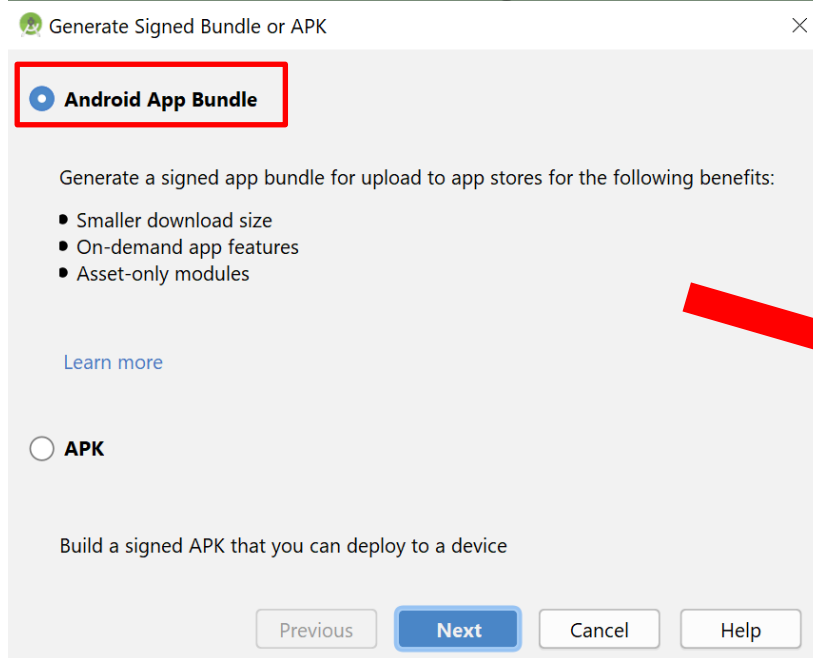
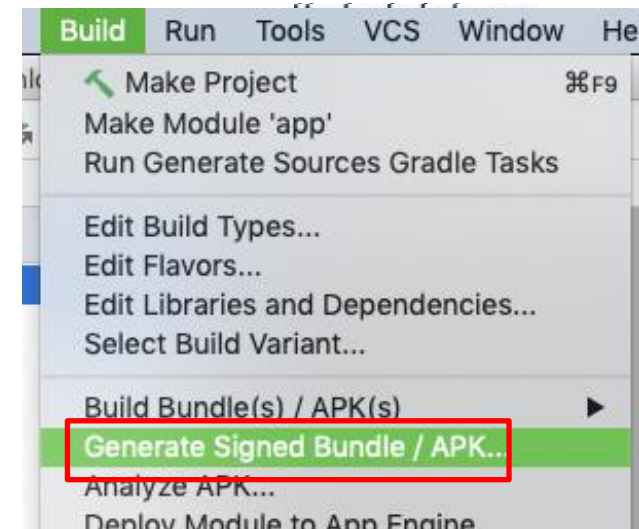
- Play Store Applikationen **müssen signiert sein**
 - Self-signing erlaubt (kein autorisiertes Zertifikat nötig)
 - Benötigt wird: Keystore mit private key(s)
 - Android-Studio-Wizard hilft beim Erstellen eines Keystores und eines privaten Schlüssels (=Zertifikat)
- Updates müssen mit gleichem Zertifikat signiert sein!
 - Wenn ja: Problemloses Update möglich
 - Wenn nein: App muss mit anderem Package-Namen als komplett neue App (!!!) installiert werden

**D.h. PROD
Schlüssel NICHT
verlieren!!!**

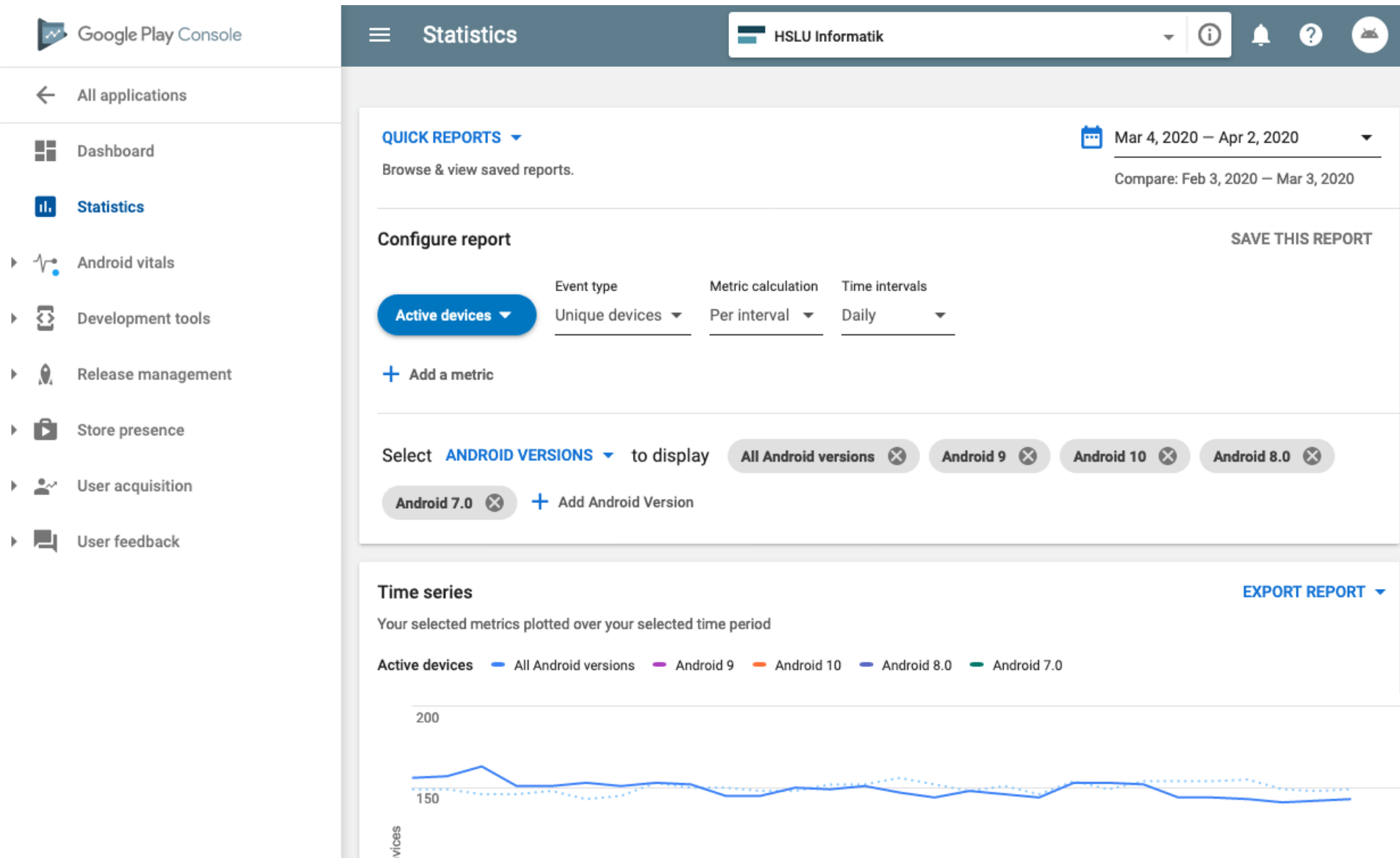
Android Studio: APK-Wizard

Menu: "Build >

Generate Signed Bundle / APK..."



Demo: Google Play – Developer Console



Andere Distributions-Möglichkeiten

- Alternativ zur Publikation bei Google Play können Apps wie folgt veröffentlicht werden:
 - Alternative Stores
 - z.B. AppBrain, Amazon Appstore for Android
 - App-Center (<https://appcenter.ms/>) oder Cloud Firestore (<https://firebase.google.com/docs/firestore/>)
 - Über eigene Webseite (Link auf .apk Datei) oder als Email-Anhang

Praktisch für Verteilung unter Freunden
 - Wichtig dazu: Gerät muss Installation aus "Unbekannter Quelle" erlauben: Settings > Security > Device Admin. > Unknown Sources

Test before release!!!

- Apps manuell & automatisch testen
 - Sollte selbstverständlich sein 😊
 - Siehe Android 7 (nächste Woche) für autom. Testing

- Manuell auf mindestens einem Gerät

- Besser: mehrere Geräte...

Aufwand nicht zu unterschätzen für
professionelle/kommerzielle Apps!

- ...mit unterschiedlicher Bildschirmauflösung, inkl. Tablett
 - ...mit unterschiedlichen Android-Versionen
 - ...von unterschiedlichen Herstellern

☞ Fazit: Sicherstellen, dass ein App auf vielen Geräten & Android-Versionen läuft, bedeutet viel Aufwand!



<http://en.wikipedia.org/wiki/File:Somethingdifferent.jpg>

Übung 6

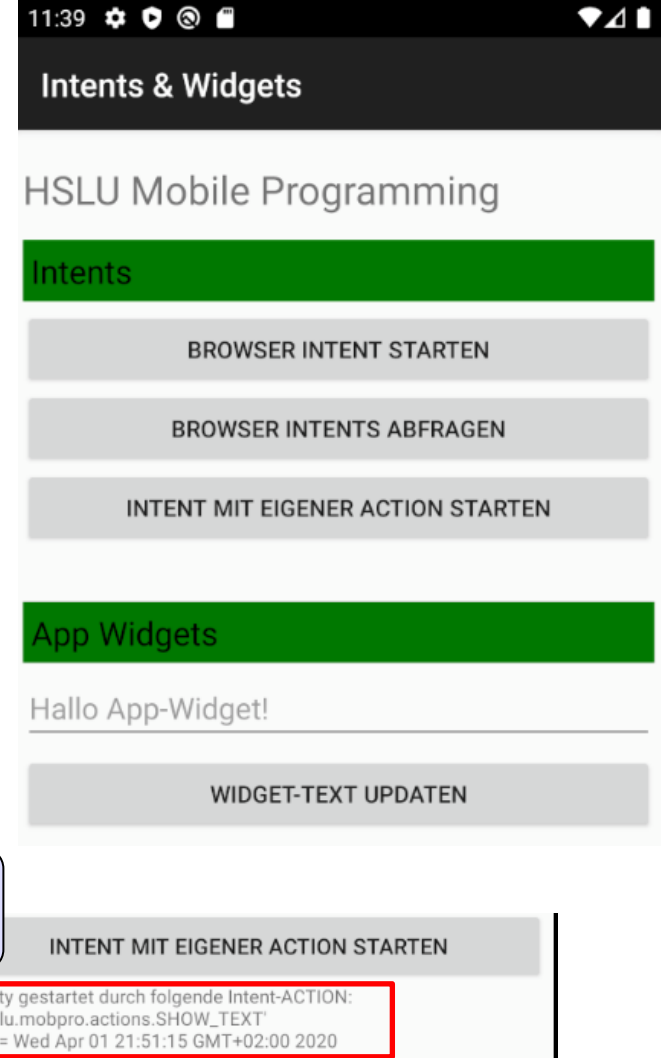
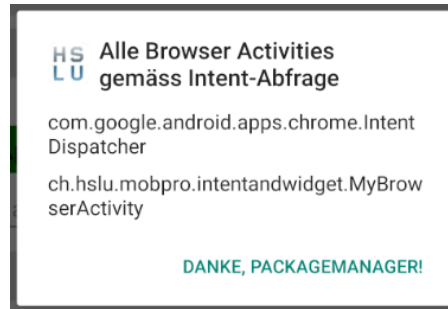
Zur Übung 6

■ Intents

- Browser-Intent abfangen und eigene Activity anzeigen
- Alle Empfänger für Browser- Intents auflisten
- Eigene Intent-Action aufrufen

■ Eigenes App-Widget

- Notwendige Deklarationen
- Inkl. Provider-Klasse
 - Spezieller Broadcast-Receiver
 - Aktualisierung via Broadcast



...auch aus
anderer App! 😊

