

Meet Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA (<https://www.jetbrains.com/idea/>). On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform (<https://cloud.google.com/tools/android-studio/docs/>), making it easy to integrate Google Cloud Messaging and App Engine

This page provides an introduction to basic Android Studio features. For a summary of the latest changes, see [Android Studio release notes](/studio/releases) (/studio/releases).

Project structure

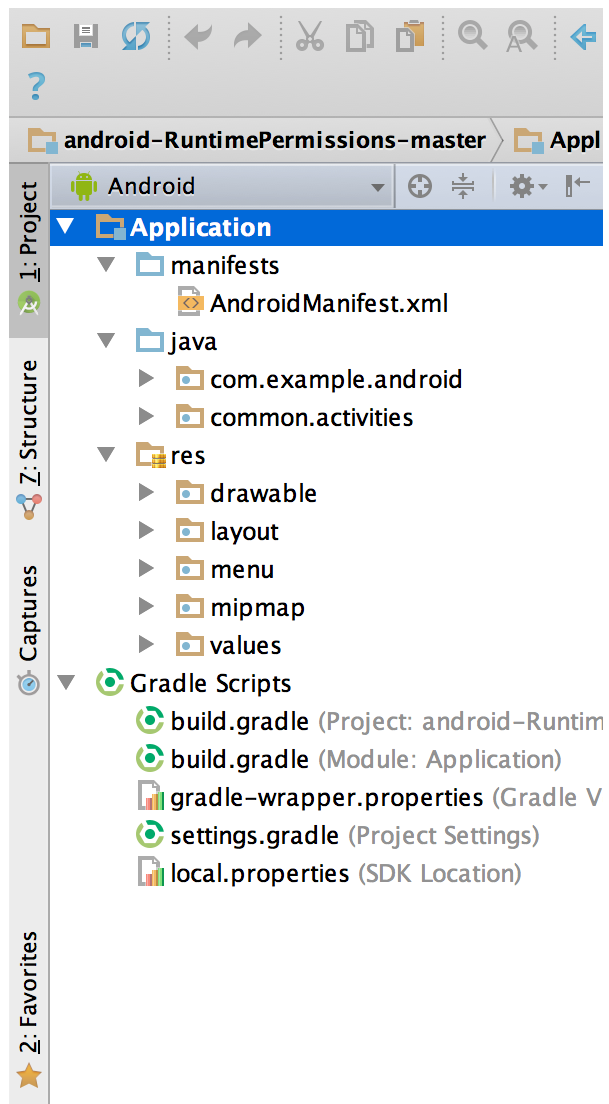


Figure 1. The project files in Android view.

Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the `AndroidManifest.xml` file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown (in figure 1, it's showing as **Android**).

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

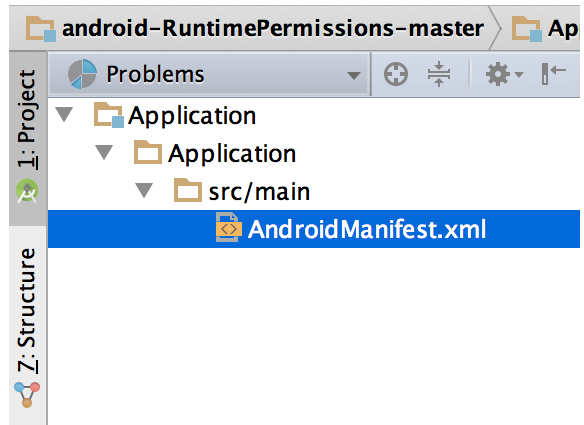
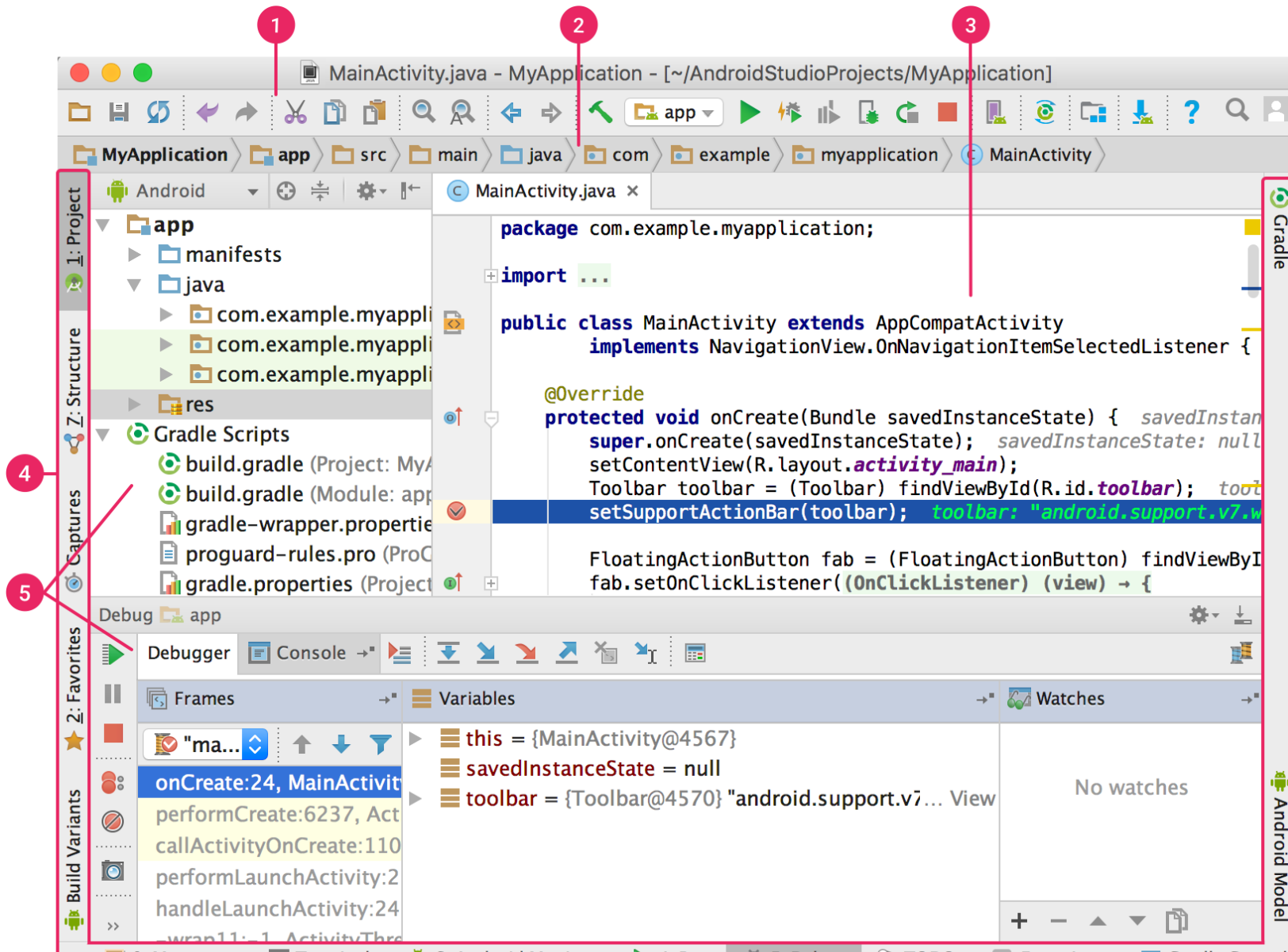


Figure 2. The project files in Problems view, showing a layout file with a problem.

For more information, see [Projects overview](/studio/projects) (/studio/projects).

The user interface

The Android Studio main window is made up of several logical areas identified in figure 3.



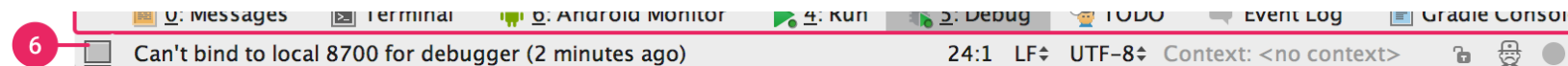


Figure 3. The Android Studio main window.


- 1 The **toolbar** lets you carry out a wide range of actions, including running your app and launching Android tools.
- 2 The **navigation bar** helps you navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
- 3 The **editor window** is where you create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
- 4 The **tool window bar** runs around the outside of the IDE window and contains the buttons that allow you to expand or collapse individual tool windows.
- 5 The **tool windows** give you access to specific tasks like project management, search, version control, and more. You can expand them and collapse them.
- 6 The **status bar** displays the status of your project and the IDE itself, as well as any warnings or messages.

You can organize the main window to give yourself more screen space by hiding or moving toolbars and tool windows. You can also use keyboard shortcuts to access most IDE features.

At any time, you can search across your source code, databases, actions, elements of the user interface, and so on, by double-pressing the Shift key, or clicking the magnifying glass in the upper right-hand corner of the Android Studio window. This can be very useful if, for example, you are trying to locate a particular IDE action that you have forgotten how to trigger.

Tool windows

Instead of using preset perspectives, Android Studio follows your context and automatically brings up relevant tool windows as you work. By default, the most commonly used tool windows are pinned to the tool window bar at the edges of the application window.

- To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.
- To return to the current default tool window layout, click **Window > Restore Default Layout** or customize your default layout by clicking **Window > Store Current Layout as Default**.
- To show or hide the entire tool window bar, click the window icon  in the bottom left-hand corner of the Android Studio window.
- To locate a specific tool window, hover over the window icon and select the tool window from the menu.

You can also use keyboard shortcuts to open tool windows. Table 1 lists the shortcuts for the most common windows.

Table 1. Keyboard shortcuts for some useful tool windows.

Tool window	Windows and Linux	Mac
Project	Alt+1	Command+1
Version Control	Alt+9	Command+9
Run	Shift+F10	Control+R

Debug	Shift+F9	Control+D
Logcat	Alt+6	Command+6
Return to Editor	Esc	Esc
Hide All Tool Windows	Control+Shift+F12	Command+Shift+F12

If you want to hide all toolbars, tool windows, and editor tabs, click **View > Enter Distraction Free Mode**. This enables *Distraction Free Mode*. To exit Distraction Free Mode, click **View > Exit Distraction Free Mode**.

You can use *Speed Search* to search and filter within most tool windows in Android Studio. To use Speed Search, select the tool window and then type your search query.

For more tips, see [Keyboard shortcuts](/studio/intro/keyboard-shortcuts) (/studio/intro/keyboard-shortcuts).

Code completion

Android Studio has three types of code completion, which you can access using keyboard shortcuts.

Table 2. Keyboard shortcuts for code completion.

Type	Description	Windows and Linux	Mac
Basic Completion	Displays basic suggestions for variables, types, methods, expressions, and so on. If you call basic completion twice in a row, you see more results, including private members and non-imported static members.	Control+Space	Control+Space

Smart Completion	Displays relevant options based on the context. Smart completion is aware of the expected type and data flows. If you call Smart Completion twice in a row, you see more results, including chains.	Control+Shift+Space Control+Shift+Space
------------------	---	--

Statement Completion	Completes the current statement for you, adding missing parentheses, brackets, braces, formatting, etc.	Control+Shift+Enter Shift+Command+Enter
----------------------	---	---

You can also perform quick fixes and show intention actions by pressing **Alt+Enter**.

Find sample code

The Code Sample Browser in Android Studio helps you find high-quality, Google-provided Android code samples based on the currently highlighted symbol in your project. For more information, see [Find sample code](/studio/write/sample-code) (/studio/write/sample-code).

Navigation

Here are some tips to help you move around Android Studio.

- Switch between your recently accessed files using the *Recent Files* action. Press **Control+E** (**Command+E** on a Mac) to bring up the Recent Files action. By default, the last accessed file is selected. You can also access any tool window through the left column in this action.
- View the structure of the current file using the *File Structure* action. Bring up the File Structure action by pressing **Control+F12** (**Command+F12** on a Mac). Using this action, you can quickly navigate to any part of your current

file.

- Search for and navigate to a specific class in your project using the *Navigate to Class* action. Bring up the action by pressing **Control+N** (**Command+O** on a Mac). *Navigate to Class* supports sophisticated expressions, including camel humps, paths, line navigate to, middle name matching, and many more. If you call it twice in a row, it shows you the results out of the project classes.
- Navigate to a file or folder using the *Navigate to File* action. Bring up the *Navigate to File* action by pressing **Control+Shift+N** (**Command+Shift+O** on a Mac). To search for folders rather than files, add a / at the end of your expression.
- Navigate to a method or field by name using the *Navigate to Symbol* action. Bring up the *Navigate to Symbol* action by pressing **Control+Shift+Alt+N** (**Command+Option+O** on a Mac).
- Find all the pieces of code referencing the class, method, field, parameter, or statement at the current cursor position by pressing **Alt+F7** (**Option+F7** on a Mac).

Style and formatting

As you edit, Android Studio automatically applies formatting and styles as specified in your code style settings. You can customize the code style settings by programming language, including specifying conventions for tabs and indents, spaces, wrapping and braces, and blank lines. To customize your code style settings, click **File > Settings > Editor > Code Style** (**Android Studio > Preferences > Editor > Code Style** on a Mac.)

Although the IDE automatically applies formatting as you work, you can also explicitly call the *Reformat Code* action by pressing **Control+Alt+L** (**Opt+Command+L** on a Mac), or auto-indent all lines by pressing **Control+Alt+I** (**Control+Option+I** on a Mac).

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mActionBar = getSupportActionBar();  
    mActionBar.setDisplayHomeAsUpEnabled(true);  
}
```

Figure 4. Code before formatting.

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    mActionBar = getSupportActionBar();  
    mActionBar.setDisplayHomeAsUpEnabled(true);  
    // Get reference to the drawer layout and set event listener
```

Formatted 7 lines

Show reformat dialog: ⌘⇧⌘L

Figure 5. Code after formatting.

Version control basics

Android Studio supports a variety of version control systems (VCS's), including Git, GitHub, CVS, Mercurial, Subversion, and Google Cloud Source Repositories.

After importing your app into Android Studio, use the Android Studio VCS menu options to enable VCS support for the desired version control system, create a repository, import the new files into version control, and perform other version control operations:

1. From the Android Studio **VCS** menu, click **Enable Version Control Integration**.
2. From the drop-down menu, select a version control system to associate with the project root, and then click **OK**.

The VCS menu now displays a number of version control options based on the system you selected.

Note: You can also use the **File > Settings > Version Control** menu option to set up and modify the version control settings.

Gradle build system

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the [Android plugin for Gradle](/studio/releases/gradle-plugin) (/studio/releases/gradle-plugin). This build system runs as an integrated tool from the Android Studio menu, and independently from the command line. You can use the features of the build system to do the following:

- Customize, configure, and extend the build process.
- Create multiple APKs for your app, with different features using the same project and modules.
- Reuse code and resources across source sets.

By employing the flexibility of Gradle, you can achieve all of this without modifying your app's core source files. Android Studio build files are named `build.gradle`. They are plain text files that use [Groovy](http://groovy-lang.org) (http://groovy-lang.org) syntax to configure the build with elements provided by the Android plugin for Gradle. Each project has one top-level build file for the entire project and separate module-level build files for each module. When you import an existing project, Android Studio automatically generates the necessary build files.

To learn more about the build system and how to configure, see [Configure your build](/studio/build) (/studio/build).

Build variants

The build system can help you create different versions of the same application from a single project. This is useful when you have both a free version and a paid version of your app, or if you want to distribute multiple APKs for different device configurations on Google Play.

For more information about configuring build variants, see [Configure build variants](/studio/build/build-variants) (/studio/build/build-variants).

Multiple APK support

Multiple APK support allows you to efficiently create multiple APKs based on screen density or ABI. For example, you can create separate APKs of an app for the hdpi and mdpi screen densities, while still considering them a single variant and allowing them to share test APK, javac, dx, and ProGuard settings.

For more information about multiple APK support, read [Build multiple APKs](/studio/build/configure-apk-splits) (/studio/build/configure-apk-splits).

Resource shrinking

Resource shrinking in Android Studio automatically removes unused resources from your packaged app and library dependencies. For example, if your application is using [Google Play services](https://developers.google.com/android/guides/overview) (https://developers.google.com/android/guides/overview) to access Google Drive functionality, and you are not currently using [Google Sign-In](/training/sign-in) (/training/sign-in), then resource shrinking can remove the various drawable assets for the `SignInButton` buttons.

Note: Resource shrinking works in conjunction with code shrinking tools, such as ProGuard.

For more information on shrinking code and resources, see [Shrink your code and resources](/studio/build/shrink-code) (/studio/build/shrink-code).

Managing dependencies

Dependencies for your project are specified by name in the `build.gradle` file. Gradle takes care of finding your dependencies and making them available in your build. You can declare module dependencies, remote binary dependencies, and local binary dependencies in your `build.gradle` file. Android Studio configures projects to use the Maven Central Repository by default. (This configuration is included in the top-level build file for the project.) For more information about configuring dependencies, read [Add build dependencies](/studio/build/dependencies) (/studio/build/dependencies).

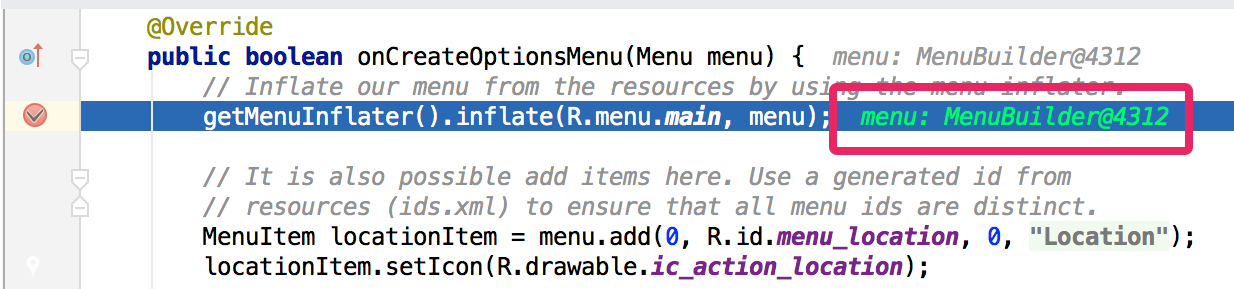
Debug and profile tools

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools.

Inline debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object
- Method return values
- Lambda and operator expressions
- Tooltip values



The screenshot shows a Java code snippet in Android Studio. The code is for the `onCreateOptionsMenu` method. A red box highlights the inline variable `menu: MenuBuilder@4312` in the `getMenuInflater().inflate(R.menu.main, menu);` line. The code is as follows:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate our menu from the resources by using the menu inflater.
    getMenuInflater().inflate(R.menu.main, menu);
    // It is also possible add items here. Use a generated id from
    // resources (ids.xml) to ensure that all menu ids are distinct.
    MenuItem locationItem = menu.add(0, R.id.menu_location, 0, "Location");
    locationItem.setIcon(R.drawable.ic_action_location);
}
```

Figure 6. An inline variable value.

To enable inline debugging, in the **Debug** window, click **Settings**



and select the checkbox for **Show Values Inline**.

Performance profilers

Android Studio provides performance profilers so you can more easily track your app's memory and CPU usage, find deallocated objects, locate memory leaks, optimize graphics performance, and analyze network requests. With your app running on a device or emulator, open the **Android Profiler** tab.

For more information about performance profilers, see [Performance profiling tools](/studio/profile) (/studio/profile).

Heap dump

When you're profiling memory usage in Android Studio, you can simultaneously initiate garbage collection and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks.

For more information about working with heap dumps, see [Inspect the heap and allocations](/studio/profile/memory-profiler) (/studio/profile/memory-profiler).

Memory Profiler

You can use Memory Profiler to track memory allocation and watch where objects are being allocated when you perform certain actions. Knowing these allocations enables you to optimize your app's performance and memory use by adjusting the method calls related to those actions.

For information about tracking and analyzing allocations, see [Inspect the heap and allocations](/studio/profile/memory-profiler) (/studio/profile/memory-profiler).

Data file access

The Android SDK tools, such as [Systrace](/topic/performance/tracing/command-line) (/topic/performance/tracing/command-line), and [logcat](/studio/debug/am-logcat) (/studio/debug/am-logcat), generate performance and debugging data for detailed app analysis.

To view the available generated data files, open the Captures tool window. In the list of the generated files, double-click a file to view the data. Right-click any `.hprof` files to convert them to the standard [Investigate your RAM usage](#) (/studio/profile/investigate-ram) file format.

Code inspections

Whenever you compile your program, Android Studio automatically runs configured [Lint](#) (/studio/write/lint) and other [IDE inspections](#) (<https://www.jetbrains.com/help/idea/2019.1/code-inspection.html>) to help you easily identify and correct problems with the structural quality of your code.

The Lint tool checks your Android project source files for potential bugs and optimization improvements for correctness, security, performance, usability, accessibility, and internationalization.

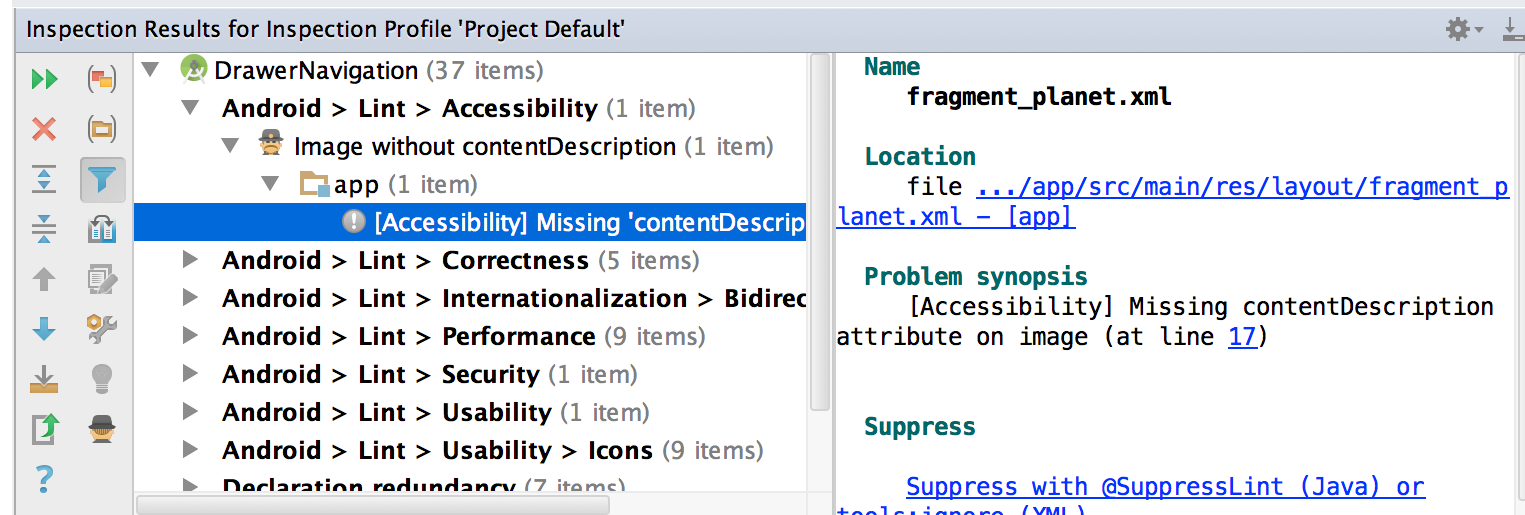


Figure 7. The results of a Lint inspection in Android Studio.

In addition to Lint checks, Android Studio also performs IntelliJ code inspections and validates annotations to streamline your coding workflow.

For more information, see [Improve your code with lint checks](/studio/write/lint) (/studio/write/lint).

Annotations in Android Studio

Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the Support-Annotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection.

For more details about Android annotations, see [Improve code inspection with annotations](/studio/write/annotations) (/studio/write/annotations).

Log messages

When you build and run your app with Android Studio, you can view [adb](/studio/command-line/adb) (/studio/command-line/adb) output and device log messages in the **Logcat window** (/studio/debug/am-logcat).

Performance profiling

If you want to profile your app's CPU, memory, and network performance, open the [Android Profiler](/studio/profile/android-profiler) (/studio/profile/android-profiler), by clicking **View > Tool Windows > Android Profiler**.

Sign in to your developer account

You can sign in to your developer account in Android Studio to access additional tools that requires authentication, such as [Cloud Tools for Android Studio](https://cloud.google.com/tools/android-studio/docs/) and the [App Actions test tool](https://developers.google.com/assistant/app/test-tool). By signing in, you give those tools permission to view and manage your data across Google services.

After you open a project in Android Studio, you can sign in to your developer account or switch developer accounts, as follows:

1. Click the profile icon



at the end of the toolbar, as shown in figure 8.



Figure 8. Click the profile icon at the end of the toolbar to sign in.

2. In the window that appears, do one of the following:

- If you're not yet signed in, click **Sign In** and allow Android Studio to access the listed services.
- If you're already signed in, click **Add Account** to sign in with another Google account. Alternatively, you can click **Sign Out** and repeat the previous steps to sign in to a different account.

Content and code samples on this page are subject to the licenses described in the [Content License \(/license\)](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-05-01.

