Scope Language Syntax Reference

The *scopes language* is used in specifying project scopes involved in various kinds of analysis.

In this section:

Sets of classes

- Single class is defined by a class name, i.e. `com.intellij.openapi.MyClass`
- Set of all classes in a package, not recursing into subpackages, is defined by an asterisk after dot, for example: `com.intellij.openapi.*`
- Set of all classes in a package including contents of subpackages, is defined by an asterisk after double dot, for example `com.intellij.openapi..*`

Sets of files

- Single file is defined by a file name, i.e. `MyDir/MyFile.txt`
- Set of all files in a directory, not recursing into subdirectories, is defined by an asterisk after slash, for example: `file:src/main/myDir/*`
- Set of all files in a directory including contents of subdirectories, is defined by an asterisk after double slash, for example `file:src/main/myDir//*`

Modifiers

Location modifiers

help you specify whether the desired set is located in the source files, library classes or test code in the form of location modifiers `src:`, `lib:`, `file:`, or `test:`.

For example, the following scope

```
src:com.intellij.openapi.*
```

implies all classes under the source root in the `com.intellij.openapi` package, excluding subpackages.

The default location is the module root.

Module modifiers

help you narrow down the scope by specifying the name of the related module in one of the following ways:

```
src[module name]:<E>
lib[module name]:<E>
test[module name]:<E>
```

For example, the following scope

```
src[MyModule]:com.intellij.openapi.*
```

implies all classes under the source folders related to the module `MyModule` in the package `com.intellij.openapi`, excluding subpackages.

Group modifier

help you narrow down the scope by specifying the name of the related module group (several modules can be joined into a group in the Project Structure dialog).

The group modifier has the following format:

```
[group:<group name>]
```

For example, the following scope

```
file[group:mygroup]:*//*
```

denotes a scope of all files in the group of modules with the specified name.

Logical operators

The scope language allows you to use common logical operators:

```
 && for AND
 || for OR
 ! for NOT
```

Besides that, the parentheses can be used to join the logical operators into groups. For example, the following scope

```
(<a>||<b>)&&<c>
```

implies either <a> and <c>, or <b> and <c>.

Another example

```
file[*web*]:src/main/java//*
```

denotes a scope of all modules whose name contains `web`, and all the files recursively in the directory `src/main/java`

Defining scopes

Scopes are defined in the Scopes dialog box in the following ways:

- Manually
- With the pointing device

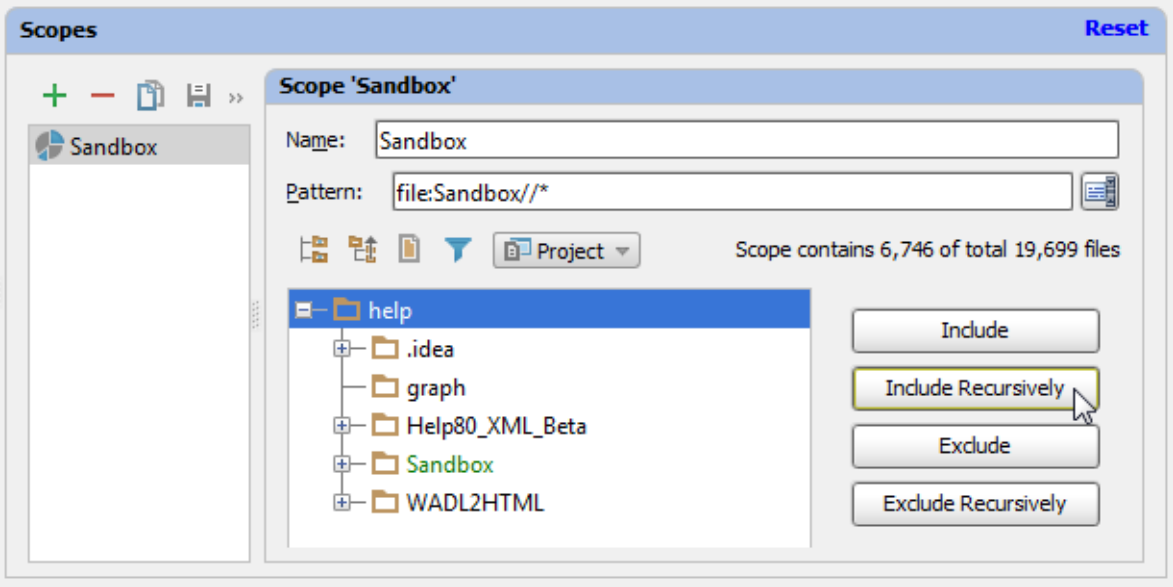
Manually

Specify file masks in the **Pattern** text box, or click  and type the pattern in the editor.

Using mouse

Select files and folders in the project tree view and click the buttons **Include**, **Include Recursively**, **Exclude**, and **Exclude Recursively**. For information about the controls, refer to Scope page description.

Based on the inclusion/exclusion of file and directories, IntelliJ IDEA creates an expression and displays it in the **Pattern** field.



Examples

- `file[MyMod]:src/main/java/com/example/my_package//*` - include in a project all the files from module "MyMod", located in the specified directory and all subdirectories.
- `src[MyMod]:com.example.my_package..*` - recursively include all classes in a package in the source directories of the module.
- `lib:com.company..*||com.company..*` - recursively include all classes in a package from both project and libraries.
- `test:com.company.*` - include all test classes in a package, but not in subpackages.
- `[MyMod]:com.company.util.*` - include all classes and test classes in the package of the specified module.
- `file:*.js||file:*.coffee` - include all JavaScript and CoffeeScript files.
- `file:*js&&!file:*.min.*` - include all JavaScript files except those that were generated through *minification*, which is indicated by the `min` extension.

See Also

Procedures:

- Validating Dependencies
- Analyzing Duplicates

Reference:

- Scopes

Web Resources:

- Developer Community