Content Root

Content root is a folder that contains all the files that make up your module.

A module can have more than one content root, however, in most of the cases one content root will suffice. (In certain situations, the modules without content roots may be useful.)

Content roots in IntelliJ IDEA are shown as  or  .

- Source, resource and test roots, and excluded folders
- Modules without content roots: Collections of dependencies
- Package prefix for Java source roots

See also, Configuring Content Roots.

Source, resource and test roots, and excluded folders

Folders within content roots may be assigned to the following categories:

- *Source roots* (or source folders; shown as ). By assigning a folder to this category, you tell IntelliJ IDEA that this folder and its subfolders contain source code that should be compiled as part of the build process.

  In Java modules, the subfolders within the source roots represent your Java package structure.

  You can assign a package prefix to a source root (e.g. `com.mycompany.myapp`) instead of creating the corresponding folder structure within that source root (e.g. `com/mycompany/myapp`). For more information, see Package prefix for Java source roots.

  In the absence of the package prefix and the subfolders, a source root would represent the default package (an unnamed package).

- *Generated source roots* (or generated source folders; shown as  ; in certain views the *[generated]* text marker is used) are similar to source roots. The difference is that the generated source roots are not suggested as target folders when performing the Move Class refactoring or using the Create Class from Usage quick fix.
- *Resource roots* (or resource folders; shown as  ; available only in Java modules) are for resource files used in your application (images, various configuration XML and properties files, etc.).

  During the build process, all the contents of the resource folders are copied to the output folder as is.

- *Test source roots* (or test source folders; shown as ) are similar to source roots but are for code intended for testing (e.g., for unit tests). Test source folders let you keep the code related to testing separate from the production code.

  Compilation results for sources and test sources, normally, are placed into different folders.

- *Generated test source roots* (or generated test source folders; shown as  ; in certain views the *[generated]* text marker is used) are similar to test source roots. The difference is the same as between the source roots and the generated source roots.
- *Test resource roots* (or test resource folders; shown as  ; available only in Java modules) are for resource files associated with your test sources. In all other respects, these folders are similar to resource folders.
- *Excluded folders* (when appropriate, shown as ) are ones that IntelliJ IDEA ignores. You don't see the excluded folders in the Project tool window, cannot, generally, navigate to their contents, etc.

  Normally excluded are the compilation output folders.

Modules without content roots: Collections of dependencies

A module can be used solely as a collection of dependencies for other modules. In such cases, instead of specifying the necessary dependencies separately, you can add the dependency on the corresponding module.

A module used for such purpose, obviously, doesn't need a content root.

Package prefix for Java source roots

A package prefix specifies which Java package corresponds to a folder and can be assigned to the following categories of the Java source roots:

- Sources
- Generated sources
- Test sources
- Generated test sources

If specified, the package prefix acts as an equivalent of the corresponding folder structure which has to be created otherwise.

To illustrate, let's assume you are going to work with the `com.mycompany.myapp` package. In the absence of the package prefix, you would create the folder structure `com/mycompany/myapp` in the corresponding source root folder (e.g. `src`).

The alternative would be to assign `com.mycompany.myapp` to `src` as its package prefix and store the corresponding classes right in `src`.

See Also

Procedures:

- Configuring Content Roots

Reference:

- Project Structure
- Sources Tab
- Paths Tab

Web Resources:

- Developer Community