# Exploring App Installation APIs in Android

Aayush Gupta

# About Me

## Aayush Gupta

- Independent Contractor
- Android Developer and Developer Relations
- Senior Staff @ XDA Developers
- Full-time FOSS Maintainer and Contributor

# Agenda

- Intent-based App Installation
- PackageInstaller API
- Shell and Third-party Apps
- Bonus: Uninstalling Apps

# Intent-based App Installation

# Intent-based App Installation

- Supported by all Android versions (although deprecated)
- Easy to use, delegates heavy lifting to OS
- Limited configurability with modern requirements
- Doesn’t works with Split APK and Shared Libs

```
fun xInstall(file: File) {
    val intent: Intent

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
        intent = Intent(Intent.ACTION_INSTALL_PACKAGE)
        intent.data = getUri(file)
        intent.flags = Intent.FLAG_GRANT_READ_URI_PERMISSION or Intent.FLAG_ACTIVITY_NEW_TASK
    } else {
        intent = Intent(Intent.ACTION_VIEW)
        intent.setDataAndType(Uri.fromFile(file), "application/vnd.android.package-archive");
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
    }

    intent.putExtra(Intent.EXTRA_NOT_UNKNOWN_SOURCE, true)
    intent.putExtra(Intent.EXTRA_INSTALLER_PACKAGE_NAME, context.packageName)
    context.startActivity(intent)
}
```

# PackageInstaller API

# PackageInstaller API

- Supported since API 21 (Android 5.0)
- Powerful and complex compared to Intent-based app installation
- Supports installing Split APK and Shared Libs as well
- Allows updating apps without user intervention on new Android versions
- Not reliable on some brands such as Xiaomi

# PackageInstaller API

# PackageInstaller API

## Creating Session with Appropriate Params

- Installing app requires creating session first
- Developers specify required parameters and OS creates it
- Specifying package name for app being installed is required
- Can be opened any number of times (even across reboot) until abandoned
- Valid until abandoned manually or by OS automatically

```
val packageInstaller = context.packageManager.packageInstaller

val sessionParams = SessionParams(SessionParams.MODE_FULL_INSTALL).apply {
    setAppPackageName(downloadedApp.packageName)
    setInstallLocation(PackageInfo.INSTALL_LOCATION_AUTO)
    if (isOAndAbove()) {
        setInstallReason(PackageManager.INSTALL_REASON_USER)
    }
    if (isNAndAbove()) {
        setOriginatingUid(Process.myUid())
    }
    if (isSAndAbove()) {
        setRequireUserAction(SessionParams.USER_ACTION_NOT_REQUIRED)
    }
    if (isTAndAbove()) {
        setPackageSource(PACKAGE_SOURCE_STORE)
    }
    if (isUAndAbove()) {
        setInstallerPackageName(context.packageName)
        setRequestUpdateOwnership(true)
    }
}
val sessionId = packageInstaller.createSession(sessionParams)
```

# PackageInstaller API

## Copying Files

- Session requires copying all APKs using the openWrite method
- APKs must be for same package name for which session was created

```
val session = packageInstaller.openSession(sessionId)

try {
    Log.i("Writing splits to session for $packageName")
    getDownloadedAppFiles().forEach { file ->
        file.inputStream().use { input ->
            session.openWrite("${packageName}_${System.currentTimeMillis()}", 0, -1).use { output ->
                input.copyTo(output)
                session.fsync(output)
            }
        }
    }
} catch (exception: Exception) {
    session.abandon()
    postError(packageName, exception.localizedMessage, exception.stackTraceToString())
}
```

# PackageInstaller API

## Committing Session and Handling Status

- Committing session requires IntentSender
- IntentSender is responsible for handling different status of the session
  - Example scenarios include prompting user to install the app or notifying about an error
  - In case of failures, also responsible for cleanup tasks (if any)

```kotlin
fun commitInstall(packageName: String, sessionId: Int) {
    Log.i("Starting install session for $packageName")
    val session = packageInstaller.openSession(sessionId)
    session.commit(getCallBackIntent(packageName).intentSender)
    session.close()
}

private fun getCallBackIntent(packageName: String): PendingIntent {
    val callBackIntent = Intent(context, InstallerStatusReceiver::class.java).apply {
        action = InstallerStatusReceiver.ACTION_INSTALL_STATUS
        setPackage(context.packageName)
        putExtra(PackageInstaller.EXTRA_PACKAGE_NAME, packageName)
        addFlags(Intent.FLAG_RECEIVER_FOREGROUND)
    }
    val flags = if (isSAndAbove()) {
        PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_MUTABLE
    } else {
        PendingIntent.FLAG_UPDATE_CURRENT
    }

    return PendingIntent.getBroadcast(context, parentSessionId, callBackIntent, flags)
}
```

```kotlin
class InstallerStatusReceiver : BroadcastReceiver() {

    companion object {
        const val ACTION_INSTALL_STATUS =
            "com.example.app.data.receiver.InstallReceiver.INSTALL_STATUS"
    }

    override fun onReceive(context: Context, intent: Intent) {
        if (intent.action == ACTION_INSTALL_STATUS) {
            val packageName = intent.getStringExtra(PackageInstaller.EXTRA_PACKAGE_NAME)
            val status = intent.getIntExtra(PackageInstaller.EXTRA_STATUS, -1)
            val extra = intent.getStringExtra(PackageInstaller.EXTRA_STATUS_MESSAGE)

            // Exit early if package was successfully installed, nothing to do
            if (status == PackageInstaller.STATUS_SUCCESS) return

            if (inForeground() && status == PackageInstaller.STATUS_PENDING_USER_ACTION) {
                promptUser(intent, context)
            } else {
                postStatus(status, packageName, extra, context)
                notifyUser(context, packageName!!, status)
            }
        }
    }
}
```

# Shell and Third-party Apps

# Root

## Running elevated shell commands

- Using libraries such as Magisk's libsu, elevated shell commands can be used to install apps
- Allows to install/update apps without user intervention
- Installation source can also be changed without any extra privileges

```
Check `RootInstaller.kt` on gitlab.com/AuroraOSS/AuroraStore
```

# Shizuku

**Let your app use system APIs directly**

- Shizuku allows apps to use system APIs
- User sets up Shizuku with root or shell and Shizuku shares its permission with other apps
- Alternatively, Sui, which is a Magisk module, can also be used to achieve similar behaviour

```
Check `ShizukuInstaller.kt` on gitlab.com/AuroraOSS/AuroraStore
```

# Uninstalling Apps

# Uninstalling Apps

- Apps can also be uninstalled using Intent or PackageInstaller APIs
- Elevated shell commands also allows uninstalling apps
- Simplest and recommended way is using Intent
- Intent based uninstallation prompts user confirmation before uninstall

```kotlin
fun uninstall(context: Context, packageName: String) {
    val intent = Intent().apply {
        data = Uri.fromParts("package", packageName, null)
        addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        if (isPAndAbove()) {
            action = Intent.ACTION_DELETE
        } else {
            @Suppress("DEPRECATION")
            action = Intent.ACTION_UNINSTALL_PACKAGE
            putExtra(Intent.EXTRA_RETURN_RESULT, true)
        }
    }
    context.startActivity(intent)
}
```

# Thank You!