Complete Guide to Using MT Manager for Android Modding By CKS

August 2025  A Comprehensive Tutorial for Beginners

and Intermediate Users

Published by CKS

Contents

1 Keywords **4**

2 Required Apps and Tools for Modding **4**
2.1 MT Manager 4
2.2 AntiSplit G2 5
2.3 BotX Atcher 5
2.4 NP Manager 5
2.5 Current Activity 5
2.6 Additional Tools 5

3 Introduction to MT Manager **5**
3.1 What is MT Manager? 6
3.2 Key Features 6
3.3 Use Cases 7
3.4 Why Choose MT Manager? 7

4 Installation Guide **7**
4.1 Where to Download MT Manager Safely 7
4.2 Installing on Rooted vs Non-Rooted Devices 8
4.3 Permissions Required and Initial Setup 8

5 Navigating the Interface **8**
5.1 File Explorer Layout 9
5.2 Dual-Pane Mode 9
5.3 Accessing Internal and External Storage 9

6 APK Editing Workflow **9**
6.1 Locating and Opening an APK File 9
6.2 Viewing Contents 10
6.3 Editing Smali Code and XML Files 10
6.4 Replacing Assets 10

7 Recompiling and Signing **10**
7.1 Rebuilding the APK After Edits 11
7.2 Using MT Manager's Auto-Sign Feature 11
7.3 Verifying the Signature 11
7.4 Installing the Modified APK 11

8 Advanced Features **11**
8.1 Hex Editor for Binary Modifications 11
8.2 Text Editor for Scripts and Config Files 11
8.3 ZIP Compression and Extraction 12
8.4 Permissions Manager and Manifest Tweaks 12

9 Working with Split APKs **12**
9.1 Understanding Split APKs 12
9.2 Extracting and Combining Split APKs 12

9.3 Modding Split APKs 12

10 Adding Custom Dialogues **13**
10.1 Understanding Android Dialogues 13
10.2 Modifying Existing Dialogues 13
10.3 Adding New Dialogues 13
10.4 Triggering Custom Dialogues 14

11 Adding Custom Notifications **14**
11.1 Understanding Android Notifications 14
11.2 Modifying Existing Notifications 14
11.3 Adding Interactive Notifications 16
11.4 Creating Notification Channels 16

12 Increasing Coins or Diamonds in Games **17**
12.1 Understanding Game Currency Systems 17
12.2 Method 1: Modifying Shared Preferences 17
12.3 Method 2: Modifying SQLite Databases 17
12.4 Method 3: Modifying JSON Files 18
12.5 Method 4: Smali Code Modification 18
12.6 Method 5: Memory Editing with GameGuardian 19
12.7 Method 6: Patching Purchase Methods 19
12.8 Testing Currency Modifications 19

13 Removing Ads from Apps **19**
13.1 Identifying Ad-Related Code 20
13.2 Modifying Smali to Disable Ads 20
13.3 Removing Ad Permissions 20
13.4 Testing and Validation 20

14 Bypassing Payment Features **20**
14.1 Locating Payment Checks 21
14.2 Modifying Smali to Unlock Features 21
14.3 Patching Billing Libraries 21
14.4 Ethical Considerations and Testing 21

15 Modifying App User Interfaces **21**
15.1 Changing App Icons 22
15.2 Modifying App Names 22
15.3 Customizing Layouts 22
15.4 Changing Themes and Colors 22
15.5 Testing UI Changes 22

16 Legal and Ethical Considerations **23**
16.1 Respecting Developer Rights 23
16.2 Avoiding Piracy and Malware 23
16.3 Safe Distribution Practices 23

17 Troubleshooting and Tips **23**

17.1 Common Errors and How to Fix Them 23
17.2 Backup Strategies 24
17.3 Compatibility Issues 24

18 Resources and Communities **24**
18.1 XDA Developers Forum 24
18.2 Telegram Modding Groups 24
18.3 YouTube Tutorials and GitHub Repositories 24

19 VIP Features and Store **25**

20 Getting Support **25**

1 Keywords

- **MT Manager**: Android file manager and APK editor.
- **APK Modding**: Modifying Android application packages.
- **Smali**: Human-readable Dalvik bytecode format.
- **Decompilation**: Converting APK to editable code and resources.
- **Recompilation**: Rebuilding modified APK.
- **APK Signing**: Adding a digital signature to APKs.
- **Root Access**: Privileged access to Android system files.
- **XML Editing**: Modifying app layouts and configurations.
- **Hex Editor**: Tool for binary file modifications.
- **Android Modding**: Customizing apps and system behavior.
- **Split APKs**: App bundles divided into multiple APK files.
- **Dialogues**: UI elements that prompt user interaction.
- **Notifications**: Alerts displayed in the notification panel.
- **Game Currency Modding**: Altering in-game coins, diamonds, or resources.
- **Memory Editing**: Direct modification of runtime values.
- **AntiSplit G2**: Tool for handling split APKs.
- **BotX Atcher**: Network traffic interception tool.
- **NP Manager**: Advanced APK analysis tool.
- **Current Activity**: Debugging tool for activity tracking.

2 Required Apps and Tools for Modding

To effectively mod Android apps and games, you'll need a collection of specialized tools. This section covers the essential applications every modder should have.

2.1 MT Manager

- **Purpose**: Primary APK editor and file manager
- **Features**: Smali editing, XML modification, APK signing, root access
- **Download**: Official Telegram Channel
- **Why Essential**: Core tool for all APK modifications covered in this guide

2.2 AntiSplit G2

- **Purpose**: Handling split APKs and app bundles
- **Features**: Combine split APKs, extract base APKs, manage app bundles
- **Download**: XDA Thread
- **Why Essential**: Modern apps use split APKs; this tool makes them moddable

2.3 BotX Atcher

- **Purpose**: Network traffic interception and modification
- **Features**: Capture and modify HTTP/HTTPS requests, bypass server checks
- **Download**: GitHub Repository
- **Why Essential**: For bypassing server-side verification in games and apps

2.4 NP Manager

- **Purpose**: Advanced APK analysis and modification
- **Features**: DEX editing, ARSC modification, string obfuscation
- **Download**: GitHub Repository
- **Why Essential**: Alternative to MT Manager with specialized features

2.5 Current Activity

- **Purpose**: Debugging and activity tracking
- **Features**: View current app activities, services, and broadcasts
- **Download**: Google Play
- **Why Essential**: Helps identify app components for targeted modding

2.6 Additional Tools

- **JADX**: DEX to Java decompiler (GitHub)
- **Apktool**: Command-line APK tool (Website)
- **GameGuardian**: Memory editor for runtime modding (Website)
- **Frida**: Dynamic instrumentation toolkit (Website)

3 Introduction to MT Manager

MT Manager is a premier Android application that combines robust file management with advanced APK editing, making it a cornerstone tool for modders, developers, and enthusiasts. It allows users to explore file systems, modify APKs,

Figure 1: Essential modding tools collection

and perform system-level tweaks on rooted devices. This guide provides an exhaustive, step-by-step tutorial tailored for beginners and intermediate users to master MT Manager for Android modding, with practical examples and safety considerations.

3.1 What is MT Manager?

Developed by Lin Jin Bin, MT Manager is a dual-purpose app that serves as a file explorer and an APK editor. It supports decompiling, editing, and recompiling APKs directly on an Android device, eliminating the need for PC-based tools like APKTool. Its intuitive interface and powerful features make it a favorite in the modding community.

3.2 Key Features

- **File Management**: Navigate and manipulate files with a dual-pane interface, supporting copy, move, delete, and rename operations.
- **APK Editing**: Decompile APKs to access Smali code, XML files, and resources for deep customization.
- **Root Access**: Modify system files (e.g., `/system/app`) on rooted devices.
- **Signing Tools**: Automatically sign APKs with test or custom keys for installation.
- **Advanced Tools**: Includes a hex editor, text editor, ZIP manager, and permissions editor.
- **Split APK Support**: Handle app bundles divided into multiple APK files.
- **Dialogue/Notification Editor**: Modify or add custom dialogues and notifications.

- **Game Modding Tools**: Specialized features for modifying game resources and values.

3.3 Use Cases

MT Manager excels in various modding scenarios:

- **App Customization**: Alter functionality, such as bypassing restrictions or adding features.
- **Ad Removal**: Eliminate intrusive ads for a seamless user experience.
- **Premium Unlocking**: Access paid features without subscriptions.
- **UI Modification**: Customize app icons, themes, or layouts.
- **Game Modding**: Add cheats, modify resources, or create custom interfaces.
- **Split APK Handling**: Combine and modify app bundles for installation.
- **Custom Dialogues**: Add personalized prompts and alerts to apps.
- **Currency Modding**: Increase coins, diamonds, or in-game resources.

3.4 Why Choose MT Manager?

Unlike PC-based tools (e.g., Android Studio, APKTool), MT Manager operates natively on Android, offering convenience and speed. Its all-in-one design integrates decompilation, editing, and signing, making it accessible to users with minimal technical expertise. Regular updates ensure compatibility with modern Android versions.

4 Installation Guide

Proper installation ensures MT Manager functions safely and effectively. This section covers sourcing, installing, and configuring the app.

4.1 Where to Download MT Manager Safely

To avoid malware, download MT Manager from verified sources:

- **Official Source**: Developer's Telegram channel (t.me/mtmanager) or official website.
- **Trusted Communities**: XDA Developers (xdaforums.com) or other reputable forums.
- **Warnings**: Avoid third-party app stores or unverified sites. Check the APK's SHA256 checksum if provided (e.g., via `APK Info` in MT Manager).

4.2 Installing on Rooted vs Non-Rooted Devices

- **Non-Rooted Devices**:
  1. Download the MT Manager APK.
  2. Enable `Install from Unknown Sources` (`Settings > Security > Unknown Sources`).
  3. Open the APK and tap `Install`.
  4. Limitation: No access to system files or advanced root features.
- **Rooted Devices**:
  1. Ensure a root solution (e.g., Magisk v27 or SuperSU) is installed.
  2. Install the APK as above.
  3. Grant root permissions during setup via Magisk/SuperSU prompt.

4.3 Permissions Required and Initial Setup

MT Manager requires:

- **Storage**: Read/write access to internal/external storage.
- **Root** (optional): For system file access.
- **Install Apps**: For installing modified APKs.
- **Notifications**: For displaying alerts and status updates.

Setup steps:

1. Launch MT Manager and grant storage permissions.
2. For rooted devices, allow root access when prompted.
3. Configure settings (`Menu > Settings`):
   - Enable `Show Hidden Files` for system files.
   - Set theme (light/dark) for readability.
   - Define default paths (e.g., `/storage/emulated/0/Download`).
   - Enable `Auto-Backup` for edited APKs.
   - Configure notification preferences.

5 Navigating the Interface

MT Manager's interface is optimized for efficiency, with a dual-pane layout and robust navigation tools.

5.1 File Explorer Layout

The main screen includes:

- **Top Toolbar**: Buttons for new folder, search, sort, and settings.
- **Sidebar**: Quick access to storage locations (internal, SD card, root).
- **File List**: Displays files/folders with details (size, date, permissions).
- **Context Menu**: Long-press files for options like copy, rename, or APK Editor.
- **Notification Panel**: Access alerts and status updates.

5.2 Dual-Pane Mode

Dual-pane mode boosts productivity:

- **Left Pane**: Primary explorer for navigating directories.
- **Right Pane**: Secondary explorer or file preview.
- **Usage**: Drag files between panes to copy/move or tap `Menu > Dual Pane` to toggle.
- Tip: Use `Split View` for large screens (e.g., tablets).

5.3 Accessing Internal and External Storage

- **Internal Storage**: `/storage/emulated/0` (user files like downloads, media).
- **External Storage**: SD cards/USB drives, listed in the sidebar.
- **Root Directory**: On rooted devices, access `/system`, `/data`, or `/root`.

Tip: Use `Search > Advanced Search` to filter by file type (e.g., `.apk`).

6 APK Editing Workflow

Editing an APK involves decompiling, modifying, and recompiling. This section details the workflow with practical guidance.

6.1 Locating and Opening an APK File

1. Navigate to the APK (e.g., `/storage/emulated/0/Download`).
2. Long-press and select `APK Editor`.
3. Choose:
    - `Full Edit`: Decompiles code, resources, and manifest.
    - `Simple Edit`: For quick resource replacements (e.g., images).

- `Common Edit`: For basic changes (e.g., app name, icon).

Tip: Back up the original APK (`Menu > Backup`) before editing.

6.2 Viewing Contents

Decompilation reveals:

- `classes.dex`: App's compiled code in Dalvik bytecode.
- `smali/`: Decompiled Smali code (human-readable).
- `res/`: Resources (images in `res/drawable`, layouts in `res/layout`, strings in `res/values`).
- `AndroidManifest.xml`: App metadata, permissions, and activities.
- `assets/`: Non-compiled files (e.g., fonts, JSON).

6.3 Editing Smali Code and XML Files

- **Smali Code**:
  - Located in `smali/` or `smali_classesX/`.
  - Edit using MT Manager's text editor with Smali syntax highlighting.
  - Example: Disable a function by replacing logic with `return-void`.
- **XML Files**:
  - Edit `res/values/strings.xml` for text (e.g., app name, button labels).
  - Modify `res/layout/*.xml` for UI layouts.
  - Update `AndroidManifest.xml` for permissions or package name.

Listing 1: Smali edit to bypass a check

```
.method public isPremium()Z
    .registers 2
    const/4 v0, 0x1  # Return true for premium status
    return v0
.end method
```

6.4 Replacing Assets

- Navigate to `res/drawable` (images) or `res/raw` (sounds).
- Copy new assets, ensuring identical format (e.g., PNG, 512x512 pixels).
- Example: Replace `ic_launcher.png` with a custom icon.

7 Recompiling and Signing

Recompiling and signing ensure the modified APK is installable.

7.1 Rebuilding the APK After Edits

1. In APK Editor, tap `Recompile`.
2. Select `Optimize resources` to reduce APK size (recommended).
3. Monitor the progress bar; errors will be logged if compilation fails.

7.2 Using MT Manager's Auto-Sign Feature

1. After recompiling, select `Sign APK`.
2. MT Manager uses a test key by default.
3. For custom keys, use `Menu > Sign with Custom Key` and import a keystore.

7.3 Verifying the Signature

1. Select `APK Info` from the file menu.
2. Verify the `Signature` field shows a valid signature (e.g., SHA256 hash).

7.4 Installing the Modified APK

1. Tap the recompiled APK and select `Install`.
2. Uninstall the original app if signatures differ.
3. Enable `Unknown Sources` in Android settings.
4. Test the app to ensure modifications work.

8 Advanced Features

MT Manager offers specialized tools for advanced modding.

8.1 Hex Editor for Binary Modifications

- Access via `Open with > Hex Editor`.
- Modify binary files like `classes.dex` or resources.
- Example: Patch a hardcoded value (e.g., game lives) by editing its hex representation.

8.2 Text Editor for Scripts and Config Files

- Edit scripts (e.g., Shell, Python) or configs (e.g., `build.prop`).
- Supports syntax highlighting for Smali, XML, JSON, etc.
- Example: Change `ro.build.version.release` in `build.prop` to spoof Android version.

8.3 ZIP Compression and Extraction

- Extract ZIPs to view/modify contents (`Open with > ZIP Manager`).
- Create ZIPs for backups or mod distribution.

8.4 Permissions Manager and Manifest Tweaks

- Edit `AndroidManifest.xml` to modify permissions or package name.
- Use `Permissions Manager` for a graphical interface to toggle permissions.
- Example: Add `INTERNET` permission:

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

9 Working with Split APKs

Split APKs (App Bundles) are divided into multiple files for efficient delivery. This section explains how to handle them in MT Manager.

9.1 Understanding Split APKs

Split APKs consist of:

- **Base APK**: Contains core app code and resources.
- **Configuration APKs**: Device-specific resources (e.g., languages, screen densities).
- **Dynamic Feature APKs**: Optional modules downloaded on demand.

9.2 Extracting and Combining Split APKs

1. Place all split APKs in one folder.
2. Long-press the base APK (usually the largest file).
3. Select `APK Editor > Split APK Installer`.
4. Choose `Combine Split APKs` to merge into a single APK.
5. Alternatively, install all split APKs simultaneously using `Split APK Installer`.

9.3 Modding Split APKs

1. Extract all split APKs using `ZIP Manager`.
2. Modify resources in each APK as needed.
3. Recompile each modified APK separately.
4. Sign all APKs with the same key.
5. Reinstall using `Split APK Installer`.

10 Adding Custom Dialogues

Dialogues are UI elements that prompt user interaction. This section explains how to modify or add custom dialogues to apps.

10.1 Understanding Android Dialogues

Dialogues include:

- **Alert Dialogues**: Simple prompts with buttons.
- **Custom Dialogues**: Complex layouts with custom views.
- **Bottom Sheets**: Modal panels sliding from the bottom.

10.2 Modifying Existing Dialogues

1. Decompile the APK and locate dialogue layouts in `res/layout/`.
2. Identify dialogue-related Smali code in `smali/` (search for `AlertDialog`, `DialogFragment`).
3. Modify XML layouts to change appearance:

```
<!-- Example: Change button text in a dialogue -->
<Button
    android:id="@+id/dialog_button"
    android:text="Custom Action"
    android:background="#FF5722"/>
```

10.3 Adding New Dialogues

1. Create a new layout file in `res/layout/` (e.g., `custom_dialogue.xml`).
2. Add the dialogue logic to the relevant Smali file:

Listing 2: Adding a custom dialogue

```
.method public showCustomDialog()V
    .registers 3

    # Create a new AlertDialog builder
    new-instance v0, Landroid/app/AlertDialog$Builder;
    invoke-direct {v0, p0}, Landroid/app/AlertDialog$Builder;-><init
       >(Landroid/content/Context;)V

    # Set title and message
    const-string v1, "Custom Dialogue"
    invoke-virtual {v0, v1}, Landroid/app/AlertDialog$Builder;->
       setTitle(Ljava/lang/CharSequence;)Landroid/app/
       AlertDialog$Builder;

    const-string v1, "This is a custom dialogue added by MT Manager"
```

```
    invoke-virtual {v0, v1}, Landroid/app/AlertDialog$Builder;->
        setMessage(Ljava/lang/CharSequence;)Landroid/app/
        AlertDialog$Builder;

    # Set positive button
    const-string v1, "OK"
    new-instance v2, Lcom/example/MyDialogListener;
    invoke-direct {v2, p0}, Lcom/example/MyDialogListener;-><init>(
        Lcom/example/MainActivity;)V
    invoke-virtual {v0, v1, v2}, Landroid/app/AlertDialog$Builder;->
        setPositiveButton(Ljava/lang/CharSequence;Landroid/content/
        DialogInterface$OnClickListener;)Landroid/app/
        AlertDialog$Builder;

    # Create and show the dialogue
    invoke-virtual {v0}, Landroid/app/AlertDialog$Builder;->create()
        Landroid/app/AlertDialog;
    move-result-object v1
    invoke-virtual {v1}, Landroid/app/AlertDialog;->show()V

    return-void
.end method
```

10.4 Triggering Custom Dialogues

- Call the dialogue method from existing UI elements (e.g., button clicks).
- Add a button to an existing layout and reference it in Smali code.
- Example: Modify a button's click listener to show the custom dialogue.

11 Adding Custom Notifications

Notifications are alerts displayed in the notification panel. This section explains how to modify or add custom notifications to apps.

11.1 Understanding Android Notifications

Notifications include:

- **Basic Notifications**: Simple alerts with title and text.
- **Expanded Notifications**: Rich content with images and actions.
- **Foreground Service Notifications**: Persistent notifications for ongoing services.

11.2 Modifying Existing Notifications

1. Locate notification-related code in `smali/` (search for `Notification`, `NotificationMa`

2. Modify notification appearance and behavior:

Listing 3: Modifying notification appearance

```
.method public showNotification()V
    .registers 5

    # Create a notification builder
    new-instance v0, Landroidx/core/app/NotificationCompat$Builder;
    const-string v1, "default_channel"
    invoke-direct {v0, p0, v1}, Landroidx/core/app/
       NotificationCompat$Builder;-><init>(Landroid/content/Context;
       Ljava/lang/String;)V

    # Set notification content
    const-string v1, "Custom Notification"
    invoke-virtual {v0, v1}, Landroidx/core/app/
       NotificationCompat$Builder;->setContentTitle(Ljava/lang/
       CharSequence;)Landroidx/core/app/NotificationCompat$Builder;

    const-string v1, "This notification was added using MT Manager"
    invoke-virtual {v0, v1}, Landroidx/core/app/
       NotificationCompat$Builder;->setContentText(Ljava/lang/
       CharSequence;)Landroidx/core/app/NotificationCompat$Builder;

    # Set small icon
    const v1, 0x7f080001  # Reference to drawable resource
    invoke-virtual {v0, v1}, Landroidx/core/app/
       NotificationCompat$Builder;->setSmallIcon(I)Landroidx/core/
       app/NotificationCompat$Builder;

    # Set priority
    const/4 v1, 0x2  # PRIORITY_HIGH
    invoke-virtual {v0, v1}, Landroidx/core/app/
       NotificationCompat$Builder;->setPriority(I)Landroidx/core/app
       /NotificationCompat$Builder;

    # Build the notification
    invoke-virtual {v0}, Landroidx/core/app/
       NotificationCompat$Builder;->build()Landroid/app/Notification
       ;
    move-result-object v1

    # Get notification manager and show notification
    const-string v2, "notification"
    invoke-virtual {p0, v2}, Landroid/content/Context;->
       getSystemService(Ljava/lang/String;)Ljava/lang/Object;
    move-result-object v2
    check-cast v2, Landroid/app/NotificationManager;

    const/4 v3, 0x1  # Notification ID
    invoke-virtual {v2, v3, v1}, Landroid/app/NotificationManager;->
```

```
        notify(ILandroid/app/Notification;)V

    return-void
.end method
```

11.3 Adding Interactive Notifications

1. Add action buttons to notifications:

Listing 4: Adding notification actions

```
# Add action button
const-string v1, "Action"
new-instance v2, Landroid/content/Intent;
const-string v3, "com.example.ACTION_CUSTOM"
invoke-direct {v2, v3}, Landroid/content/Intent;-><init>(Ljava/lang/
    String;)V

const/4 v3, 0x0
invoke-static {p0, v3, v2, v3}, Landroid/app/PendingIntent;->
    getActivity(Landroid/content/Context;ILandroid/content/Intent;I)
    Landroid/app/PendingIntent;
move-result-object v2

invoke-virtual {v0, v1, v2}, Landroidx/core/app/
    NotificationCompat$Builder;->addAction(ILjava/lang/CharSequence;
    Landroid/app/PendingIntent;)Landroidx/core/app/
    NotificationCompat$Builder;
```

11.4 Creating Notification Channels

For Android 8.0+ (API 26+):

1. Create a notification channel in the app's main activity:

Listing 5: Creating notification channel

```
.method public createNotificationChannel()V
    .registers 4

    # Check if Android version is Oreo or higher
    sget v0, Landroid/os/Build$VERSION;->SDK_INT:I
    const/16 v1, 0x1a  # API level 26
    if-lt v0, v1, :cond_0

    # Create notification channel
    const-string v0, "default_channel"
    const-string v1, "Default Channel"
    const/4 v2, 0x3  # IMPORTANCE_DEFAULT
    new-instance v3, Landroid/app/NotificationChannel;
    invoke-direct {v3, v0, v1, v2}, Landroid/app/NotificationChannel
        ;-><init>(Ljava/lang/String;Ljava/lang/CharSequence;I)V
```

```
15
16      # Get notification manager and create channel
17      const-string v1, "notification"
18      invoke-virtual {p0, v1}, Landroid/content/Context;->
            getSystemService(Ljava/lang/String;)Ljava/lang/Object;
19      move-result-object v1
20      check-cast v1, Landroid/app/NotificationManager;
21
22      invoke-virtual {v1, v3}, Landroid/app/NotificationManager;->
            createNotificationChannel(Landroid/app/NotificationChannel;)V
23
24      :cond_0
25      return-void
26  .end method
```

12 Increasing Coins or Diamonds in Games

Modding game currency is one of the most popular uses of MT Manager. This section covers various methods to increase coins, diamonds, or other in-game resources.

12.1 Understanding Game Currency Systems

Games typically store currency values in:

- **Client-side storage**: Shared preferences, SQLite databases, or JSON files
- **Server-side storage**: Remote databases (harder to modify)
- **Memory**: Runtime values during gameplay

12.2 Method 1: Modifying Shared Preferences

Many games store currency values in shared preferences (XML files):

1. Navigate to `/data/data/com.example.game/shared_prefs/` (requires root).
2. Locate the preferences file (e.g., `game_prefs.xml`).
3. Edit the currency value:

```
1  <!-- Original -->
2  <int name="coins" value="100" />
3
4  <!-- Modified -->
5  <int name="coins" value="999999" />
```

12.3 Method 2: Modifying SQLite Databases

For games using SQLite databases:

1. Navigate to `/data/data/com.example.game/databases/` (requires root).
2. Open the database file with MT Manager's SQLite editor.
3. Locate the currency table and update the value:

```
-- Original
UPDATE player SET coins = 100 WHERE id = 1;

-- Modified
UPDATE player SET coins = 999999 WHERE id = 1;
```

12.4 Method 3: Modifying JSON Files

Games often store currency in JSON files:

1. Locate the JSON file (usually in `assets/` or `files/`).
2. Edit the currency value:

```
{
  "player": {
    "name": "Player1",
    "coins": 100,
    "diamonds": 50
  }
}

// Modified
{
  "player": {
    "name": "Player1",
    "coins": 999999,
    "diamonds": 99999
  }
}
```

12.5 Method 4: Smali Code Modification

For games with currency logic in code:

1. Decompile the APK and search for currency-related methods (e.g., `addCoins`, `getDiamonds`).
2. Modify the method to always return a high value:

Listing 6: Modifying coin retrieval method

```
.method public getCoins()I
    .registers 2

    # Original logic might call database or preferences
    # Replace with fixed high value
    const v0, 0xf4240  # 1,000,000 in hex
```

```
    return v0
.end method
```

12.6 Method 5: Memory Editing with GameGuardian

For real-time modification during gameplay:

1. Install GameGuardian (requires root).
2. Launch the game and GameGuardian.
3. Search for the current currency value.
4. Modify the value to desired amount.

12.7 Method 6: Patching Purchase Methods

For in-app purchases:

1. Locate the purchase verification method (e.g., `onPurchaseCompleted`).
2. Modify to always grant currency without actual purchase:

Listing 7: Patching purchase method

```
.method public onPurchaseCompleted(Ljava/lang/String;I)V
    .registers 4

    # Original purchase processing code
    # Replace with direct currency grant
    const v0, 0x186a0  # 100,000 in hex
    invoke-virtual {p0, v0}, Lcom/example/GameActivity;->addCoins(I)
       V

    return-void
.end method
```

12.8 Testing Currency Modifications

1. Apply the modification and recompile/sign the APK.
2. Launch the game and verify currency values.
3. Test spending to ensure the game doesn't crash.
4. Check if the game has server-side validation that might reset values.

Warning: Server-side games will verify currency values online. Client-side mods may not work for online games.

13 Removing Ads from Apps

Ads can disrupt app experiences. This section explains how to remove them using MT Manager.

13.1 Identifying Ad-Related Code

1. Open the APK in `APK Editor > Full Edit`.
2. Search `smali/` for ad frameworks (e.g., `com.google.ads`, `com.mopub`, `com.facebook.a`
3. Use `Menu > Search in Smali` and enter keywords like `ad`, `banner`, or `interstitial`.

13.2 Modifying Smali to Disable Ads

1. Locate ad-related methods (e.g., `showBannerAd()`).
2. Replace the method's logic with `return-void` to skip ad display:

```
.method public showBannerAd()V
    .registers 1
    return-void  # Prevents ad from loading
.end method
```

```
const/4 v0, 0x0  # Change to false
invoke-virtual {p0, v0}, Lcom/example/AdView;->setVisibility(I)V
```

13.3 Removing Ad Permissions

1. Open `AndroidManifest.xml`.
2. Remove ad-related permissions (e.g., `com.google.android.gms.permission.AD_ID`).
3. Example:

```
<!-- Remove this line -->
<uses-permission android:name="com.google.android.gms.permission.
   AD_ID"/>
```

13.4 Testing and Validation

1. Recompile and sign the APK.
2. Install and test the app in ad-heavy sections (e.g., main menu, transitions).
3. Use a network monitor (e.g., Charles Proxy) to confirm no ad requests are sent.

Warning: Over-editing Smali can cause crashes. Test incrementally and keep backups.

14 Bypassing Payment Features

Bypassing payment features allows access to premium content without payment, but must be done ethically for personal use.

14.1 Locating Payment Checks

1. Decompile the APK in `APK Editor > Full Edit`.
2. Search `smali/` for methods like `isPremium()`, `isSubscribed()`, or `checkPurchase()`.
3. Use keywords like `billing`, `purchase`, or `premium`.

14.2 Modifying Smali to Unlock Features

1. Find the premium check method, e.g.:

```
.method public isPremium()Z
    .registers 2
    invoke-static {}, Lcom/example/Billing;->checkPurchase()Z
    move-result v0
    return v0
.end method
```

```
.method public isPremium()Z
    .registers 2
    const/4 v0, 0x1  # Force true
    return v0
.end method
```

14.3 Patching Billing Libraries

1. Identify billing libraries (e.g., `com.android.billingclient`).
2. Disable billing calls by replacing methods with `return-void` or setting purchase status to true.
3. Example: Patch `verifyPurchase()` to return true.

14.4 Ethical Considerations and Testing

1. **Ethics**: Use for personal testing only; distributing modified APKs is illegal.
2. Recompile, sign, and install the APK.
3. Test premium features (e.g., ad-free mode, exclusive content).
4. If the app uses server-side checks, client-side mods may fail.

Warning: Server-side verification cannot be bypassed with client-side edits. Always test thoroughly.

15 Modifying App User Interfaces

Customizing an app's UI enhances its look and feel, such as changing themes, icons, or layouts.

15.1 Changing App Icons

1. Navigate to `res/drawable` or `res/mipmap`.
2. Replace `ic_launcher.png` with a new icon (same resolution, e.g., 512x512).
3. Update `AndroidManifest.xml` if the icon reference differs:

```
<application android:icon="@mipmap/new_icon">
```

15.2 Modifying App Names

1. Open `res/values/strings.xml`.
2. Change the `app_name` string:

```
<string name="app_name">Custom App Name</string>
```

15.3 Customizing Layouts

1. Navigate to `res/layout` (e.g., `activity_main.xml`).
2. Modify XML to adjust UI elements (e.g., button colors, text sizes).
3. Example: Change a button's background color:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#FF0000"  <!-- Red background -->
    android:text="Click Me"/>
```

15.4 Changing Themes and Colors

1. Open `res/values/colors.xml` to define new colors:

```
<color name="primary_color">#2196F3</color>
```

```
<style name="AppTheme" parent="Theme.AppCompat.Light">
    <item name="colorPrimary">@color/primary_color</item>
</style>
```

15.5 Testing UI Changes

1. Recompile, sign, and install the APK.
2. Test the app to verify UI changes (e.g., new icons, colors).
3. Use Android's `Developer Options > Layout Bounds` to inspect UI elements.

Warning: Incorrect XML syntax can cause crashes. Validate changes with `XML Lint` in MT Manager.

16 Legal and Ethical Considerations

Modding carries legal and ethical responsibilities.

16.1 Respecting Developer Rights

- Modifying apps may violate terms of service or EULAs.
- Use mods for personal experimentation, not commercial gain.
- Respect intellectual property by avoiding unauthorized distribution.

16.2 Avoiding Piracy and Malware

- **Piracy**: Distributing modified APKs without permission is illegal.
- **Malware**: Download tools/APKs from trusted sources. Scan with antivirus (e.g., Malwarebytes).
- Use `APK Info` to check for suspicious permissions.

16.3 Safe Distribution Practices

- Document changes clearly if sharing mods.
- Obtain developer consent before distribution.
- Use platforms like GitHub for open-source mods with proper licensing.

17 Troubleshooting and Tips

Modding errors are common; this section provides fixes and best practices.

17.1 Common Errors and How to Fix Them

- **Signature Mismatch**: Uninstall the original app before installing the modified APK.
- **Recompile Errors**: Check `Error Log` for Smali/XML syntax issues.
- **App Crashes**: Revert changes using backups and test incrementally.
- **Split APK Issues**: Ensure all APKs are signed with the same key.
- **Notification Problems**: Verify notification channels are properly created for Android 8.0+.
- **Currency Reset**: Server-side games will verify and reset modified currency values.

17.2 Backup Strategies

- Back up original APKs (`Menu > Backup`).
- Use TWRP for full device backups (rooted devices).
- Store backups on external storage or cloud (e.g., Google Drive).
- Create versioned backups for complex mods.

17.3 Compatibility Issues

- Match APK to Android version (e.g., Android 12, 13).
- Verify architecture (ARM64, ARMv7) via `APK Info`.
- Test on an emulator (e.g., BlueStacks) before deploying to a primary device.
- For split APKs, ensure all parts are compatible with your device.

18 Resources and Communities

The modding community offers valuable support and learning resources.

18.1 XDA Developers Forum

- Visit XDA Developers for tutorials and forums.
- Search for MT Manager or app-specific modding threads.
- Explore **Artificial Intelligence & Machine Learning**: XDA AI & ML.
- Visit **XDA Computing**: XDA Computing.

18.2 Telegram Modding Groups

- Join MT Manager Official or general modding channels.
- Follow group rules to avoid bans.
- For support, contact `@mtmanager_support_bot`.

18.3 YouTube Tutorials and GitHub Repositories

- Search YouTube for "MT Manager modding" or app-specific tutorials.
- Explore GitHub for Smali patches or modding scripts (e.g., `android-modding`).
- Check out repositories for split APK tools and notification libraries.

19 VIP Features and Store

MT Manager offers a VIP version with enhanced capabilities:

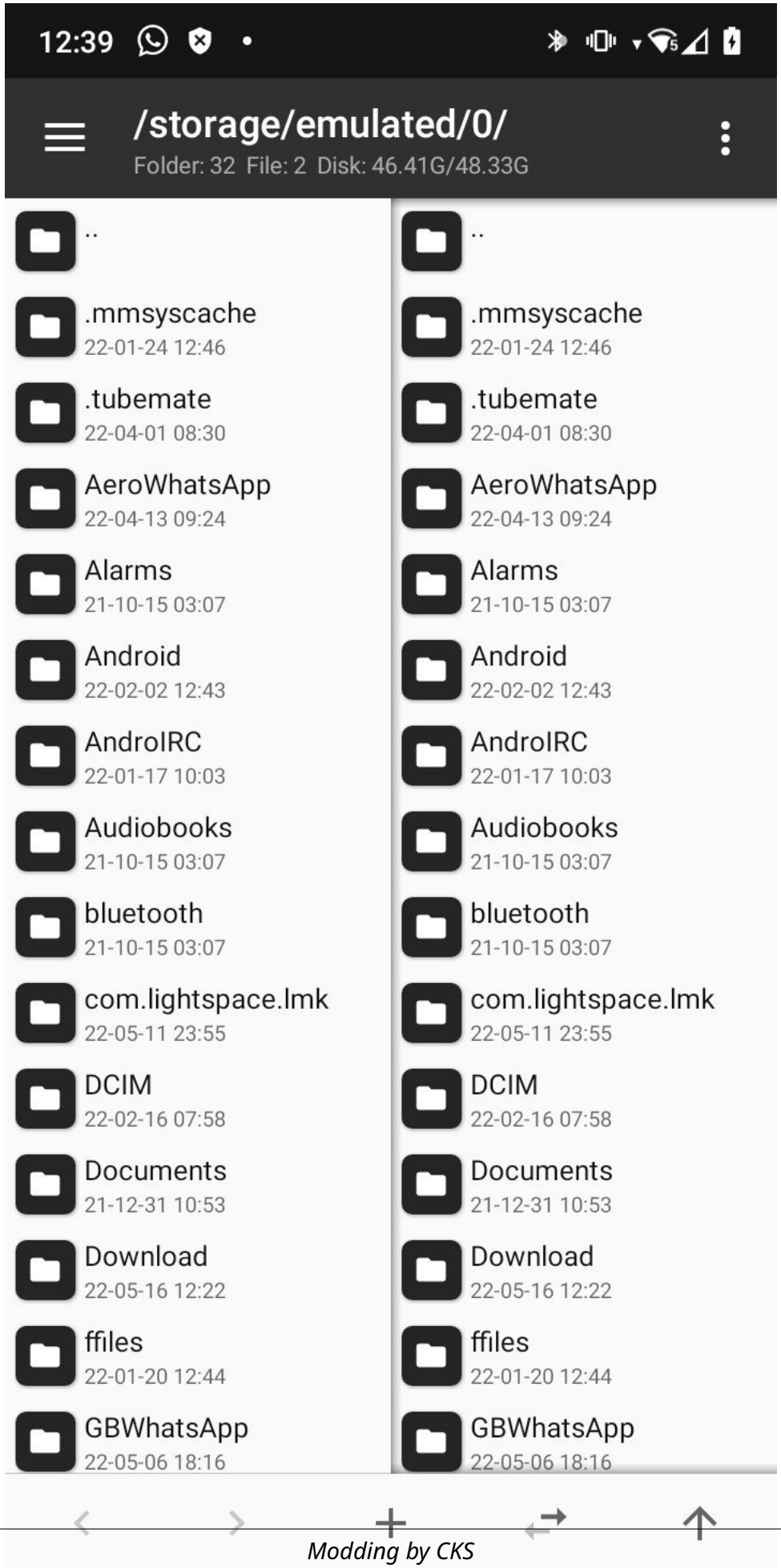
- **No Ads**
- **Advanced Tools**
- **Early Access**
- **Support Developer**

Official Store: paypal.mt2.cn (only official).

20 Getting Support

For issues with VIP or bugs:

- Contact official support bot: `@mtmanager_support_bot`
- Do not use cracked versions — they may contain malware.
- Check the official Telegram channel for updates and announcements.


12:39
/storage/emulated/0/
Folder: 32 File: 2 Disk: 46.41G/48.33G
..
.mmsyscache
22-01-24 12:46
.tubemate
22-04-01 08:30
AeroWhatsApp
22-04-13 09:24
Alarms
21-10-15 03:07
Android
22-02-02 12:43
AndroIRC
22-01-17 10:03
Audiobooks
21-10-15 03:07
bluetooth
21-10-15 03:07
com.lightspace.lmk
22-05-11 23:55
DCIM
22-02-16 07:58
Documents
21-12-31 10:53
Download
22-05-16 12:22
ffiles
22-01-20 12:44
GBWhatsApp
22-05-06 18:16

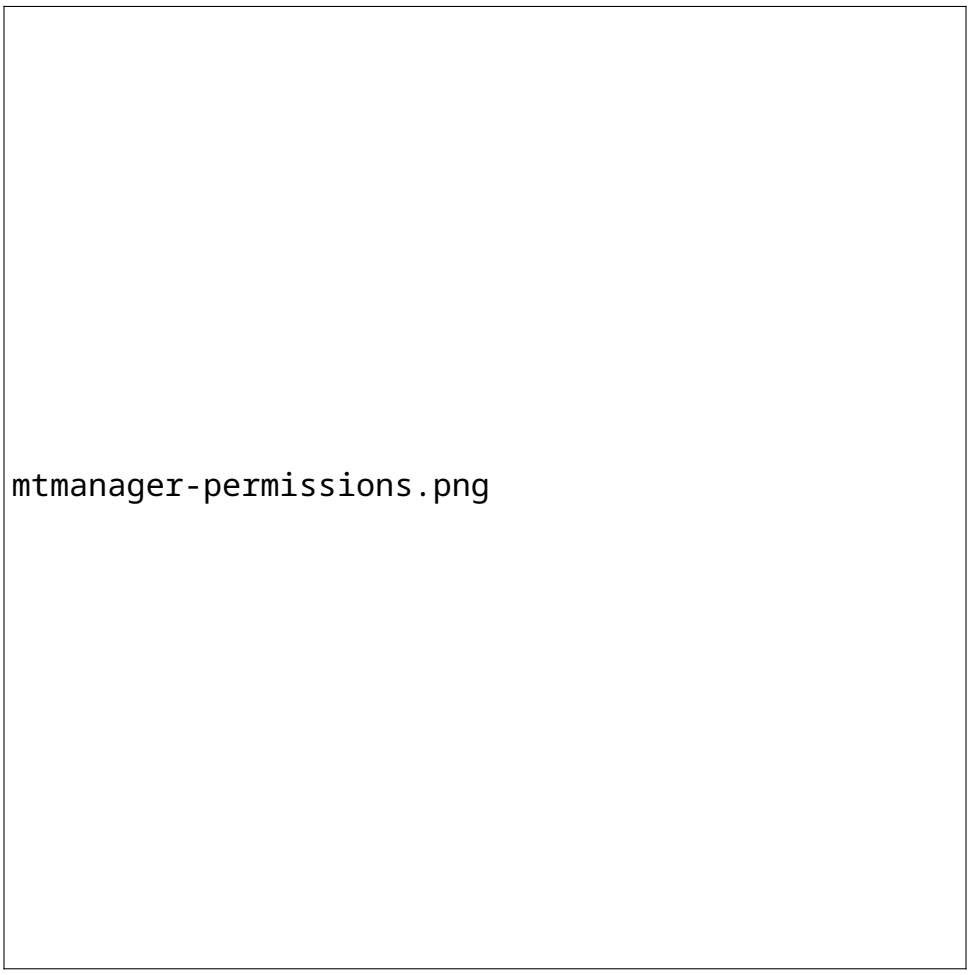


Figure 3: MT Manager requesting storage and root permissions on first launch.

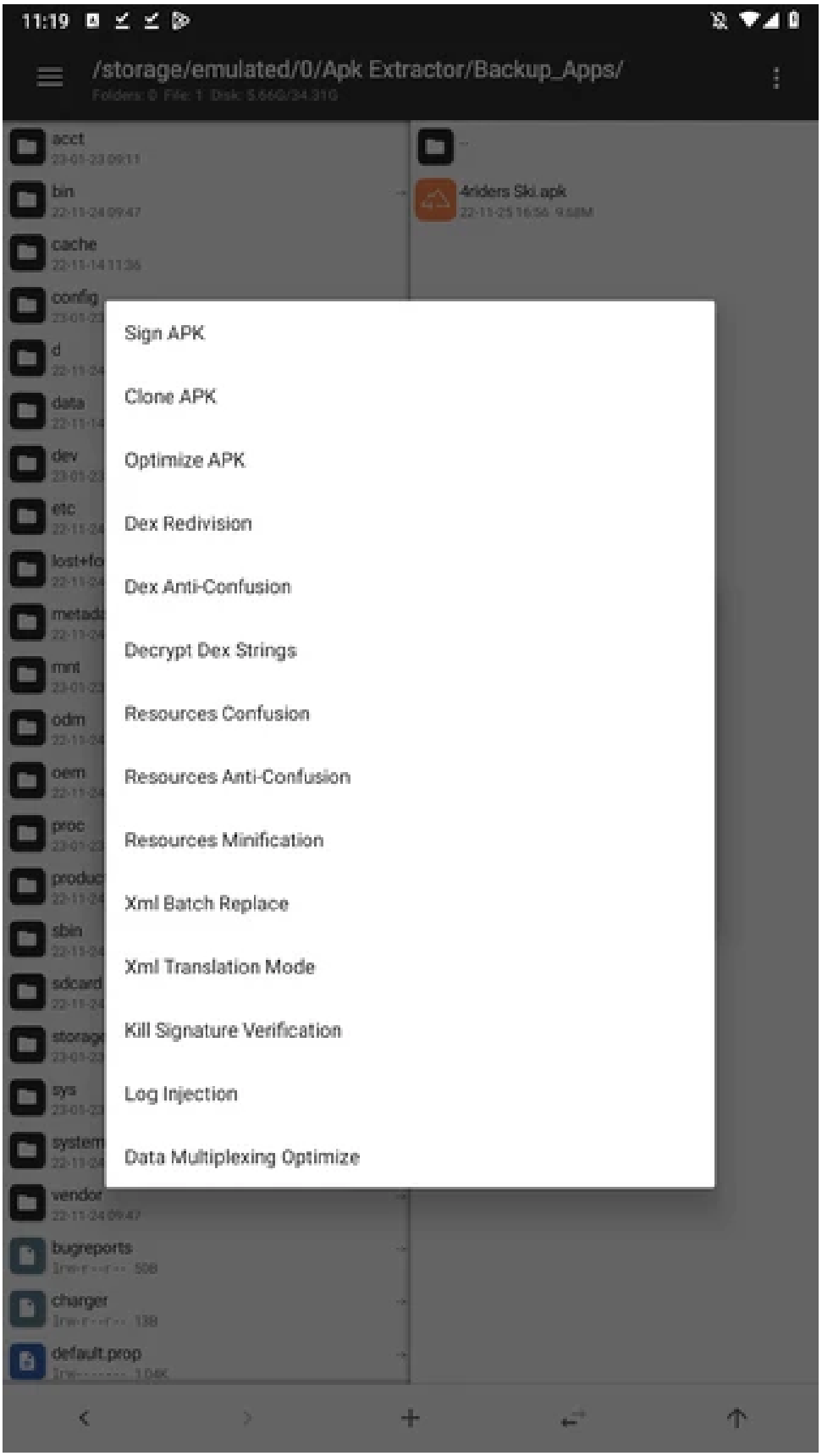


Figure 4: Dual-pane view showing internal storage and an APK file.

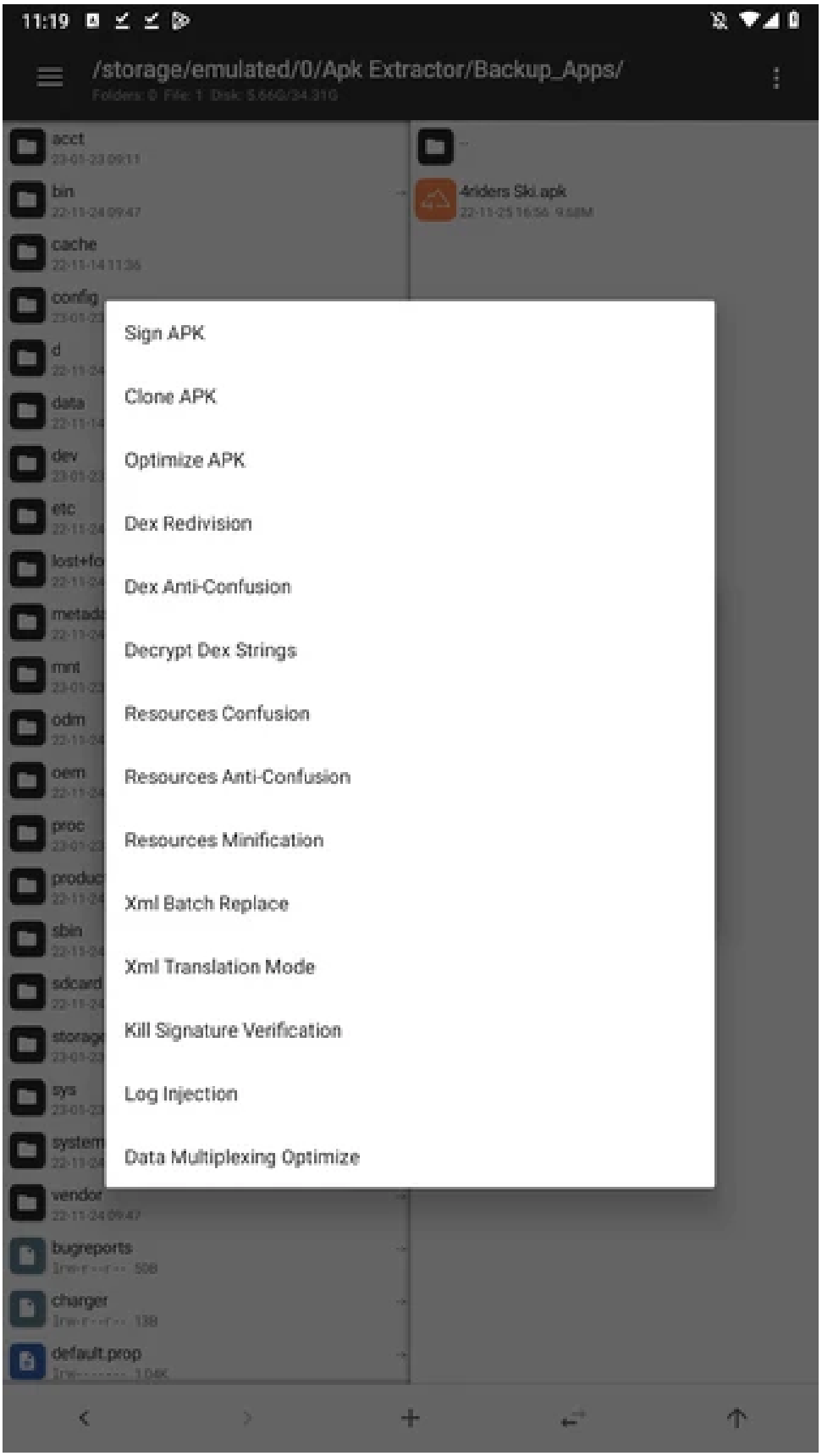


Figure 5: APK Editor view with classes.dex, resources, and manifest.

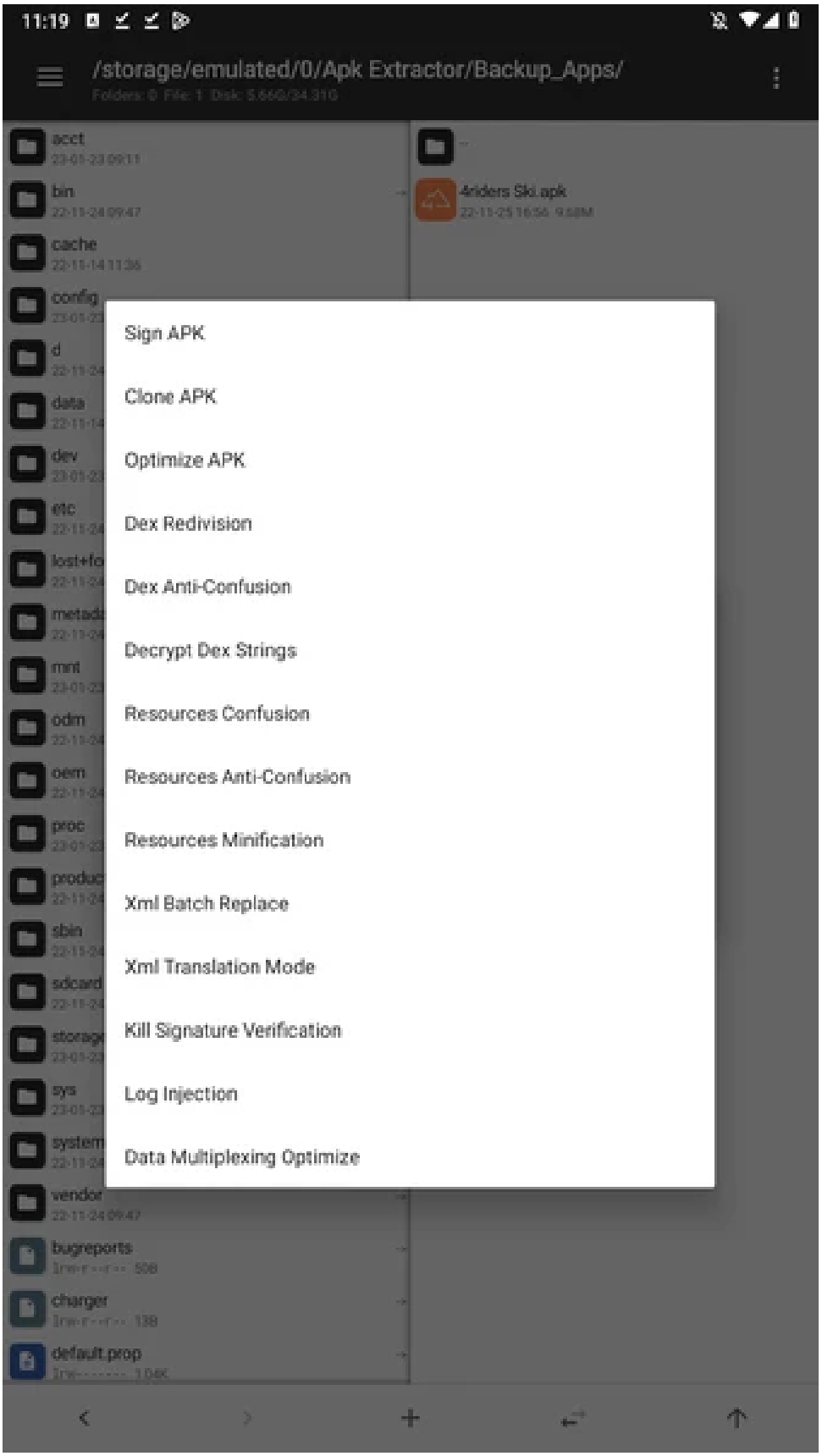


Figure 6: Confirmation of successful APK signing in MT Manager.

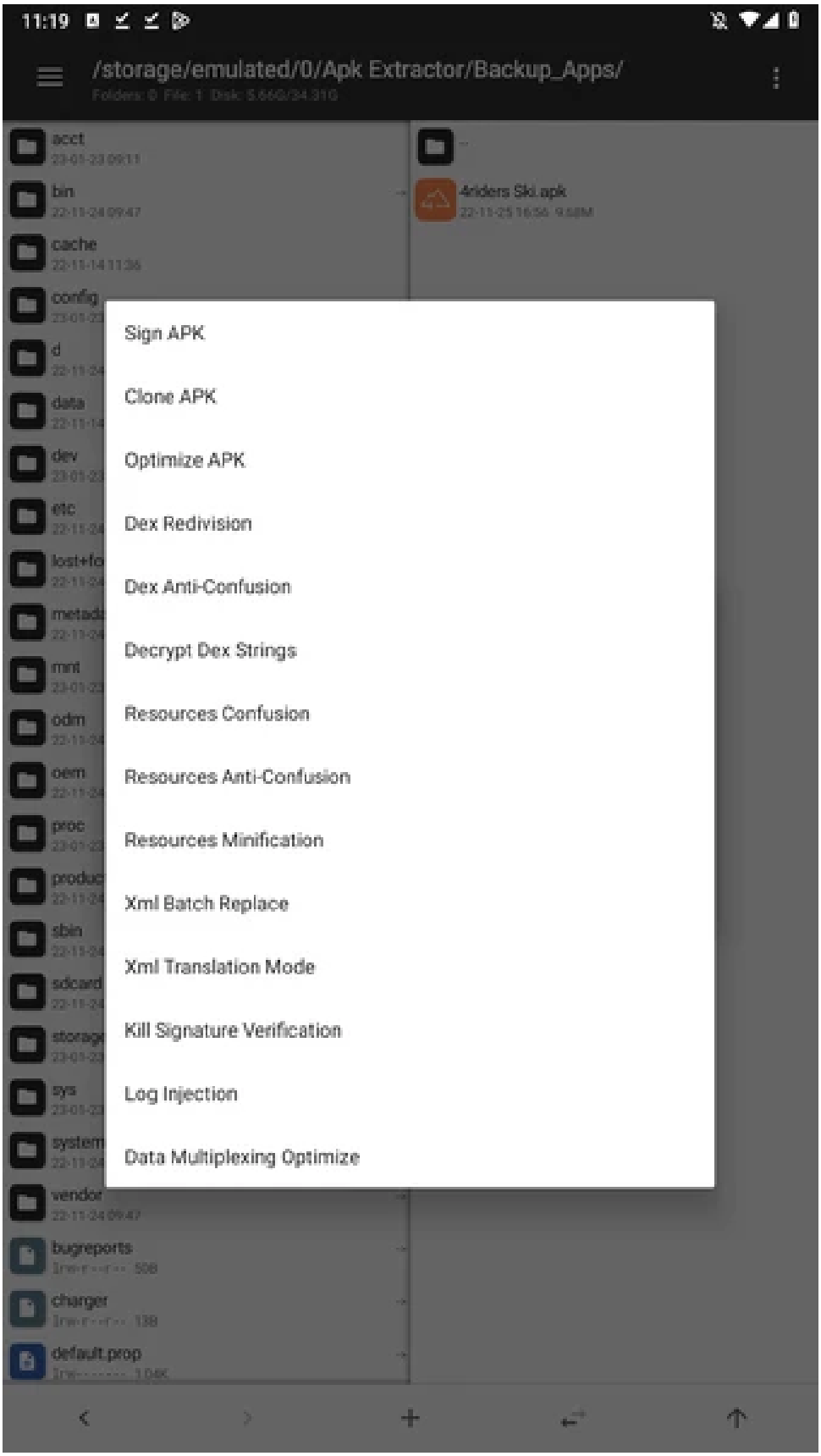


Figure 7: Hex editor modifying a binary resource in an APK.

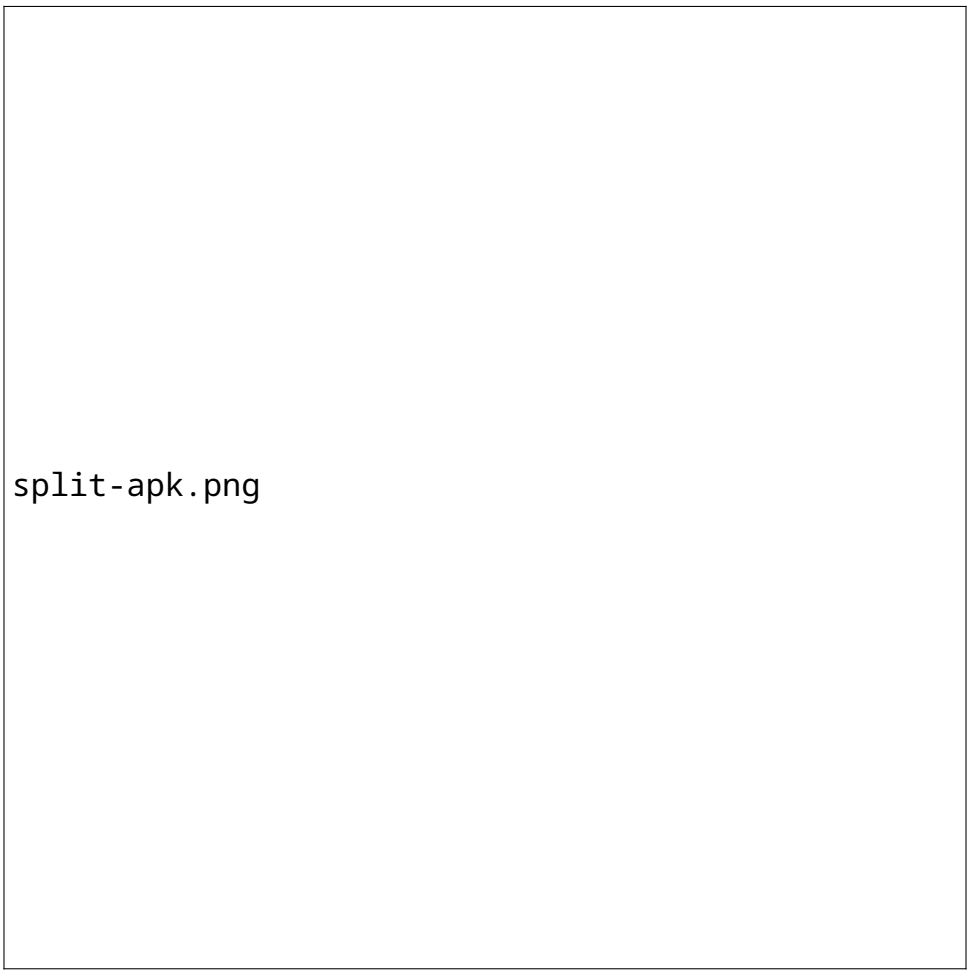


Figure 8: MT Manager’s Split APK Installer combining multiple APK files.

Figure 9: Example of a custom dialogue added to an app using MT Manager.

Figure 10: Example of a custom notification added to an app using MT Manager.

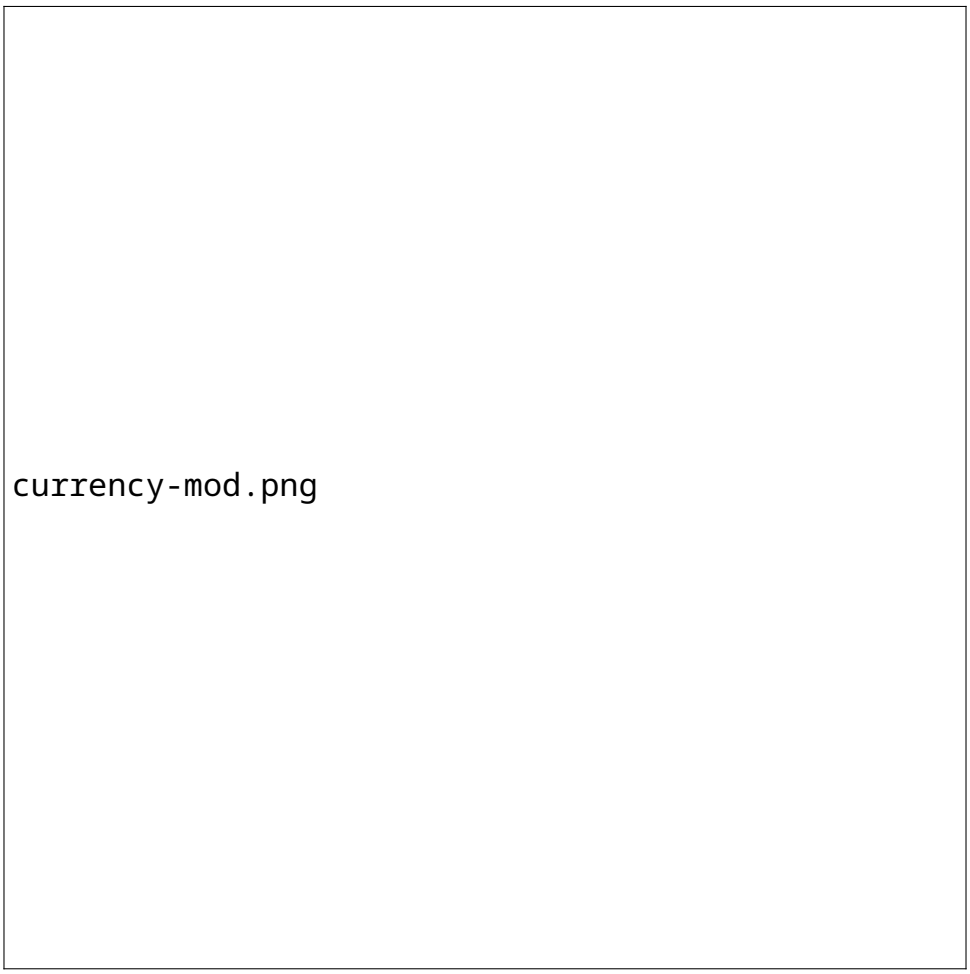


Figure 11: Example of modified currency values in a game.

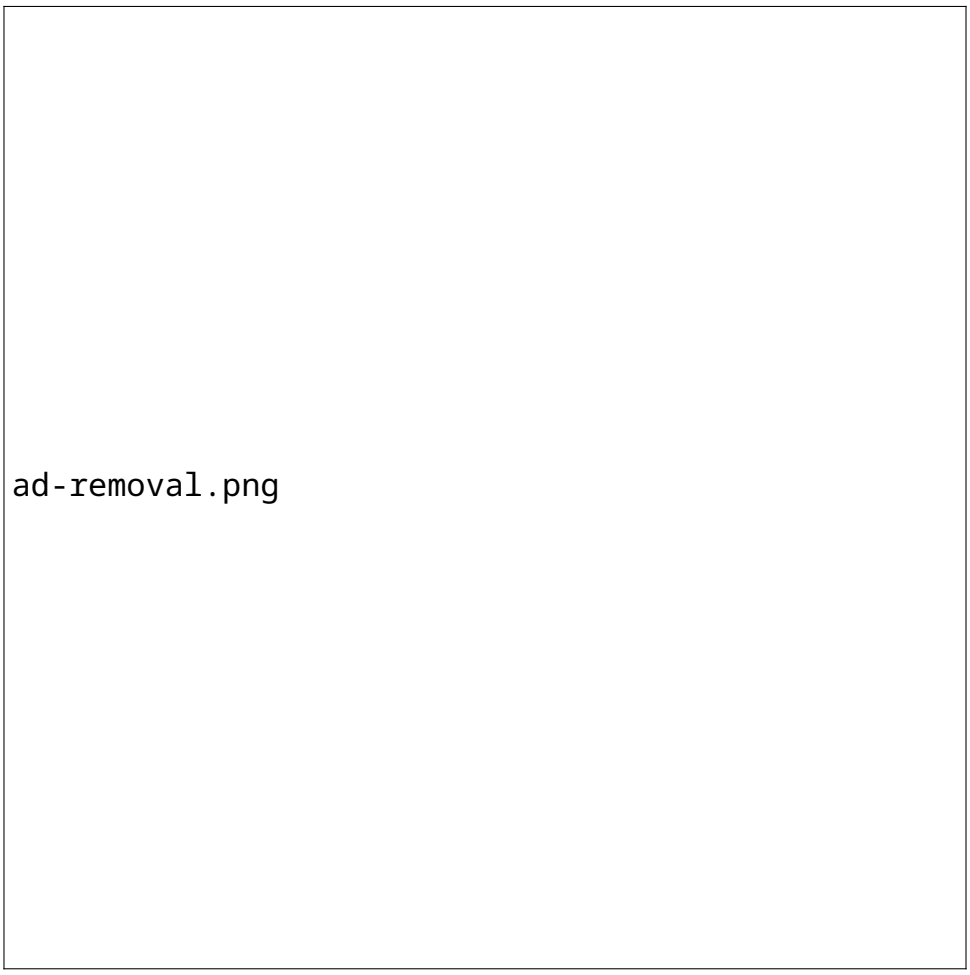


Figure 12: Editing Smali code to disable an ad banner.

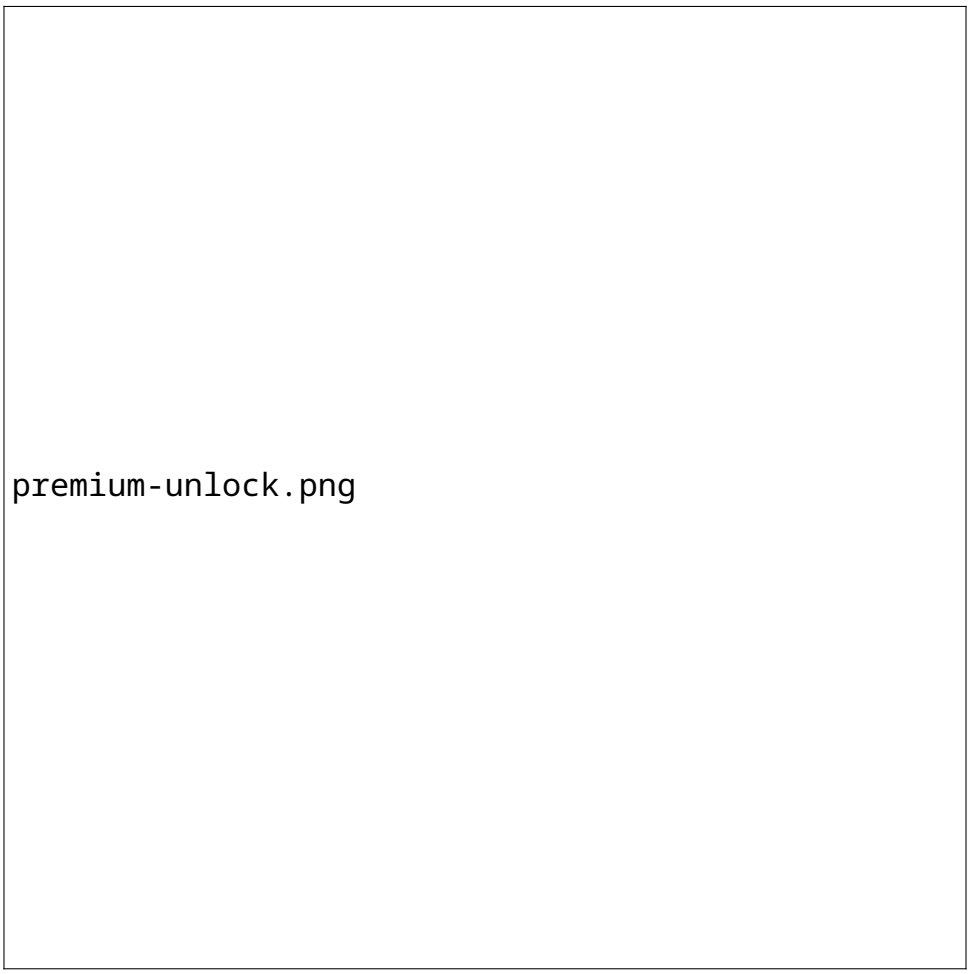


Figure 13: Editing Smali to bypass a premium check.

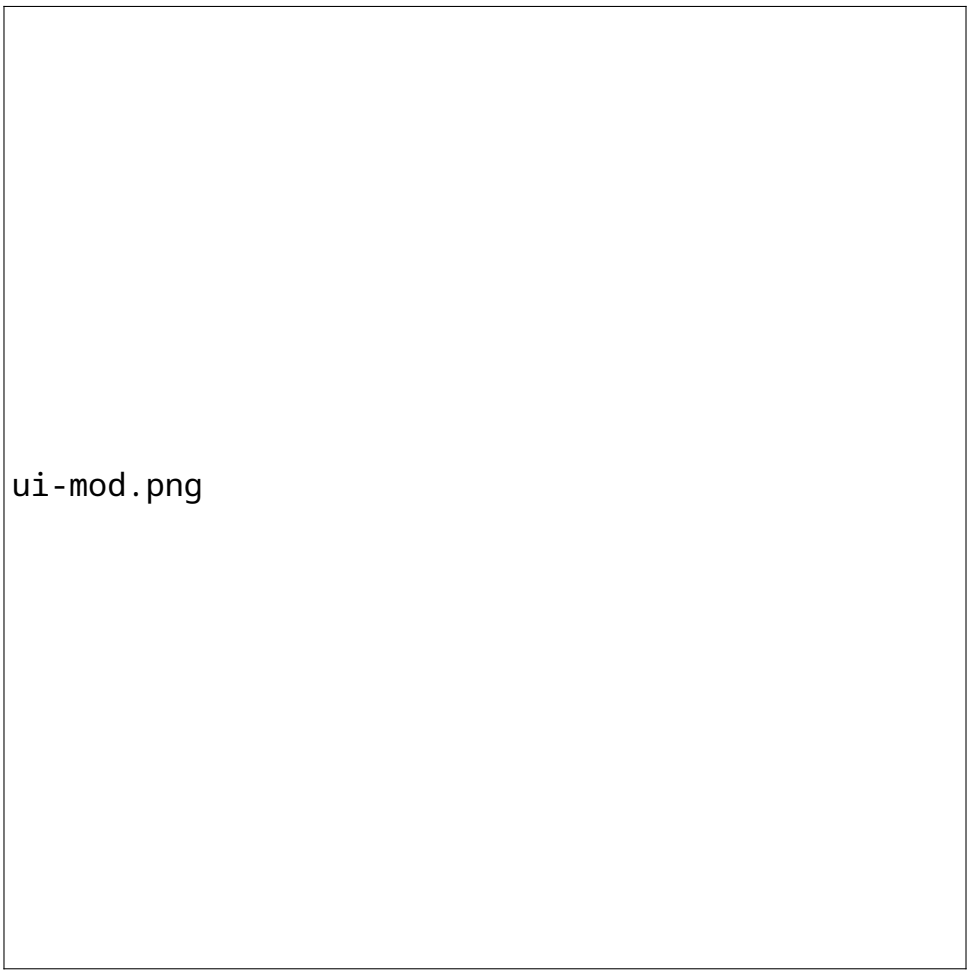


Figure 14: Editing a layout XML to change button color.

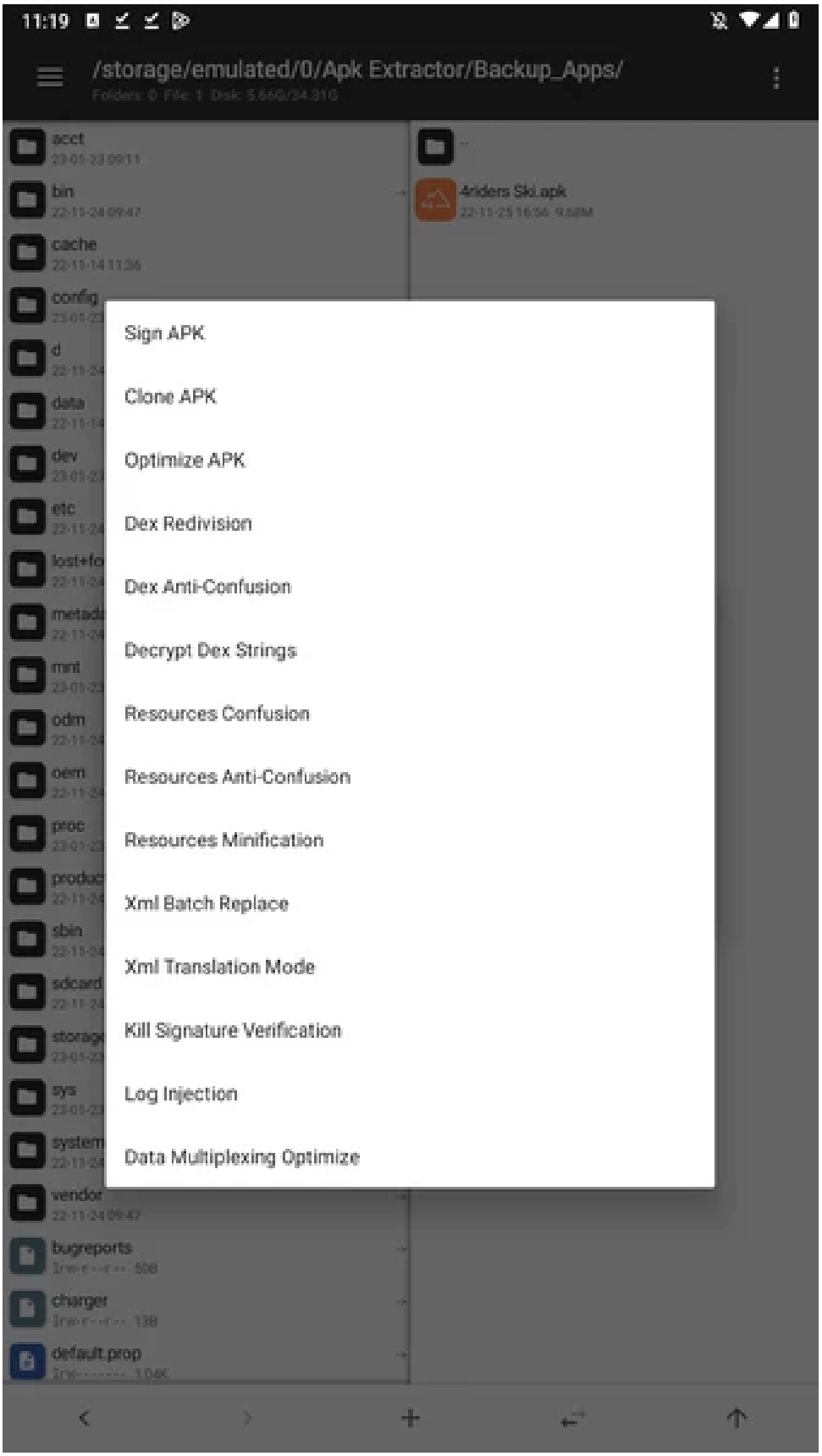


Figure 15: Error log showing a Smali syntax error during recompilation.