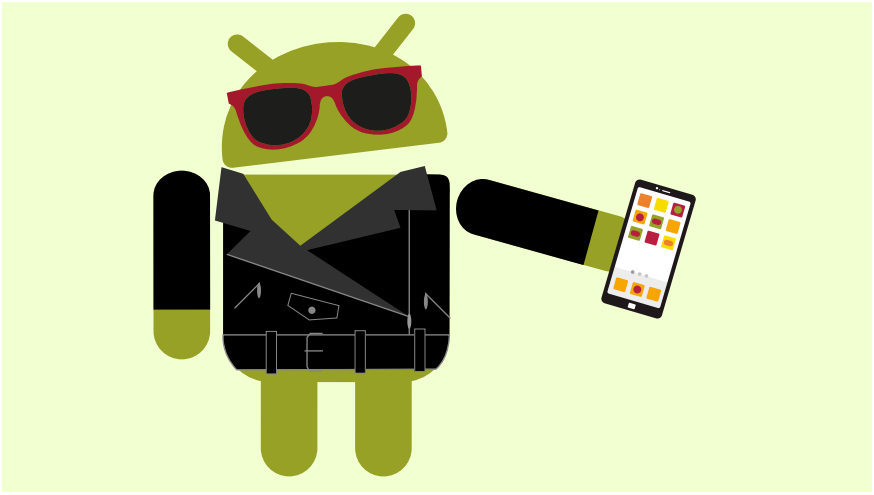


Android 12: Das ändert sich in der Anwendung und beim Entwickeln

Neuer Look

Thomas Künneth

Einmal pro Jahr veröffentlicht Google eine neue Android-Hauptversion. Dieses Mal punktet sie mit einer komplett überarbeiteten Designsprache, Splash Screens und einer geräteweiten Suche.

Alljährlich im Herbst bringt Google eine neue Android-Hauptversion. Von Android 12 lag zum Redaktionsschluss die Beta-5-Version vor, die erfahrungsgemäß alle Funktionen der finalen Version enthält. Wie üblich erhalten Googles Pixel-Telefone (ab Pixel 3) als Erstes das Update. Einige Funktionen sind zunächst diesen vorbehalten, üblicherweise werden aber die meisten davon im Laufe der Zeit auf anderen Geräten nachgereicht. Mehrere Hersteller haben am Betaprogramm teilgenommen, unter anderen ASUS, OnePlus, realme, Xiaomi und ZTE. Deren aktuelle Modelle dürften das fertige Android 12 bald erhalten. Gegen Ende des Jahres sollten weitere wichtige Geräte versorgt sein. Weil die Zahl an Modellen zu groß ist, werden aber einige leer ausgehen: Viele Hersteller sichern nur zwei Major Updates zu.

Die auffälligste Neuerung der Oberfläche sind sicher die vielen abgerundeten Ecken. Sie gehören zu Material You, der vielleicht umfassendsten Designänderung in der Geschichte Androids. Material You ist die konsequente Weiterentwicklung der 2014 eingeführten Designsprache Material Design (siehe ix.de/z6nq). Google hat die Verwendung von Farben, Licht und Schatten, Animationen und Formen vollständig überarbeitet. Ziel war, Android ausdrucksstärker, dynamischer und persönlicher zu machen. So lässt sich die Gerätefarbpalette individualisieren: Aus dem aktuellen Hintergrundbild ausgewählte Farben wendet Android auf Systembedienelemente an, zumindest auf Pixel-Telefonen. Derzeit ist nämlich noch unklar, wie viel von Material You schon jetzt anderen Herstellern zur Verfügung steht. Wie schon Material Design soll die neue Designsprache aber im Laufe der Zeit auf den unterschiedlichsten Plattformen zum Einsatz kommen und möglichst viele Geräteklassen abdecken.

Gut sichtbar sind diese Änderungen in den Schnelleinstellungen: Aus unscheinbaren Icons wurden (vielleicht etwas zu große) abgerundete Buttons mit Symbol, Beschriftung und gegebenenfalls scrollenden Status (Abbildung 1). Mit einem Klick lassen sich an dieser Stelle nun auch Mikrofon und Kamera ein- und ausschalten. Privatsphäre und Datenschutz nehmen seit Jahren viel Raum bei Android-Updates ein, und dieses Release bildet keine Ausnahme.

Indikatoren zeigen Zugriff auf Kamera und Mikrofon

Praktisch ist das neue Privatsphäredashboard in den Systemeinstellungen unter Datenschutz (Abbildung 2). Es zeigt an, welche Apps zuletzt welche Berechtigungen genutzt haben. Falls nötig, kann man diese schnell entziehen. Ebenfalls neu sind – die von iOS 14 abgeschauten – Indikatoren bei Zugriffen auf Mikrofon und Kamera. Verwendet man diese Komponenten, erscheint in der rechten oberen Ecke ein kleiner grüner Kreis, aus dem durch Öffnen des Benachrichtigungsbereichs Symbole werden. Einmaliges Antippen zeigt sie als Liste an, ein zweites öffnet die App-Berechtigungen.

Mittlerweile setzen viele Android-Funktionen auf maschinelles Lernen und künstliche Intelligenz. Aus Datenschutzgründen sind sie lokal auf dem Gerät auszuführen. Das Erkennen von Musikstücken und das Einblenden von Liveuntertiteln sind schon so umgesetzt. Android 12 lagert sie in einen vom Rest des Systems abgeschotteten Bereich aus, der Private Compute Core heißt. Darauf lässt sich nur über wenige wohldefinierte Schnittstellen zugreifen. Sensible Daten wie Audio- oder Videoströme sind damit besser vor allzu neugierigen Anwendungen geschützt.

iX-TRACT

- Mit dem Release 12 setzt Google in Android auf die neue Designsprache Material You. Kennzeichen sind neben der individualisierbaren Farbpalette die neuen Formen mit abgerundeten Ecken.
- Die neue geräteinterne Suchmaschine AppSearch ist mehrsprachig und arbeitet auch offline.
- Auf maschinelles Lernen und künstliche Intelligenz setzende Android-Funktionen sind nun in einen geschützten Bereich mit dem Namen Private Compute Core ausgelagert, auf den man nur über definierte APIs Zugriff erhält.

Darüber hinaus erfreut Android 12 mit weiteren neuen und verbesserten Funktionen. Beispielsweise gibt es einen systemweiten Einhandmodus. Ist er in den Einstellungen unter „System / Gesten und Bewegungen“ aktiviert, schiebt eine Wischgeste über den schmalen Balken in der unteren Bildmitte den oberen Bildschirmbereich ganz nach unten und macht ihn so bequem mit einer Hand zugänglich. Wischen in die andere Richtung beendet die Einhandbedienung.

Ebenfalls nützlich sind scrollbare Screenshots. Nach dem gleichzeitigen Drücken von Einschalt- und Leiserknopf erscheint der Button „Mehr aufnehmen“, sofern der Bildschirm scrollbar ist. Danach wählt man in einer Vorschau den zu verwendenden Bereich aus. Das weitere Bearbeiten des Screenshots erfolgt dann wie gehabt.

Scrolling Screenshots entwickeln

Einige der bisher vorgestellten Neuerungen haben auch Auswirkungen auf die Entwicklung. Entwicklerinnen und Entwickler können dem System beim Erstellen scrollbarer Screenshots helfen, indem sie die App das Interface android.view.ScrollCaptureCallback implementieren lassen. Auch die Zugriffsindikatoren stehen ihnen offen, zumindest was das Ermitteln ihrer Position betrifft. Hierzu gibt es in der Klasse `android.view.WindowInsets` die neue Methode `getPrivacyIndicatorBounds()`. Änderungen sind tabu.

Damit sich der Start von Apps harmonischer anfühlt, bringt Google mit Android 12 einheitliche Splash Screens, die es rudimentär seit Android Oreo (API-Level 26) gibt. Das Einfügen der Zeile

```
<item name="android:windowSplashscreen
                Content">@drawable/...</item>
```

in das App-Theme zeigte bisher ein einfaches Drawable an. Ab API-Level 31 sollte man stattdessen `android:windowSplashscreen AnimatedIcon` verwenden. Es erlaubt, animierte Vector Drawables zu nutzen.

`android:windowSplashScreenAnimationDuration` legt die Dauer der Animation in Millisekunden fest; der Wert sollte 1000 nicht überschreiten.

`android:windowSplashScreenIconBackgroundColor` setzt die Hintergrundfarbe des Icons und `android:windowSplashScreenBackground` die Farbe des Splash Screens.

`android:windowSplashScreenBrandingImage` zeigt ein Firmen- oder Markenlogo am unteren Bildschirmrand an.

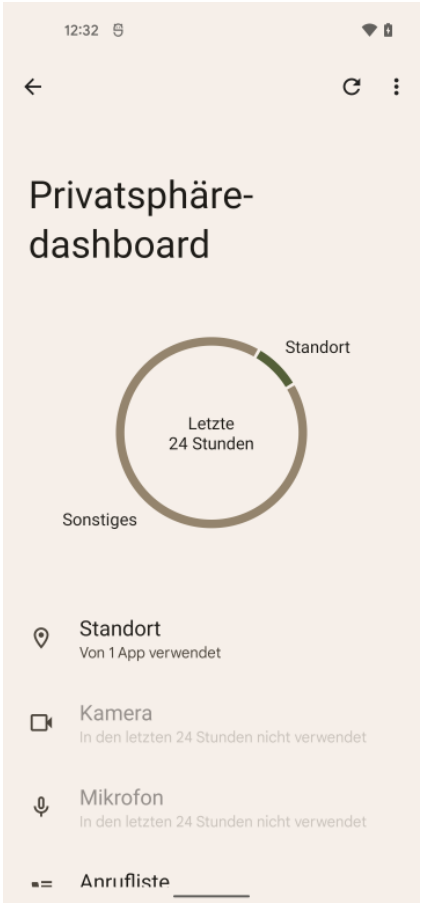
Eine einfache Theme-Konfiguration ist in Listing 1 zu sehen.

Android zeigt Splash Screens an, sofern der App-Prozess nicht läuft (Kaltstart) oder die Activity erzeugt werden muss (Warmstart). Nach kurzen App-Wechseln ist kein Splash Screen zu sehen. Sobald die App mit ihrer Initialisierung fertig ist und die Benutzeroberfläche anzeigt, verschwindet der Splash Screen. Um seine Anzeigedauer zu verlängern, kann man mit `addOnPreDrawListener()` einen Listener setzen. Der Splash Screen sollte so kurz wie möglich zu sehen sein. Zu ihm gehören neben den Icons und den Farben eine Beginn- und Endeanimation. Die Beginnanimation ist vom System vorgegeben und nicht veränderbar, die Endeanimation lässt sich hingegen beeinflussen (Listing 2).

Die seit Kurzem von Google angebotene Komponente Jetpack Core Splash Screen ermöglicht einheitliche Splash Screens auch auf älteren Plattformen. Ihre Verwendung ist in der App JetpackSplashScreenDemo zu sehen (siehe ix.de/z6nq). Dabei muss man darauf achten, in der Theme-Datei nicht das Präfix `android:` für Iconfarben und den Hintergrund zu verwenden. Außerdem steht derzeit `windowSplashScreenIconBackgroundColor` nicht zur Verfügung. Der größte Unterschied besteht aber darin, dass das Theme der App und der Start-Activity `Theme.SplashScreen` als `parent`-Attribut enthalten und

```
<item name="postSplashScreenTheme">@style/
          Theme.JetpackSplashScreenDemo</item>
```

Das neue Aussehen von Android 12 zeigt sich in den Schnelleinstellungen: Aus Icons wurden Buttons mit abgerundeten Ecken (Abb. 1).



Über das neue Privatsphäredashboard erhalten Nutzerinnen und Nutzer Auskunft darüber, welche Apps wann bestimmte Komponenten verwenden (Abb. 2).

Listing 1: Splash Screen im App-Theme konfigurieren

```
<?xml version="1.0"?>
<resources>
    <style parent="Theme.AppCompat.DayNight.DarkActionBar" name="Theme.SplashScreenDemo">
      <item name="colorPrimary">@color/purple_500</item>
      <item name="colorAccent">@color/teal_200</item>
    </style>

    <style parent="Theme.SplashScreenDemo" name="Theme.SplashScreenDemo.Activity">
      <item name="android:windowSplashScreenBackground">#ff808080</item>
      <item name="android:windowSplashScreenIconBackgroundColor">#ffe0e0e0</item>
      <item name="android:windowSplashScreenAnimatedIcon">@drawable/green_clock</item>
      <item name="android:windowSplashScreenAnimationDuration">1000</item>
    </style>
</resources>
```

auf das eigentliche App-Theme verweisen muss.

Die neue Suchmaschine AppSearch funktioniert auch offline

Ebenfalls neu ist eine hochperformante Suchmaschine direkt auf dem Gerät. Apps legen Teile ihrer Daten strukturiert in ihr ab und können über Volltextsuche darauf zugreifen. Eine Recherche ist auch möglich, wenn das Gerät nicht mit dem Internet verbunden ist. AppSearch unterstützt mehrere Sprachen und ein Ranking nach Relevanz. Die Suchmaschine steht in zwei Ausprägungen zur Verfügung: lokal (nur für die nutzende App) und zentral (systemübergreifend). Die lokale Variante ist auch unter älteren Plattformversionen verwendbar, den zentralen Index gibt es erst ab Android 12.

Google rät ausdrücklich, die neue Jetpack-Bibliothek AppSearch zu verwenden (siehe ix.de/z6nq). Sie steht derzeit als Alphaversion zur Verfügung. Es ist deshalb noch mit Änderungen an der API zu rechnen. Wie man eine Datenbank anlegt, Informationen speichert und danach sucht, zeigt die App AppSearchDemo (siehe ix.de/z6nq).

Data Classes und ihre Eigenschaften sind annotiert

Für den Zugriff auf AppSearch ist eine Session erforderlich, die sich mit `PlatformStorage.createSearchSession()` oder `LocalStorage.createSearchSession()` erzeugen lässt. Die zu speichernden Informationen befinden sich in Data Classes, die mit `@Document` annotiert sind. Deren Eigenschaften werden ebenfalls annotiert, etwa mit `@Document.Id` und `@Document.Namespace`. Während des Builds entstehen durch den Annotation Processor oder das Kotlin-Tool kapt die zur Laufzeit relevanten Klassen. Für Entwicklerinnen und Entwickler ist das aber unerheblich, sie nutzen stets ihre Data Classes (im Beispiel MyDocument). Jedes Dokument muss zu einem Schema gehören.

```
PutDocumentsRequest.Builder().
                    addDocuments(doc).build()
```

erzeugt eine `PutDocumentsRequest`-Instanz und übergibt sie an die Session. Ganz ähnlich funktioniert das Suchen. Zunächst erstellen und konfigurieren Entwickler den Builder `SearchSpec.Builder()`. Dessen Methode `build()` liefert ein `SearchSpec`-Objekt, das sie an die Session-Methode `search()` weiterreichen.

Listing 2: Activity der App SplashScreenDemo

```
package com.thomaskuenneth.splashscreendemo

import android.animation.AnimatorSet
import android.animation.ValueAnimator
import android.annotation.SuppressLint
import android.os.Bundle
import android.util.Log
import android.view.View
import android.view.ViewTreeObserver
import android.view.animation.AccelerateDecelerateInterpolator
import android.window.SplashScreen
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.lifecycleScope
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch


private const val TAG = "MainActivity"

@SuppressLint("CustomSplashScreen")
class SplashScreenDemoActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        var ready = false
        lifecycleScope.launch {
            delay(5000)
            ready = true
        }
        val content = findViewById<View>(android.R.id.content)
        content.viewTreeObserver.addOnPreDrawListener(
            object : ViewTreeObserver.OnPreDrawListener {
                override fun onPreDraw(): Boolean {
                    return if (ready) {
                        content.viewTreeObserver.removeOnPreDrawListener(this)
                        true
                    } else {
                        false
                    }
                }
            }
        )

        val l = SplashScreen.OnExitAnimationListener { view ->
            Log.d(TAG, view::class.java.name)
            view.iconView?.let { icon ->
                val animator = ValueAnimator
                    .ofInt(icon.height, 0)
                    .setDuration(2000)
                animator.addUpdateListener {
                    val value = it.animatedValue as Int
                    icon.layoutParams.width = value
                    icon.layoutParams.height = value
                    icon.requestLayout()
                    if (value == 0) {
                        view.remove()
                    }
                }
                val animationSet = AnimatorSet()
                animationSet.interpolator = AccelerateDecelerateInterpolator()
                animationSet.play(animator)
                animationSet.start()
            }
        }
        splashScreen.setOnExitAnimationListener(l)
    }
}
```

Die API von Jetpack AppSearch setzt auf die Implementierungen des Interface java.util.concurrent.Future – ein im Java-Umfeld modernes Programmierparadigma. Schaut man sich aber andere neue Jetpack-Komponenten an, dominieren dort Kotlin-Flows. Einer so dringend nötigen Vereinheitlichung von Programmierstilen trägt das nicht Rechnung. Google hat signalisiert, Jetpack AppSearch auf Flows umstellen zu wollen. Konkrete Pläne sind aber nicht bekannt.

Neben den großen Neuerungen gibt es auch viele kleine Verbesserungen. Beispielsweise kann man mit dem Game Mode Spiele im Hinblick auf Performance,

Listing 3: Eine Activity ergänzen und einrichten

```
<activity android:name=".DataAccessRationaleActivity"
     android:permission="android.permission.START_VIEW_PERMISSION_USAGE"
     android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.VIEW_PERMISSION_USAGE" />
        <action android:name="android.intent.action.VIEW_PERMISSION_USAGE_FOR_PERIOD" />
      </intent-filter>
</activity>
```

Darstellung und Batterieverbrauch optimieren. Mit den Rounded-Corner-APIs können Anwender Mittelpunkt und Radius abgerundeter Ecken abfragen und so das Abschneiden von Bedienelementeteilen auf Bildschirmen mit abgerundeten Ecken verhindern. Eine vereinheitlichte Programmierschnittstelle erlaubt das Konsumieren angereicherter Inhalte (Texte mit Grafiken, Animationen, Emojis) über das Systemclipboard, Drag-and-drop und die Tastatur. Die neuen Bluetoothberechtigungen BLUETOOTH_SCAN, BLUETOOTH_ADVERTISE und BLUETOOTH_CONNECT vereinfachen Apps, die keine Standortinformationen abfragen möchten, die Interaktion mit Bluetoothgeräten. Um zu erklären, warum eine bestimmte Berechtigung notwendig ist, können Apps hierfür eine Activity hinzufügen und über Intent-Filter in der Manifestdatei konfigurieren (Listing 3). Auf der Berechtigungsseite der App zeigt Android dann ein klickbares Icon an, das auf die Activity verweist.

Die aktuellen Build-Tools ermöglichen endlich die Nutzung von Java-11-Features. Das lohnt nicht nur für in Java geschriebene Projekte. Auch Kotlin-Code kann von Verbesserungen in Java profitieren. Hierzu müssen Entwicklerinnen und Entwickler in der modulspezifischen Datei build.gradle nur den folgenden Block hinzufügen:

```
kotlinOptions {
    ...
    jvmTarget = '11'
    ...
}
```

Damit lassen sich aber nicht alle APIs von Java 11 verwenden. Immerhin hat Google eine Reihe fehlender Methoden hinzugefügt.

Fazit: neues Design und Datenschutzfunktionen

Verglichen mit Android 11 ist das diesjährige Update wieder ein größeres Release. Die umfassende Überarbeitung der Oberfläche wird sicher nicht nur Begeisterung auslösen, zumal viele Apps ihr eigenes UI-Süppchen kochen und deshalb das neue Aussehen nicht automatisch übernehmen. Und noch ist unklar, wie viel Material You es auf Nicht-Pixel-Geräten überhaupt gibt. Tatsache ist: Der individuellere Touch tut Android gut. Die abgerundeten Ecken und die harmonische Farbgebung wirken erfrischend.

Die Arbeiten unter der Haube machen das System sicherer und stabiler. Wie gut Private Compute Core tatsächlich ist, muss sich zeigen. Aber jede Initiative zum Schutz sensibler Daten ist richtig und willkommen. Auch Entwickler haben viel auszuprobieren. Neben den hier im Artikel vorgestellten Neuerungen gibt es zahlreiche weitere, unter anderem einen erweiterten Dateizugriff über `getMediaUri()`, ein neues Verzeichnis für Sprachaufzeichnungen, Verbesserungen hinsichtlich Wi-Fi Aware und Companion Device Manager Profiles.

Schon seit mehreren Jahren findet viel Arbeit an Android außerhalb der Plattform und deren Veröffentlichungsrhythmus statt. Das neue UI-Framework Jetpack Compose ist seit Ende Juli 2021 in einer ersten Version stabil. Das heißt aber nicht, dass keine größeren Änderungen mehr zu erwarten sind. Entwicklerinnen und Entwickler müssen sich sogar auf noch kürzere Intervalle als bei der Plattform einstellen. Gleiches gilt für die vielen anderen Bibliotheken des Jetpack-Kosmos, deren Nutzung Google propagiert. Man kann neue Funktionen nutzen, bleibt aber kompatibel zu älteren Betriebssystemen. Ihr Anteil am Code einer App wird also stetig zunehmen. Was das bedeutet, wenn eine neue Bibliotheksversion Anpassungen am Code erfordert (und das geschieht nicht selten), mag man sich eigentlich kaum ausmalen.

Auch dieses Jahr nähern sich iOS und Android weiter an. Mal bringt Apple ein Feature zuerst, mal Google. Solange die jeweiligen Funktionen sich gut in das übrige System integrieren, kann das den Anwenderinnen und Anwendern nur recht sein. Beide Welten warten aber nach wie vor auf eine wirklich echte Neuerung. So schön also die beiden neuen Plattformen auch sind, letztlich betreiben beide derzeit nur Detailpflege, wenn auch auf sehr hohem Niveau. (nb@ix.de)

Quellen

App „AppSearchDemo“, Infos zu Material-You-Design und Jetpack AppSearch: ix.de/z6nq

Thomas Künneth

arbeitet als Head of Mobile für die MATHEMA GmbH. Neben zahlreichen Artikeln hat er drei Bücher über Android, Java und Eclipse veröffentlicht.