Android Facet Page

File | Project Structure | Modules - module - Android facet

Use this page to configure individual settings of an Android facet attached to a specific module.

In this section:

- Common Android Facet options
- Structure Tab
- Compiler Tab

Common Android Facet options

In this area, configure Android platforms to build the application against, choose the platform to use in the current module, and enable or disable sharing the module source code and resources.

Item	Description
Library module	Select this check box to enable using the module as library project so Android application projects can reference its source code and resources.
Reset paths to defaults	Click this button to return to the default Android facet settings.

Structure tab

In this tab, specify the location of the key application components in the module tree structure. Based on these settings, IntelliJ IDEA supports code completion, resolves references, and provides other types of coding assistance.

Item	Description
Manifest file	In this text box, specify the path to the AndroidManifest.xml file. The `AndroidManifest.xml` file should be located in the module root directory.
Resources directory	In this text box, type the path to the folder where the application resources are stored. Resources located in this directory are assigned IDs and can be referenced through the `R.java` file or from XML resource definition files. The default location is `<module root>/res`.
Assets directory	In this text box, type the path to the folder where the application assets are stored. Files located in this directory are not assigned IDs and cannot be referenced through the `R.java` file or from XML resource definition files. You can access this directory like a normal file system and read data from these files using the AssetManager. The default location is `<module root>/assets`.
Native libs directory	In this text box, type the path to the folder where the Android native libraries. . The default location is `<module root>/libs`.

Compiler tab

In this tab, configure the behaviour of various Android compilation tools.

Item	Description
R.java and Manifest.java files	In this area, appoint the folder to store the R.java file with the IDs of all the resources defined in your project and the `AndroidManifest.java` file with permissions. Both files will be generated automatically based on the definitions of resources and the AndroidManifest.xml file respectively. By default, IntelliJ IDEA suggests to store the generated files in the `gen` folder. To change this setting, type the path to the required folder manually in the **Directory for generated files** text box or click the **Browse** button … and select the folder in the dialog box that opens.
AIDL files	In this area, appoint the folder to store the .aidl files generated by the AIDL Compiler. Accept the location suggested by IntelliJ IDEA in the **Directory for generated files** text box, or type the path to another folder manually, or click the **Browse** button … and select the folder in the dialog box that opens.

Item	Description
Resource Packaging	In this area, configure the behaviour of the Android Asset Packaging Tool (aapt) that is responsible for creating an Android Package `.apk` file and specify the location of components to be included in it. - **Use resource directory specified in "Structure" section:** when this option is selected, the compiler takes the resources at the location specified in the **Resources directory** text box of the **Structure** tab. - **Use custom resource directory:** select this option to have the package contain resources from another folder than specified in the **Resources directory** text box of the **Structure** tab. This option is helpful in Maven projects when you need to package the contents of a combined resources directory that contains merged resources from two mutually dependent modules. - **Include assets from dependencies into APK:** select this check box to have assets from dependencies also involved in packaging. - **Rename manifest package:** select this check box to have the Android application package (application ID) changed during build and specify the new name in the text box. You may need to have your application built in several versions which means that several Android application packages (`.apk` files) will be generated. If these files have this single name, the user will be unable to deploy them on the same device simultaneously. To overcome this obstacle, you can have IntelliJ IDEA generate several `.apk` files with different names (*application IDs*) from the same source code. The check box is available only in *Application* modules. - **Enable manifest merging:** select this check box to have manifest files of library modules automatically merged with the manifest of the project that contains this library module. The main goal of declaring a module as *library module* and attaching it to other projects is re-using the components from the library module. To be successfully integrated into another application and function inside it, these components must be presented in the application `AndroidManifest.xml` file. You can either add this information manually or have it extracted from `AndroidManifest.xml` of the library module and added to the `AndroidManifest.xml` of the application automatically. The second approach is referred to as *merging manifests*. The check box is available only in *Application* modules. - **Additional command line parameters:** - In this text box, type the parameters to be passed to the *Android Asset Packaging Tool (aapt)*. For example, to have resources of a certain type included in the `.apk` file uncompressed, type `-0 <file extension for this type of resources>`. As a result, all the files with the specified extension will be excluded from compression. You can specify as many extensions as necessary using commas as separators to suppress compressing numerous resource types. If the set of parameters does not fit the text box, click 🗎 and type the parameters in the dialog box that opens.

Item	Description
ProGuard	In this area, enable the ProGuard tool to obfuscate the application during packaging. - **Run ProGuard:** select this check box to have IntelliJ IDEA obfuscate the debug APK through integration with the ProGuard built-in tool. After you have selected this check box, the **Config file** text box becomes available. Note that for the release APK there is a dedicated check box in the Export signed application package wizard and the Android tab on the **Artifact** page of the **Structure** dialog box. - **Config file**: in this text box, specify the location of the configuration file for *ProGuard*. IntelliJ IDEA fills in the field with the path to the default configuration file `proguard-project.txt` that is created together with the Android module. To change this settings, type the fully-qualified path to the desired file manually or click the **Browse** button … and choose the file in the Select Path dialog box that opens.
APK Path	In this drop-down list, specify the target directory for the `.apk` file, which is obtained as a result of Android module compilation. Choose the APK path from the list or click the **Browse** button … and choose the desired folder in the Select Path dialog box that opens.
Custom debug keystore	In this text box, specify the location of the keystore to use the debug key from. Type path manually or click the **Browse** button … and appoint the location in the dialog that opens. By default, debug keys are stored in <user home>/.android/debug.keystore>. When using builds created on different machines, they all have different signing, so you have to uninstall the existing application before installing the application from another machine. To solve this problem, keystore files are often stored in a version control system alongside the source code. To have IntelliJ IDEA use the repository keystore instead of the default one, specify the relevant location in the **Custom debug keystore** field.
Include test code and resources into APK	Select this check box to include the sources and resources, located under the test roots, into the *debug APK*, which is created in course of project build. (Test data is never included in the release APK, which is created via the Export signed application package wizard. Note that check box is cleared by default this for the regular Android modules; for the test modules this check box is selected by default.

Item	Description
Pre-dex external jars and Android library dependencies	The main goal of declaring a module as *library module* and attaching it to other projects is re-using the components from the library module. During the application packaging, the `.class` files of the library module are converted into `.dex` files, this operation is referred to as *dexing*. Finally the `.dex` files output from the library module are included in the final application `.apk`. Learn more about the building procedure at http://developer.android.com/tools/building/index.html. It often happens, that the contents of a library module remain unchanged. In this case you can have them `dexed` only once whereupon the output `.dex` files are included in `.apk`. This approach is referred to as *pre-dexing*. By default, IntelliJ IDEA pre-dexes library mode dependencies as well as external `jars` that have not been updated since the previous build. You can change this settings so all `.class` files are always dexed. ▪ When the check box is selected, `.dex` files output from `.class` files of library modules or external `.jars` are predexed, that is, they are not dexed anew if the corresponding `.class` files have not ben updated since the previous build. This setting is default for IntelliJ IDEA. ▪ Clear the check box to have all `.class` files dexed during every build. The check box is not available in modules of the type *Library*.

See Also

Procedures:

- Enabling Android Support
- Sharing Android Source Code and Resources Using Library Projects

Reference:

- Android Reference

External Links:

- http://developer.android.com/guide/index.html
- http://developer.android.com/guide/developing/tools/proguard.html

Web Resources:

- Developer Community