Sharing Android Source Code and Resources Using Library Projects

IntelliJ IDEA supports Android library projects that hold shared Android source code and resources. Other Android application projects can reference a library project and include its compiled sources in their `.apk` files at build time.

In IntelliJ IDEA, library projects are supported through separate library modules. To use a library module in another IntelliJ IDEA project, import the required library module in the target project.

To enable sharing Android source code and resources, do one of the following:

- Create a Library module.
- Turn an application module with relevant contents into a library module.

In this topic:

Creating a Library module

1. Do one of the following:
   - Start creating a project from scratch. On the first page of the wizard, select **Android** and select **Library Module**.
   - Start adding a module to an existing project. On the first page of the wizard, select **Android** and select **Library Module**.
2. If you are creating a project from scratch, specify the common project settings.
3. Specify the name of the application and the destination package to store the application classes in.
4. Click **Finish**.

Turning an application module into a library module

An application module as marked as *library module* by updating its *Android facet*.

1. Open the **Project Structure** dialog box by choosing **File | Project Structure** . See Accessing Project Structure and Accessing Module Settings for more information.
2. In the left-hand pane, choose **Modules**, then in the central pane click the **Android** facet under the module whose code and resources you want to share.
3. In the right-hand pane, that shows the Android Facet page, switch to the **Compiler** tab and select the **Library module** check box.

Using a library module in another project

To use a library module in a project you need to import the library module into it.

1. On the main menu, choose **File | New Module**.
2. On the first page of the New Module wizard, choose the **Import existing module** option, and click **Next**.
3. In the dialog that opens, specify the fully qualified name of the module file (`*.iml`), and click **Finish**.
4. Add dependencies on the imported library module to the modules where you want to use its data:
    1. Open the settings of the required non-library module (**File | Project Structure | Modules - required module**).
    2. In the Dependencies tab, click the **Add** button +, then choose **Module Dependency** on the context menu.
    3. In the **Choose Modules** dialog box that opens, select the imported library module from the list.
    4. Click **OK** to return to the Project Structure dialog box. Confirm the added dependency by clicking **OK**.

Adding the data from the AndroidManifest.xml file of a library module to the AndroidManifest.xml file for the entire application

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*.

To have the manifest of a library module merged with the application manifest automatically:

1. Open the **Project Structure** dialog box by choosing **File | Project Structure** . See Accessing Project Structure and Accessing Module Settings for more information.
2. In the left-hand pane, choose **Modules**, then in the central pane click the **Android** facet under the application module with the main launcher activity.
3. In the right-hand pane, that shows the Android Facet page, switch to the **Compiler** tab and select the **Enable manifest merging** check box.

Including the .dex file of a library module into the .apk of the entire application without rebuilding (pre-dexing)

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.

1. Open the **Project Structure** dialog box by choosing **File | Project Structure** . See Accessing Project Structure and Accessing Module Settings for more information.
2. In the left-hand pane, choose **Modules**, then in the central pane click the **Android** facet under the application module with the main launcher activity.
3. In the right-hand pane, that shows the Android Facet page, switch to the **Compiler** tab and enable or disable pre-dexing by selecting or clearing the **Pre-dex external jars and Android library dependencies** check box. By default, the check box is selected.

See Also

Concepts:

- Module
- Dependencies

Procedures:

- Enabling Android Support
- Adding Modules to a Project
- Configuring Module Dependencies and Libraries
- Android

Reference:

- New Project Wizard
- Android Reference

External Links:

- http://developer.android.com/guide/developing/other-ide.html#libraryProject

Web Resources:

- Developer Community