

Runtime Resource Overlay

1. Change the value of an app's resources at runtime bookmark_border A runtime resource overlay (RRO) is a package that changes the resource values of a target backage at runtime. For example, an app installed on the system image might change its behavior based upon the value of a resource. Rather than hardcoding the resource value at build time, an RRO installed on a different partition can change the values of the app's resources at runtime. RROs can be enabled or disabled. You programmatically set the enable/disable state to toggle an RRO's ability to change resource values. RROs are disabled by default (however, static RROs are enabled by default). Note: An RRO can only be used to change the values for an existing resource. Overlay. 2. So it works for all apps? 3. Troubleshoot runtime resource overlays bookmark_border Use this content to troubleshoot why a runtime resource overlay (RRO) might not be functioning as ntended in an Android Automotive implementation. Related documentation To learn more about RROs in Android, see Change the value of an app's resources at runtime. Be sure to maintain a constant vigilance on logical output, which can provide good information about what is happening throughout the process. Step 1: Listers of the RROs: Run the following command: adb shell cmd overlay listuser current Output like this appears: com.android.systemui []	ı can Dy 2
com.android.theme.icon_pack.rounded.systemui [] com.android.theme.icon_pack.filled.systemui [] com.android.theme.icon_pack.	11
4. Types of resource overlays.	18
5. Actor or Config-Signature RROs (Android 12+)explain this	21
5. Give working examples of both.	23
7. yes android.bp	26
3. yes	28





1. Change the value of an app's resources at runtime

bookmark_border

A runtime resource overlay (RRO) is a package that changes the resource values of a target package at runtime. For example, an app installed on the system image might change its behavior based upon the value of a resource. Rather than hardcoding the resource value at build time, an RRO installed on a different partition can change the values of the app's resources at runtime.

RROs can be enabled or disabled. You can programmatically set the enable/disable state to toggle an RRO's ability to change resource values. RROs are disabled by default (however, static RROs are enabled by default).

Note: An RRO can only be used to change the values for an existing resource.

Overlay resources

Overlays work by mapping resources defined in the overlay package to resources defined in the target package. When an app attempts to resolve the value of a resource in the target package, the value of the overlay resource the target resource is mapped to is returned instead.

Set up the manifest

A package is considered an RRO package if it contains an <overlay> tag as a child of the <manifest> tag.

The value of the required android:targetPackage attribute specifies the name of the package the RRO intends to overlay.

The value of the optional android:targetName attribute specifies the name of the overlayable subset of resources of the target package the RRO intends to overlay. If the target doesn't define an overlayable set of resources, this attribute shouldn't be present.

The following code shows an example overlay AndroidManifest.xml.

</manifest>

Overlays can't overlay code, so they can't have DEX files. In addition, the android:hasCode attribute of the <application> tag in the manifest must be set to false.

Define the resources map

In Android 11 or higher, the recommended mechanism for defining the overlay resources map is to create a file in the res/xml directory of the overlay package, enumerate the target resources that should be overlaid and their replacement values, then set the value of the android:resourcesMap attribute of the <overlay> manifest tag to a reference to the resource mapping file.

The following code shows an example res/xml/overlays.xml file.

package="com.example.overlay">



</manifest>

Note: In Android 10 or lower, resources are overlaid based upon their name. To overlay the resource string/foo in the target package, the overlay must also define the resource string/foo.

Build the package

Android 11 or higher supports a Soong build rule for overlays that prevents Android Asset Packaging Tool 2 (AAPT2) from attempting to dedupe configurations of resources with the same value (--no-resource-deduping) and from removing resources without default configurations (--no-resource-removal). The following code shows an example Android.bp file.

```
runtime_resource_overlay {
  name: "ExampleOverlay",
  sdk_version: "current",
}
```

Resolve resources

If a target resource or overlay resource has multiple configurations defined for the resource being queried, the resources runtime returns the value of the configuration that best matches the configuration of the device configuration. To determine which configuration is the best matching configuration, merge the set of the overlay resource configurations into the set of target resource configurations and then follow the regular resource resolution flow (for details, refer to How Android finds the best-matching resource).

For example, if an overlay defines a value for the drawable-en configuration and the target defines a value for drawable-en-port has a better match so the value of the target configuration drawable-en-port is chosen at runtime. To overlay all drawable-en configurations, the overlay must define a value for each drawable-en configuration the target defines.

Overlays can reference their own resources, with differing behaviors between Android releases.

In Android 11 or higher, each overlay has its own reserved resource ID space that doesn't overlap target resource ID space or other overlay resource ID spaces, so overlays referencing their own resources work as expected.

In Android 10 or lower, overlays and target packages share the same resource ID space, which can cause collisions and unexpected behavior when they attempt to reference their own resources using the @type/name syntax.

Enable/disable overlays

Overlays can manually and programmatically be enabled/disabled.

Manually disable or enable overlays

To manually enable and verify an RRO, run:

adb shell cmd overlay enable --user current com.example.carrro

adb shell cmd overlay list --user current | grep -i com.example com.example.carrro

This enables the RRO for the system user (userId = 0) which owns the SystemUI. This instruction doesn't affect apps started by the foreground user (userId = 10). To enable the RRO for the foreground user, use the parameter --user 10:

adb shell cmd overlay enable --user 10 com.example.carrro

Programmatically enable or disable overlays

Use the OverlayManager API to enable and disable mutable overlays (retrieve the API interface using

Context#getSystemService(Context.OVERLAY_SERVICE)). An overlay can be enabled only by the package it targets or by a package with the android.permission.CHANGE_OVERLAY_PACKAGES permission. When an overlay is enabled or disabled, configuration change events propagate to the target package and target activities relaunch.

Restrict overlayable resources

In Android 10 or higher, the <overlayable> XML tag exposes a set of resources that RROs are allowed to overlay. In the following example res/values/overlayable.xml file, string/foo and integer/bar are resources used for theming the device's appearance; to overlay these resources, an overlay must explicitly target the collection of overlayable resources by name.

```
<!-- The collection of resources for theming the appearance of the device -->
<overlayable name="ThemeResources">
        <policy type="public">
            <item type="string" name="foo/" />
            <item type="integer" name="bar/" />
```



```
</policy>
...
</overlayable>
An APK can define multiple <overlayable> tags, but each tag must have a unique name within the package. For example, it is:
```

OK for two different packages to both define <overlayable name="foo">.

Not OK for a single APK to have two <overlayable name="foo"> blocks.

The following code shows an example of an overlay in the AndroidManifest.xml file.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.my.theme.overlay">
    <application android:hasCode="false" />
    <!-- This overlay will override the ThemeResources resources -->
    <overlay android:targetPackage="android" android:targetName="ThemeResources">
</manifest>
```

Must specify targetName.

Can overlay only the resources listed within the <overlayable> tag.

Can target only one <overlayable> name.

You can't enable an overlay targeting a package that exposes overlayable resources but doesn't use android:targetName to target a specific <overlayable> tag.

Note: If the target package doesn't have a defined overlayable, the overlay must be preinstalled on the system image or signed with the same signature as the target to overlay the target package's resources.

Restrict policies

Use the <policy> tag to enforce restrictions on overlayable resources. The type attribute specifies which policies an overlay must fulfill to override the included resources. Supported types include the following.

public. Any overlay can override the resource.

system. Any overlay on the system partition can override the resources.

vendor. Any overlay on the vendor partition can override the resources.

product. Any overlay on the product partition can override the resources.

 $oem. \ Any \ overlay \ on \ the \ oem \ partition \ can \ override \ the \ resources.$

odm. Any overlay on the odm partition can override the resources.

signature. Any overlay signed with the same signature as the target APK can override the resources.

actor. Any overlay signed with the same signature as the actor APK can override the resources. The actor is declared in named-actor tag in system config.

config_signature. Any overlay signed with the same signature as the overlay-config apk can override the resources. The overlay-config is declared in overlay-config-signature tag in system config.

The following code shows an example <policy> tag in the res/values/overlayable.xml file.

To specify multiple policies, use vertical bars (|) as separator characters. When multiple policies are specified, an overlay needs to fulfill only one policy to override the resources listed within the <policy> tag.

Configure overlays

Android supports different mechanisms for configuring the mutability, default state, and priority of overlays depending on the Android release version.

Devices running Android 11 or higher can use an OverlayConfig file (config.xml) instead of manifest attributes. Using an overlay file is the



recommended method for overlays.

All devices can use manifest attributes (android:isStatic and android:priority) to configure static RROs.

Note: Only immutable overlays targeting the android package affect resource values retrieved through Resources.getSystem(). Use OverlayConfig

In Android 11 or higher, you can use OverlayConfig to configure the mutability, default state, and priority of overlays. To configure an overlay, create or modify the file located at partition/overlay/config/config.xml, where partition is the partition of the overlay to be configured. To be configured, an overlay must reside in the overlay/directory of the partition in which the overlay is configured. The following code shows an example product/overlay/config/config.xml.

```
<config>
  <merge path="OEM-common-rros-config.xml" />
  <overlay package="com.oem.overlay.device" mutable="false" enabled="true" />
  <overlay package="com.oem.green.theme" enabled="true" />
  </config>"
```

The <overlay> tag requires a package attribute that indicates which overlay package is being configured. The optional enabled attribute controls whether or not the overlay is enabled by default (default is false). The optional mutable attribute controls whether or not the overlay is mutable and can have its enabled state changed programmatically at runtime (default is true). Overlays not listed within a configuration file are mutable and disabled by default.

Overlay precedence

When multiple overlays override the same resources, the order of the overlays is important. An overlay has greater precedence than overlays with configurations preceding its own configuration. The precedence order of overlays in different partitions (from least to greatest precedence) is as follows.

system

vendor

odm

product

system_ext

Note: When the OverlayManagerService enables nonconfigured overlays, the order of overlays is undefined.

Merge file

Using <merge> tags allows for other configuration files to be merged at the specified position into the configuration file. The path attribute of the tag represents the path of the file to merge relative to the directory containing overlay configuration files.

Use manifest attributes/static RROs

In Android 10 or lower, overlay immutability and precedence are configured using the following manifest attributes.

android:isStatic. When the value of this boolean attribute is set to true, the overlay is enabled by default and is immutable, which prevents the overlay from being disabled.

android:priority. The value of this numeric attribute (which affects only static overlays) configures the precedence of the overlay when multiple static overlays target the same resource value. A higher number indicates a higher precedence.

The following code shows an example AndroidManifest.xml.

</manifest>

Changes in Android 11

In Android 11 or higher, if a configuration file is located in partition/overlay/config/config.xml, overlays are configured using that file and android:isStatic and android:priority don't have an effect on overlays located in the partition. Defining an overlay configuration file in any partition enforces the overlay partition precedence.

In addition, Android 11 or higher removes the ability to use static overlays to affect the values of resources read during package installation. For the common use case of using static overlays to change the value of booleans that configure component enabled state, use the <component-override> SystemConfig tag (new in Android 11).

Debug overlays

To manually enable, disable, and dump overlays, use the following overlay manager shell command.

adb shell cmd overlay

Using enable without specifying a user affects the current user, that is, the system user (userId = 0), which owns the System UI. This doesn't affect the foreground user (userId = 10), which owns the apps. To enable the RRO for the foreground user, use the parameter --user 10:

adb shell cmd overlay enable --user 10 com.example.carrro

OverlayManagerService uses idmap2 to map resource IDs in the target package to resource IDs in the overlay package. The generated ID mappings are stored in /data/resource-cache/. If your overlay isn't working correctly, find the corresponding idmap file for your overlay in /data/resource-cache/, then run the following command.

adb shell idmap2 dump --idmap-path [file]

This command prints the mapping of resources as shown below.

[target res id] - > [overlay res id] [resource name] 0x01040151 -> 0x01050001 string/config_dozeComponent

0x01040152 -> 0x01050002 string/config_dozeDoubleTapSensorType

0x01040153 -> 0x01050003 string/config_dozeLongPressSensorType

Here's a simple explanation of how Runtime Resource Overlay (RRO) works in Android, along with a summary and practical example:

What is RRO (Runtime Resource Overlay)?

RRO allows you to change the value of resources in an app at runtime, without changing the app itself.

Good for:

- · Customizing system themes (colors, strings, icons, etc.)
- · Changing feature behavior without rebuilding APKs
- · OEM customizations across partitions

How It Works

RRO is a special APK (overlay APK) that:

- 1. Contains no code (just resource files like strings, colors).
- 2. Maps its own resources to those of the target app.
- 3. Gets enabled or disabled dynamically (using ADB or code).
- 4. At runtime, Android replaces target app's resources with overlay ones.
 - Think of it as a runtime skin or patch that rewires app resources.

Setup Flow Summary

1. Create an overlay APK

With a manifest like:

```
xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"</pre>
   package="com.example.overlay">
<application android:hasCode="false" />
   android:resourcesMap="@xml/overlays" />
</manifest>
```

2. Define res/xml/overlays.xml

Maps overlay resources to target app resources:

3. Overlayable Resources in the Target App

In the target app (e.g., SystemUI), define:

4. Build with Soong (Android.bp)

```
runtime_resource_overlay {
    name: "MyCustomOverlay",
    sdk_version: "current",
}
```

Enabling/Disabling Overlay


```
adb shell cmd overlay enable --user θ com.example.overlay adb shell cmd overlay disable --user θ com.example.overlay
```

Use --user 10 if targeting foreground user.

Programmatically:

Use OverlayManager:

```
java

OverlayManager om = (OverlayManager) getSystemService(Context.OVERLAY_SERVICE);
om.setEnabled("com.example.overlay", true, UserHandle.myUserId());
```

🄰 Access Rules

Feature	Requirement
Overlay targeting app	Must define android: targetPackage in manifest
Overlaying specific resources	Target app must declare overlayable block
Static overlays (Android ≤10)	Set android:isStatic="true" in manifest
Dynamic overlays (Android 11+)	Use OverlayConfig in /overlay/config/config.xml

Nebugging Overlays

1. Check applied overlays:



bash

adb shell cmd overlay list

2. Inspect resource mapping:

bash

adb shell idmap2 dump --idmap-path /data/resource-cache/[overlay_target]@[overlay].idmap

- 3. Logs or rebuild overlay if not working:
 - Confirm resource name is same
- Ensure target declared overlayable
- Reinstall overlay and enable again

Example Use Case

You want to change a string in SystemUI from "Battery Saver" to "Power Save Mode":

Steps:

1. Create an overlay with:

```
xml
<item target="string/config_battery_saver_label" value="Power Save Mode" />
```

- 2. Target SystemUI and use correct overlayable name.
- 3. Build and push the overlay APK:

```
bash

adb install com.overlay.powersave.apk
adb shell cmd overlay enable --user θ com.overlay.powersave
```

Now, SystemUI will show "Power Save Mode" instead of "Battery Saver".

Would you like a step-by-step working example project with a sample overlay and target app?



2. So it works for all apps?

Great question — no, RRO (Runtime Resource Overlay) does not work for all apps by default. There are important restrictions and requirements.

When does RRO work for an app?

- ✓ It works if:
- 1. The target app is installed on the system partition, or:
- 2. The overlay APK is signed with the same certificate as the target app, or:
- 3. The target app explicitly declares <overlayable> resources (Android 10+), and the overlay correctly uses android:targetName.

X It does NOT work if:

- 1. The target app does not declare overlayable resources (and is not signed the same or preinstalled).
- 2. The overlay tries to override resources that aren't declared as overlayable.
- 3. The app is installed via Play Store and is not designed to support overlays.

Scenarios where RRO is commonly used:

Scenario	Works?	Notes
System apps like SystemUI, Settings, Launcher	V	Designed with overlay support
AOSP or OEM preinstalled apps	V	Often support overlays
Third-party Play Store app	X	Not supported unless signed or built with overlayable support
Your own app (with proper setup)	V	If you define <overlayable></overlayable> in it or sign overlay with same key

Android 10+ Security: overlayable tag

Starting Android 10 (Q), overlaying is blocked unless the target explicitly opts in by using the <overlayable> tag in res/values/overlayable.xml.

This is to **protect app developers** from unexpected behavior or UI tampering.

Now to make your own app support RRO

1. Add overlayable.xml:

2. Reference it in your overlay:

Now your app can be safely overlaid by any RRO that targets this resource set.

Would you like a **custom app + overlay project setup** so you can try it yourself on any device?





3. Troubleshoot runtime resource overlays

bookmark_border

Use this content to troubleshoot why a runtime resource overlay (RRO) might not be functioning as intended in an Android Automotive implementation.

Related documentation

To learn more about RROs in Android, see Change the value of an app's resources at runtime. Be sure to maintain a constant vigilance on logcat output, which can provide good information about what is happening throughout the process.

Step 1: List the RROs To list the RROs:

Run the following command:

adb shell cmd overlay list --user current Output like this appears:

com.android.systemui

- [] com.android.theme.icon_pack.rounded.systemui
- [] com.android.theme.icon_pack.filled.systemui
- [] com.android.theme.icon_pack.circular.systemui

com.android.permissioncontroller

--- com.android.permissioncontroller.googlecarui.rro

Verify that your RRO appears in the list. The following indicators denote RRO status:

Indicator RRO status

- [] Installed and ready to be activated.
- [X] Installed and activated.
- --- Installed but contains errors.

If your RRO isn't listed under the package name of the target you want to overlay, your RRO isn't installed.

Step 2: Enable and disable an RRO

If an RRO is installed:

Use the following command to enable (or disable) the RRO:

adb shell cmd overlay [enable/disable] --user current [your RRO package name]

Note: Using Enable without specifying a user affects the current, that is the system user (userId = 0), not the foreground user (userId = 10). To enable an RRO for the foreground user, use the parameter --user 10. The system user owns the System UI, but doesn't own apps started by the foreground user.

Note: Static RROs are enabled by default at boot time. Dynamic RROs aren't enabled by default at boot. There are some exceptions to these default behaviors. For details, see Change the value of an app's resources at runtime.

Step 3: Confirm RRO is installed

To confirm an RRO is installed on a device or troubleshoot why the RRO wasn't enabled:

Run the following command:

adb shell cmd overlay dump [your RRO package name]

Output such as the following appears:

com.android.car.rotaryplayground.googlecarui.rro:0 {

mPackageName.....: com.android.car.rotaryplayground.googlecarui.rro

mUserId.....: 0

mTargetPackageName.....: com.android.car.rotaryplayground

mTargetOverlayableName.: car-ui-lib

mBaseCodePath.......:/product/overlay/googlecarui-com-android-car-rotaryplayground/googlecarui-com-android-car-rotaryplayground.apk



```
mState..... STATE_MISSING_TARGET
 mIsEnabled..... true
 mIsMutable....: true
 mPriority.....: 10
 mCategory.....: BypassIdMapV1
com.android.car.rotaryplayground.googlecarui.rro:10 {
 mPackageName.....: com.android.car.rotaryplayground.googlecarui.rro
 mUserId.....: 10
 mTargetPackageName.....: com.android.car.rotaryplayground
 mTargetOverlayableName.: car-ui-lib
 mBaseCodePath.......:/product/overlay/googlecarui-com-android-car-rotaryplayground/googlecarui-com-android-car-rotaryplayground.apk
 mState.....: STATE_MISSING_TARGET
 mIsEnabled....: true
 mIsMutable....: true
 mPriority.....: 10
mCategory.....: BypassIdMapV1
Identify the user (or users) who installed the RRO. In the previous example, the RRO is available for user 0 and user10 (see the values for mUserId
in the top code block).
To enable (or disable) the RRO for a user, go to Step 2.
To check the value of mState:
STATE_ENABLED and STATE_ENABLED_IMMUTABLE. The RRO is enabled and applied to your target.
STATE_MISSING_TARGET. Your target isn't installed.
STATE_NO_IDMAP. Something is wrong with the way the AndroidManifest.xml, overlays.xml, or overlayable.xml files are set up. Use adb logcat to
run the logs and search for the keyword idmap to identify the error. See steps 4 & 5.
STATE_UNKNOWN. Something is wrong with OverlayManagerService.
Step 4: Check AndroidManifest.xml
To verify AndroidManifest.xml:
Check targetName and targetPackage.
android:targetName should have the same value of the overlayable group defined in the target app. This is required only when targeting an
overlay.
android:targetPackage is always required and should contain the package name of the target app.
Check that your RRO is static (or not). Static RROs are enabled by default at boot time. Dynamic RROs aren't enabled by default at boot. Additional
ways of enabling dynamic RROs are provided in Change the value of an app's resources at runtime.
Check the priority of your static RRO (dynamic RRO priority is always set to Integer.MAX_VALUE and the order in which they're applied is based on
when they're enabled).
Multiple RROs can apply to the same target. RROs with a higher priority are applied last. On a scale of 0 to 10, 10 is highest and 0 is lowest.
Step 5: Check overlays.xml
Note: This check applies only to Android 11 and higher.
Check overlays.xml to confirm all resources you intend to overlay are defined in this file. For example, consider the followingoverlays.xml:
<overlay>
  <item target="string/app_name" value="@string/overlaid_app_name" />
</overlay>
You must ensure that a:
string resource with the name app_name exists in the target app.
string resource with the name overlaid_app_name exists in your RRO.
If your target has an overlayable.xml file, make sure app_name is contained in that file. Make sure you use the correct targetName in your
```

AndroidManifest.xml file (Step 4).



Note: If you overlay a layout file, make sure all the IDs and app namespace attributes are included in both overlays.xml and overlayable.xml. For example: <overlay> <item target="layout/car_ui_base_layout_toolbar" value="@layout/car_ui_base_layout_toolbar" /> <item target="id/car_ui_toolbar_background" value="@id/car_ui_toolbar_background" /> <item target="attr/layout_constraintTop_toBottomOf" value="@attr/layout_constraintTop_toBottomOf" /> </overlay> Step 6: Dump the idmap At this stage, all issues with your RRO should be resolved. Next, dump the idmap of your RRO to learn how a resource is being resolved and why it's resolving to a different value than you expect. To find the path to the idmap on your device: adb shell Is data/resource-cache To dump the contents of that file: adb root adb shell idmap2 dump --idmap-path [path to your RRO idmap file] Example: adb shell idmap2 dump --idmap-path data/resourcecache/system@app@CarUiPortraitLauncherReferenceRRO@CarUiPortraitLauncherReferenceRRO.apk@idmap.cache/system@app@CarUiPortraitLauncherReferenceRRO.apk@idmap.cache/system@app@CarUiPortraitLauncherReferenceRRO.apk@idmap.cache/system@app@CarUiPortraitLauncherReferenceRRO.apk@idmap.cache/system@app@CarUiPortraitLauncherReferenceRRO.apk@idmap.cache/system@app@CarUiPortraitLauncherReferenceRRO.apk@idmap.cache/system@app.cache/system.caThe output appears similar to the following. The output displays which ID from your RRO is mapped to which ID in your target as well as the name of the overlaid resource. target apk path:/system/priv-app/CarMediaApp/CarMediaApp.apk overlay apk path:/product/overlay/googlecarui-com-android-car-media/googlecarui-com-android-car-media.apk 0x7f040008 -> 0x7f010000 bool/car ui toolbar logo fills nav icon space 0x7f040009 -> 0x7f010001 bool/car_ui_toolbar_nav_icon_reserve_space 0x7f04000b -> 0x7f010002 bool/car_ui_toolbar_tab_flexible_layout 0x7f04000c -> 0x7f010003 bool/car_ui_toolbar_tabs_on_second_row 0x7f09006c -> 0x7f020000 id/car ui base layout content container 0x7f090073 -> 0x7f020001 id/car ui recycler view 0x7f090074 -> 0x7f020002 id/car_ui_scroll_bar 0x7f090075 -> 0x7f020003 id/car_ui_scrollbar_page_down 0x7f090076 -> 0x7f020004 id/car_ui_scrollbar_page_up 0x7f090077 -> 0x7f020005 id/car_ui_scrollbar_thumb 0x7f090078 -> 0x7f020006 id/car_ui_scrollbar_track 0x7f09007a -> 0x7f020007 id/car_ui_toolbar_background 0x7f09007e -> 0x7f020008 id/car_ui_toolbar_logo 0x7f090084 -> 0x7f020009 id/car_ui_toolbar_menu_items_container 0x7f090085 -> 0x7f02000a id/car_ui_toolbar_nav_icon 0x7f090086 -> 0x7f02000b id/car_ui_toolbar_nav_icon_container 0x7f090087 -> 0x7f02000c id/car_ui_toolbar_progress_bar 0x7f090089 -> 0x7f02000d id/car_ui_toolbar_row_separator_guideline 0x7f09008d -> 0x7f02000e id/car_ui_toolbar_search_view_container 0x7f09008f -> 0x7f02000f id/car_ui_toolbar_subtitle 0x7f090092 -> 0x7f020010 id/car_ui_toolbar_tabs 0x7f090093 -> 0x7f020011 id/car_ui_toolbar_title 0x7f090094 -> 0x7f020012 id/car ui toolbar title container 0x7f090095 -> 0x7f020013 id/car ui toolbar title logo 0x7f090096 -> 0x7f020014 id/car_ui_toolbar_title_logo_container 0x7f0c0024 -> 0x7f030000 layout/car_ui_base_layout_toolbar 0x7f0c0035 -> 0x7f030001 layout/car_ui_recycler_view 0x7f0c0038 -> 0x7f030002 layout/car_ui_toolbar

Use the following command to look up a specific resource to see how they're mapped:

0x7f0c003f -> 0x7f030003 layout/car_ui_toolbar_two_row



adb shell cmd overlay lookup --verbose --user 10 com.android.car.ui.paintbooth com.android.car.ui.paintbooth:color/widget_background Note: --user 10 is supplied to ensure that the mapping has been applied for the foreground user (userId = 10) which owns apps started by the

The output is the final value of the resource:

#ff7986cb

You can also dump your layout files from your apks to view the resolved IDs to match the above output:

aapt2 dump xmltree \$OUT/system/priv-app/sharedlibraryclient/sharedlibraryclient.apk --file res/layout/activity_main.xml Output such as the following is returned:

N: android=http://schemas.android.com/apk/res/android (line=2)

N: app=http://schemas.android.com/apk/res-auto (line=2)

N: lib=http://schemas.android.com/apk/com.android.car.ui.sharedlibrary.test (line=2)

E: androidx.constraintlayout.widget.ConstraintLayout (line=2)

A: http://schemas.android.com/apk/res/android:layout_width(0x010100f4)=-1

A: http://schemas.android.com/apk/res/android:layout_height(0x010100f5)=-1

E: TextView (line=19)

A: http://schemas.android.com/apk/res/android:layout_width(0x010100f4)=-2

A: http://schemas.android.com/apk/res/android:layout_height(0x010100f5)=-2

A: http://schemas.android.com/apk/res/android:text(0x0101014f)=@0x020f0000

A: http://schemas.android.com/apk/res-auto:layout_constraintBottom_toBottomOf(0x7f0200fb)=0

A: http://schemas.android.com/apk/res-auto:layout_constraintLeft_toLeftOf(0x7f02010e)=0

A: http://schemas.android.com/apk/res-auto:layout_constraintRight_toRightOf(0x7f020112)=0

A: http://schemas.android.com/apk/res-auto:layout_constraintTop_toTopOf(0x7f020118)=0

E: com.android.car.ui.sharedlibrary.test.MyRecyclerView (line=28)

A: http://schemas.android.com/apk/res/android:layout_width(0x010100f4)=-2

A: http://schemas.android.com/apk/res/android:layout_height(0x010100f5)=-2

A: http://schemas.android.com/apk/com.android.car.ui.sharedlibrary.test:implClass="HelloWorld!" (Raw: "HelloWorld!")

E: com.android.car.ui.sharedlibraryclient.CustomView (line=34)

A: http://schemas.android.com/apk/res/android:layout_width(0x010100f4)=-2

A: http://schemas.android.com/apk/res/android:layout height(0x010100f5)=-2

A: http://schemas.android.com/apk/res-auto:implClass2(0x7f0200e8)="HelloWorld!!" (Raw: "HelloWorld!!")

Failures to resolve mappings are indicated by idmap2 through three question marks (???).

In this example, the RRO has a hard-coded color value overlaid on top of a public resource:

\$ idmap2 dump --idmap-path [file]

0x00010402 -> color 0xff00ff00 (???)

In this example, the RRO overlays a public resource by pointing it to an internal (private) color resource inside the same shared library:

\$ idmap2 dump --idmap-path [file]

0x00010402 -> 0x7f010000 (??? -> color/item_background_new)

To troubleshoot Runtime Resource Overlays (RROs) in Android (especially Automotive), follow this step-by-step cheat sheet, with explanations for each common failure and how to fix them:

Step-by-Step RRO Troubleshooting Guide



Step 1: Check if the RRO is listed

Run:

adb shell cmd overlay list --user current

Check the status:

Symbol	Meaning
[]	Installed but not enabled
[X]	Installed and enabled
	Installed but contains an error

If it's missing from the list:

- APK is not installed
- Target package not found
- Signature/permissions issues

Step 2: Try enabling the RRO

bash

adb shell cmd overlay enable --user 10 com.example.overlay

- --user 10 is foreground user (apps are launched under this user).
- --user 0 is system user (SystemUI, etc).
- 💡 Tip: For overlays affecting SystemUI, use user 0. For regular apps, use user 10.

Step 3: Dump overlay info

bash

adb shell cmd overlay dump com.example.overlay

Look at mState:

State	Meaning
STATE_ENABLED	Overlay is active
STATE_MISSING_TARGET	The target app is not installed
STATE_NO_IDMAP	Resource mapping failed (invalid resource names or mapping issues)
STATE_UNKNOWN	Unknown error in OverlayManagerService

Step 4: Validate AndroidManifest.xml

Ensure:

- android:targetPackage="com.example.target"
- android:targetName="OverlayableGroupName" (if target uses <overlayable>)
- Common mistakes:
 - targetName doesn't match the name in overlayable.xml
 - Missing targetPackage
 - Static overlays missing android:isStatic="true"
- For static overlays (enabled at boot):

xml

<overlay android:targetPackage="..." android:isStatic="true" android:priority="10" />

Step 5: Validate overlays.xml (Android 11+)



```
xml
<overlay>
    <item target="string/app_name" value="@string/custom_app_name"/>
</overlay>
```

Check:

- target exists in target app
- value exists in overlay
- If target uses <overlayable>, confirm the resource is listed there

X If a resource is **not public or not declared**, overlaying it will **fail silently**

Step 6: Dump idmap mappings

Find mapping file:

```
bash
adb shell ls /data/resource-cache/
```

Dump it:

bash

adb shell idmap2 dump --idmap-path /data/resource-cache/[idmap_filename]

Look for lines like:

rust

0x7f040008 -> 0x7f010000 bool/my_bool

If output is:

```
ruby
0x00010402 -> color 0xff00ff00 (???)
```

...then:

- Value exists, but original resource name is unknown (not public?)
- ??? = unresolved mapping

③ Step 7: Lookup specific resource

bash

adb shell cmd overlay lookup --verbose --user 10 com.target com.target:color/my_color

This shows the **final value** that will be returned at runtime.

✓ Common Errors & Fixes

Issue	Root Cause	Fix
RRO not visible	APK not installed, or incorrect partition	Install correctly, check partition rules
STATE_MISSING_TARGET	Target app isn't installed	Install the target app
STATE_NO_IDMAP	Resource mismatch or bad mapping	Fix resource names, overlays.xml
Overlay not applying	Not enabled for correct user	Useuser 10
Overlays silently ignored	Resource not public or not in <overlayable></overlayable>	Add to <overlayable> with proper <policy></policy></overlayable>
??? in idmap2 dump	ID could not be resolved	Resource doesn't exist or is private



⋘ Debug Tips

- Use adb logcat | grep idmap to see overlay manager logs
- Reboot after pushing static overlays
- Confirm resource types match (don't map a string to a bool, etc.)
- For layout, all id and attr resources must match in overlay and target

Would you like a working **RRO + target app template** to test all this yourself?



4. Types of resource overlays

In Android, resource overlays come in multiple types depending on how they're applied, when they're applied, and who can control them. Here's a breakdown of all the important types of Runtime Resource Overlays (RROs):



1. Static RROs

- V Enabled at boot time
- X Cannot be enabled/disabled at runtime
- 🄰 Immutable after system startup

✓ Use Case:

- OEM overlays preinstalled in system/vendor/product partition
- Theming, branding, pre-configured UI changes

Configuration:

In AndroidManifest.xml:

```
xml
<overlay
   android:targetPackage="com.example.target"
   android:isStatic="true"
   android:priority="10"/>
```

2. Dynamic RROs

- X Not enabled by default
- **V** Can be **enabled/disabled at runtime** using:

```
adb shell cmd overlay enable --user 10 com.example.overlay
```

• Mutable (unless overridden by OverlayConfig)

✓ Use Case:

- System theming (dark mode, accent colors)
- Development/testing overlays
- Runtime UI tuning

3. Mutable vs Immutable RROs

Туре	Can enable/disable at runtime?	
Mutable	✓ Yes	
Immutable	XNo	

Controlled via:

- Manifest (android:isStatic)
- Or, Android 11+ → config.xml:

```
xml
<overlay package="com.example.overlay" mutable="false" />
```

4. Preinstalled Overlays

- Stored under /system, /vendor, /product, /system_ext, /odm, /oem
- Can be either static or dynamic
- Trusted overlays with elevated permissions

5. Overlayable RROs (Android 10+)

- Only allowed to overlay declared <overlayable> resources
- Must use android:targetName

Example in target app:

Overlay manifest:

6. Partition-Based Overlays (by Trust Domain)

Partition	Who Can Overlay	Priority Level
system	AOSP	Low
vendor	Vendor/OEM	Medium
product	OEM customization	Higher
oem	Manufacturer	Higher
odm	Device maker	Higher
system_ext	Shared system	Highest

Used to resolve conflicting overlays when multiple overlays target the same resource.

7. Signature RROs

- Overlays signed with the same signature as the target APK
- Can override non-public resources, even without <overlayable>

✓ Use Case:

- Development overlays
- Internal app configuration by same developer/OEM

8. Actor or Config-Signature RROs (Android 12+)

• Allow third-party signed overlays if registered via system config:



Summary Table

Туре	Boot Enabled	Runtime Toggle	Overlayable Required	Example Use Case
Static RRO	✓ Yes	X No	Optional (Android <10)	Preinstalled OEM overlays
Dynamic RRO	X No	✓ Yes	✓ Yes (Android 10+)	User themes, dev overlays
Signature RRO	X No	✓ Yes	X No	Trusted internal config
Overlayable RRO	X No	✓ Yes	✓ Yes (mandatory)	Resource-safe public overlays
Partition-based	Varies	Varies	✓ Yes or signed	System/Vendor layering

Would you like help creating each type for testing or exploring which types your device currently supports?



5. L

Actor or Config-Signature RROs (Android 12+) explain this

Great! Let's break down "Actor" and "Config-Signature" RROs, which were introduced in Android 12+ to allow trusted third-party overlays while still maintaining resource security and integrity.

Problem Before Android 12

Before Android 12:

- · Only overlays preinstalled, signed with the same key, or targeting public/overlayable resources were allowed.
- You couldn't overlay restricted resources unless you:
 - Were signed with the same key
 - Or the resource was marked public in <overlayable>

So if a customization app wanted to change a resource in another app it didn't sign, it was blocked.

☑ Solution in Android 12+: Actor and Config-Signature RROs

These allow secure delegation of resource overlaying to other packages without requiring same-signature.

♦ 1. Actor RROs

Concept:

An actor is a trusted package that is allowed to apply overlays on behalf of a system-defined role, like theming or OEM customization.

How it works:

1. System defines a trusted actor in a system config XML:

2. The target app (whose resources are to be overlaid) defines this in res/values/overlayable.xml:

- 3. The **overlay APK** is signed by com.oem.customizationapp.
- Now, even though this overlay is **not signed with the same key**, it is **allowed** to override the listed resources because the system **recognizes its** signature via the actor trust.

Security:

- Only actors listed in system config XML are trusted
- Actor overlays can only touch explicitly whitelisted resources



🔷 2. Config-Signature RROs

Concept:

This is used to authorize overlays signed with the same signature as a system-defined configuration app.

This app is called the overlay-config APK.

How it works:

1. The system declares the overlay-config package in its config XML:

```
<overlay-config-signature package="com.oem.configprovider" />
```

2. The target app defines:

```
xml
</policy>
</overlayable>
```

- 3. The overlay APK must be signed with the same key as com.oem.configprovider.
- 🔽 Now, the overlay can modify the whitelisted resources even though it's not signed with the target app.

Security:

- config_signature allows dynamic and fine-grained control
- Signature match with config-provider package is required
- Ensures only system-trusted configuration apps can alter behavior

Summary Table

Туре	Trust Based On	Allows Overlays From	Must Whitelist Resources?	Android Version
actor	Named actor in system config	Actor package	✓ Yes	Android 12+
config_signature	Signature of config app	Same-signature APKs	✓ Yes	Android 12+

Use Cases

- OEM theming apps that are not signed with android
- Configuration overlays provided by separate vendor modules
- Car UI or Android TV customizations managed by privileged but separately signed apps

Would you like to see a full working example of an actor RRO setup?



6. Give working examples of both

Sure! Here's a full working example of both:

Example 1: Actor RRO

1. System Config (AOSP /etc/sysconfig/oem_actors.xml)

★ This declares the actor oem_theming and binds it to the package com.oem.themeengine.

2. Target App (com.android.settings) overlayable.xml

Located in com.android.settings:

★ This says: "I allow overlays from packages trusted as oem_theming to change toolbar_title."

3. Overlay APK (com.oem.themeengine.overlay)

Manifest:

Resource override:

★ This replaces com.android.settings:string/toolbar_title.

🗹 As long as the overlay APK is signed with the same signature as com.oem.themeengine, this overlay will be allowed.

Example 2: Config-Signature RRO

Goal: Allow an overlay signed with a config-provider APK to override resources in com.android.launcher.

1. System Config

In overlay-config.xml (AOSP device config):

```
<config>
    <overlay-config-signature package="com.oem.configprovider" />
</config>
```

🖈 This tells the system that any overlay signed with the same key as com.oem.configprovider can access resources tagged with config_signature.

2. Target App (com.android.launcher) overlayable.xml

🖈 Allows overlays signed like com.oem.configprovider to change this boolean.

3. Overlay APK (com.oem.launcher.overlay)

Manifest:

Resource override:

This turns on a hidden feature in the launcher.

✓ The overlay works only if it's signed with the same key as com.oem.configprovider.

Testing the Overlays

After pushing the overlays:

```
adb shell cmd overlay enable --user 10 com.oem.themeengine.overlay adb shell cmd overlay enable --user 10 com.oem.launcher.overlay
```

Then verify:

```
adb shell cmd overlay list --user 10
```



Check logs:

bash

adb logcat | grep idmap

Would you like me to generate the full Android.mk or Android.bp files for these examples too?



7. yes android.bp

Here are full working Android.bp examples for both Actor RRO and Config-Signature RRO overlay packages.

1. Actor RRO (com.oem.themeengine.overlay)

File structure:

```
device/oem/overlay/ThemeEngineOverlay/
— Android.bp
— AndroidManifest.xml
— res/
— values/
— strings.xml
```

💤 Android.bp:

```
runtime_resource_overlay {
   name: "ThemeEngineOverlay",
   package_name: "com.oem.themeengine.overlay",
   sdk_version: "current",
   product_specific: true, // or vendor: true / system_ext: true depending on partition
}
```

AndroidManifest.xml:

I res/values/strings.xml:

```
<ml
<resources>
     <string name="toolbar_title">OEM Branded Settings</string>
</resources>
```

2. Config-Signature RRO (com.oem.launcher.overlay)

File structure:

```
device/oem/overlay/LauncherOverlay/

— Android.bp
— AndroidManifest.xml
— res/
— values/
— bools.xml
```

🚅 Android.bp:



```
runtime_resource_overlay {
   name: "LauncherConfigOverlay",
   package_name: "com.oem.launcher.overlay",
   sdk_version: "current",
   product_specific: true, // or vendor: true / system_ext: true
}
```

AndroidManifest.xml:

res/values/bools.xml:

৵ Where to place these overlays

- If your overlay is product-specific, place it under: device/oem/overlay/product/
- If vendor-specific, use: device/oem/overlay/vendor/
- If system_ext, use: device/oem/overlay/system_ext/

🖈 Build integration

In your device's device.mk file, add:

```
PRODUCT_PACKAGES += \
    ThemeEngineOverlay \
    LauncherConfigOverlay
```

Let me know if you want examples for res/xml/overlays.xml, or integration of overlay-config.xml in system config!



8. yes

Perfect! Here's how to add the res/xml/overlays.xml file (required in Android 11+) and how to configure overlays in overlay/config/config.xml using OverlayConfig.

1. Add res/xml/overlays.xml to Actor RRO or Config-Signature RRO

This file maps which resources in the target package will be replaced by what value from the overlay.

Path:

For example, in ThemeEngineOverlay/:

```
pgsql
ThemeEngineOverlay/
    res/
            overlays.xml
```

III res/xml/overlays.xml:

```
xml
<overlay xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Replace string in target package --> <item target="string/toolbar_title" value="@string/toolbar_title" />
</overlay>
```

This maps com.android.settings:string/toolbar_title → com.oem.themeengine.overlay:string/toolbar_title

Update AndroidManifest.xml to reference this file:

```
xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"</pre>
      package="com.oem.themeengine.overlay">
<application android:hasCode="false" />
      <overlay
            android:targetPackage="com.android.settings"
android:targetName="ThemeResources"
android:resourcesMap="@xml/overlays" />
</manifest>
```

Same applies for LauncherConfigOverlay—just use the appropriate bool or other resource names.

2. Overlay Configuration: overlay/config/config.xml

On Android 11+, overlay mutability, default enable state, and priority are configured in this file instead of the manifest.

Path on device:

```
arduino
<partition>/overlay/config/config.xml
```

For example:

- /product/overlay/config/config.xml
- /system_ext/overlay/config/config.xml

Ensure your overlay APK is under the same partition's overlay/ directory.

Example config.xml:

Summary

File	Purpose
res/xml/overlays.xml	Explicitly map target → overlayed resources (required on Android 11+)
android:resourcesMap	Tells system to load res/xml/overlays.xml from overlay APK
overlay/config/config.xml	Sets mutability, enabled state, and priority for overlays

Now your overlays will:

- Use mapped values (overlays.xml)
- Follow strict security policies (overlayable.xml)
- Enable cleanly at boot (config.xml)

Would you like the overlayable.xml file from the target app example also?