Language Support for Java(TM) by Red Hat

Red Hat | 16,543,119 installs | (132) | Free

Java Linting, Intellisense, formatting, refactoring, Maven/Gradle support and more...

Install Trouble Installing?

Overview Version History Q & A Rating & Review

Language support for Java ™ for Visual Studio Code

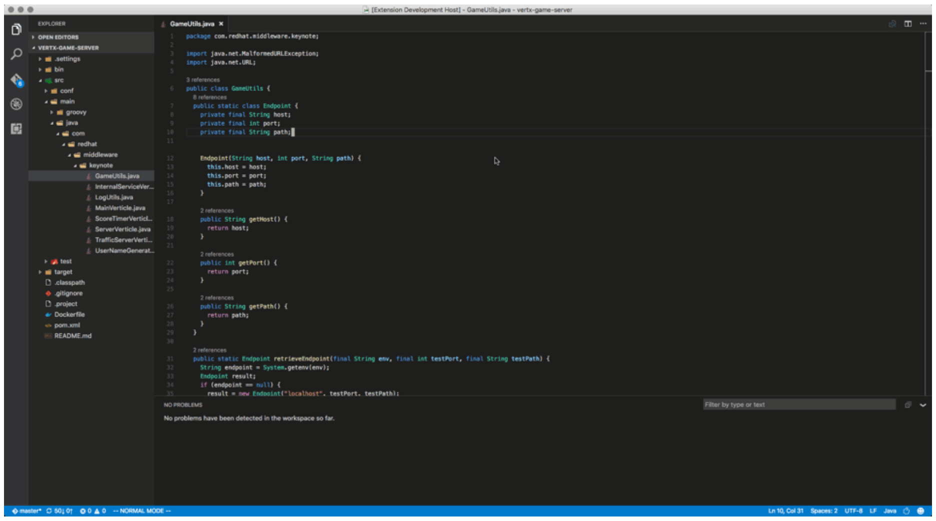
chat on gitter build passing Visual Studio Marketplace v1.8.2022062304

Provides Java ™ language support via Eclipse ™ JDT Language Server, which utilizes Eclipse ™ JDT, M2Eclipse and Buildship.

Quick Start

1. Install the Extension
2. If you do not have a *Java* Development Kit correctly set
   - Download and install a Java Development Kit for your project (Java 1.5 or above is supported)
3. Extension is activated when you first access a Java file
   - Recognizes projects with *Maven* or *Gradle* build files in the directory hierarchy.

Features



```
package com.redhat.middleware.keynote;

import java.net.MalformedURLException;
import java.net.URL;

3 references
public class GameUtils {
  8 references
  public static class Endpoint {
    private final String host;
    private final int port;
    private final String path;


    Endpoint(String host, int port, String path) {
      this.host = host;
      this.port = port;
      this.path = path;
    }

    2 references
    public String getHost() {
      return host;
    }

    2 references
    public int getPort() {
      return port;
    }

    2 references
    public String getPath() {
      return path;
    }
  }

  2 references
  public static Endpoint retrieveEndpoint(final String env, final int testPort, final String testPath) {
    String endpoint = System.getenv(env);
    Endpoint result;
    if (endpoint == null) {
      result = new Endpoint("localhost", testPort, testPath);
```

No problems have been detected in the workspace so far.

- Supports code from Java 1.5 to Java 18
- Maven pom.xml project support
- Basic Gradle Java project support (Android not supported)
- Standalone Java files support
- As-you-type reporting of parsing and compilation errors
- Code completion
- Code/Source actions / Refactoring
- Javadoc hovers
- Organize imports
  - triggered manually or on save
  - when pasting code into a java file with `Ctrl+Shift+v` (`Cmd+Shift+v` on Mac).
- Type search
- Code outline
- Code folding
- Code navigation

Categories

Programming Languages Snippets Linters Formatters

Tags

java json keybindings linters multi-root ready snippet

Works with

Linux x64, Windows x64, Linux ARM64, Alpine Linux ARM64, Windows ARM, Alpine Linux 64 bit, macOS Intel, Windows ia32, macOS Apple Silicon, Linux ARM32

Resources

Issues
Repository
Homepage
License
Changelog

Download Extension

Project Details

redhat-developer/vscode-java
Last Commit: 7 hours ago
16 Pull Requests
687 Open Issues

More Info

Version	1.8.2022062304
Released on	9/12/2016, 4:41:27 PM
Last updated	6/23/2022, 4:28:15 AM
Publisher	Red Hat
Unique Identifier	redhat.java
Report	Report Abuse

- Code lens (references/implementations)
- Highlights
- Code formatting (on-type/selection/file)
- Code snippets
- Annotation processing support (automatic for Maven projects)
- Semantic selection
- Diagnostic tags
- Call Hierarchy
- Type Hierarchy

Please note that Gradle-based Android projects are not supported.

To launch and debug your Java programs, it's recommended you install *Java Debug Extension for Visual Studio Code*.

See the changelog for the latest release. You might also find useful information in the project Wiki.

Setting the JDK

Java Tooling JDK

Now that Java extension will publish platform specific versions, it will embed a JRE for supported platforms such as `win32-x64`, `linux-x64`, `linux-arm64`, `darwin-x64`, `darwin-arm64`. The embedded JRE is used to launch the Language Server for Java. Users are only responsible for configuring Project JDKs to compile your Java projects.

The following part is only kept for the universal version without embedded JRE.

> The tooling JDK will be used to launch the Language Server for Java. And by default, will also be used to compile your projects.
>
> The path to the Java Development Kit can be specified by the `java.jdt.ls.java.home` setting in VS Code settings (workspace/user settings). If not specified, it is searched in the following order until a JDK meets current minimum requirement.
>
> - the `JDK_HOME` environment variable
> - the `JAVA_HOME` environment variable
> - on the current system path

Project JDKs

If you need to compile your projects against a different JDK version, it's recommended you configure the `java.configuration.runtimes` property in your user settings, eg:

```
"java.configuration.runtimes": [
  {
    "name": "JavaSE-1.8",
    "path": "/path/to/jdk-8",
  },
  {
    "name": "JavaSE-11",
    "path": "/path/to/jdk-11",
  },
  {
    "name": "JavaSE-18",
    "path": "/path/to/jdk-18",
    "default": true
  },
]
```

The default runtime will be used when you open standalone Java files.

Available commands

The following commands are available:

- `Switch to Standard Mode`: switches the Java Language Server to `Standard` mode. This command is only available when the Java Language Server is in `LightWeight` mode.
- `Java: Update Project` (`Shift+Alt+U`): is available when the editor is focused on a Maven pom.xml or a Gradle file. It forces project configuration / classpath updates (eg. dependency changes or Java compilation level), according to the project build descriptor.

- `Java: Import Java Projects into Workspace`: detects and imports all the Java projects into the Java Language Server workspace.
- `Java: Open Java Language Server Log File`: opens the Java Language Server log file, useful for troubleshooting problems.
- `Java: Open Java Extension Log File`: opens the Java extension log file, useful for troubleshooting problems.
- `Java: Open All Log Files`: opens both the Java Language Server log file and the Java extension log file.
- `Java: Force Java Compilation` (`Shift+Alt+B`): manually triggers compilation of the workspace.
- `Java: Open Java Formatter Settings`: opens the Eclipse formatter settings. Creates a new settings file if none exists.
- `Java: Clean Java Language Server Workspace`: cleans the Java language server workspace.
- `Java: Attach Source`: attaches a jar/zip source to the currently opened binary class file. This command is only available in the editor context menu.
- `Java: Add Folder to Java Source Path`: adds the selected folder to its project source path. This command is only available in the file explorer context menu and only works for unmanaged folders.
- `Java: Remove Folder from Java Source Path`: removes the selected folder from its project source path. This command is only available in the file explorer context menu and only works for unmanaged folders.
- `Java: List All Java Source Paths`: lists all the Java source paths recognized by the Java Language Server workspace.
- `Java: Show Build Job Status`: shows the Java Language Server job status in Visual Studio Code terminal.
- `Java: Go to Super Implementation`: goes to the super implementation for the current selected symbol in editor.

Supported VS Code settings

The following settings are supported:

- `java.home` : **Deprecated, please use 'java.jdt.ls.java.home' instead.** Absolute path to JDK home folder used to launch the Java Language Server. Requires VS Code restart.

- `java.jdt.ls.vmargs` : Extra VM arguments used to launch the Java Language Server. Requires VS Code restart.

- `java.errors.incompleteClasspath.severity` : Specifies the severity of the message when the classpath is incomplete for a Java file. Supported values are `ignore`, `info`, `warning`, `error`.

- `java.trace.server` : Traces the communication between VS Code and the Java language server.

- `java.configuration.updateBuildConfiguration` : Specifies how modifications on build files update the Java classpath/configuration. Supported values are `disabled` (nothing happens), `interactive` (asks about updating on every modification), `automatic` (updating is automatically triggered).

- `java.configuration.maven.userSettings` : Path to Maven's user settings.xml.

- `java.configuration.checkProjectSettingsExclusions`: **Deprecated, please use 'java.import.generatesMetadataFilesAtProjectRoot' to control whether to generate the project metadata files at the project root. And use 'files.exclude' to control whether to hide the project metadata files from the file explorer.** Controls whether to exclude extension-generated project settings files (`.project`, `.classpath`, `.factorypath`, `.settings/`) from the file explorer. Defaults to `false`.

- `java.referencesCodeLens.enabled` : Enable/disable the references code lenses.

- `java.implementationsCodeLens.enabled` : Enable/disable the implementations code lenses.

- `java.signatureHelp.enabled` : Enable/disable signature help support (triggered on `(`).

- `java.signatureHelp.description.enabled` : Enable/disable to show the description in signature help. Defaults to `false`.

- `java.contentProvider.preferred` : Preferred content provider (see 3rd party decompilers available in vscode-java-decompiler).

- `java.import.exclusions` : Exclude folders from import via glob patterns. Use `!` to negate patterns to allow subfolders imports. You have to include a parent directory. The order is important.

- `java.import.gradle.enabled` : Enable/disable the Gradle importer.

- Specify the Gradle distribution used by the Java extension:

  - `java.import.gradle.wrapper.enabled`: Use Gradle from the 'gradle-wrapper.properties' file. Defaults to `true`.

    - `java.import.gradle.version`: Use Gradle from the specific version if the Gradle wrapper is missing or disabled.
    - `java.import.gradle.home`: Use Gradle from the specified local installation directory or GRADLE_HOME if the Gradle wrapper is missing or disabled and no 'java.import.gradle.version' is specified.
- `java.import.gradle.arguments`: Arguments to pass to Gradle.
- `java.import.gradle.jvmArguments`: JVM arguments to pass to Gradle.
- `java.import.gradle.user.home`: setting for GRADLE_USER_HOME.
- `java.import.gradle.offline.enabled`: Enable/disable the Gradle offline mode. Defaults to `false`.
- `java.import.maven.enabled` : Enable/disable the Maven importer.
- `java.autobuild.enabled` : Enable/disable the 'auto build'.
- `java.maxConcurrentBuilds`: Set max simultaneous project builds.
- `java.completion.enabled` : Enable/disable code completion support.
- `java.completion.guessMethodArguments` : When set to true, method arguments are guessed when a method is selected from as list of code assist proposals.
- `java.completion.filteredTypes`: Defines the type filters. All types whose fully qualified name matches the selected filter strings will be ignored in content assist or quick fix proposals and when organizing imports. For example 'java.awt.*' will hide all types from the awt packages.
- `java.completion.favoriteStaticMembers` : Defines a list of static members or types with static members.
- `java.completion.importOrder` : Defines the sorting order of import statements.
- `java.progressReports.enabled` : [Experimental] Enable/disable progress reports from background processes on the server.
- `java.format.enabled` : Enable/disable the default Java formatter.
- `java.format.settings.url` : Specifies the url or file path to the Eclipse formatter xml settings.
- `java.format.settings.profile` : Optional formatter profile name from the Eclipse formatter settings.
- `java.format.comments.enabled` : Includes the comments during code formatting.
- `java.format.onType.enabled` : Enable/disable on-type formatting (triggered on `;`, `}` or `<return>`).
- `java.foldingRange.enabled`: Enable/disable smart folding range support. If disabled, it will use the default indentation-based folding range provided by VS Code.
- `java.maven.downloadSources`: Enable/disable download of Maven source artifacts as part of importing Maven projects.
- `java.maven.updateSnapshots`: Force update of Snapshots/Releases. Defaults to `false`.
- `java.codeGeneration.hashCodeEquals.useInstanceof`: Use 'instanceof' to compare types when generating the hashCode and equals methods. Defaults to `false`.
- `java.codeGeneration.hashCodeEquals.useJava7Objects`: Use Objects.hash and Objects.equals when generating the hashCode and equals methods. This setting only applies to Java 7 and higher. Defaults to `false`.
- `java.codeGeneration.useBlocks`: Use blocks in 'if' statements when generating the methods. Defaults to `false`.
- `java.codeGeneration.generateComments`: Generate method comments when generating the methods. Defaults to `false`.
- `java.codeGeneration.toString.template`: The template for generating the toString method. Defaults to `${object.className} [${member.name()}=${member.value}, ${otherMembers}]`.
- `java.codeGeneration.toString.codeStyle`: The code style for generating the toString method. Defaults to `STRING_CONCATENATION`.
- `java.codeGeneration.toString.skipNullValues`: Skip null values when generating the toString method. Defaults to `false`.
- `java.codeGeneration.toString.listArrayContents`: List contents of arrays instead of using native toString(). Defaults to `true`.

- `java.codeGeneration.toString.limitElements`: Limit number of items in arrays/collections/maps to list, if 0 then list all. Defaults to `0`.
- `java.selectionRange.enabled`: Enable/disable Smart Selection support for Java. Disabling this option will not affect the VS Code built-in word-based and bracket-based smart selection.
- `java.showBuildStatusOnStart.enabled`: Automatically show build status on startup, defaults to `notification`.
  - `notification`: Show the build status via progress notification.
  - `terminal`: Show the build status via terminal.
  - `off`: Do not show any build status.

  > For backward compatibility, this setting also accepts boolean value, where `true` has the same meaning as `notification` and `false` has the same meaning as `off`.
- `java.project.outputPath`: A relative path to the workspace where stores the compiled output. `Only` effective in the `WORKSPACE` scope. The setting will `NOT` affect Maven or Gradle project.
- `java.project.referencedLibraries`: Configure glob patterns for referencing local libraries to a Java project.
- `java.completion.maxResults`: Maximum number of completion results (not including snippets). `0` (the default value) disables the limit, all results are returned. In case of performance problems, consider setting a sensible limit..
- `java.configuration.runtimes`: Map Java Execution Environments to local JDKs.
- `java.server.launchMode`:
  - `Standard`: Provides full features such as intellisense, refactoring, building, Maven/Gradle support etc...
  - `LightWeight`: Starts a syntax server with lower start-up cost. Only provides syntax features such as outline, navigation, javadoc, syntax errors. The lightweight mode won't load thirdparty extensions, such as java test runner, java debugger, etc.
  - `Hybrid`: Provides full features with better responsiveness. It starts a standard language server and a secondary syntax server. The syntax server provides syntax features until the standard server is ready. And the syntax server will be shutdown automatically after the standard server is fully ready.

  Default launch mode is `Hybrid`. Legacy mode is `Standard`
- `java.sources.organizeImports.starThreshold`: Specifies the number of imports added before a star-import declaration is used, default is 99.
- `java.sources.organizeImports.staticStarThreshold`: Specifies the number of static imports added before a star-import declaration is used, default is 99.
- `java.imports.gradle.wrapper.checksums`: Defines allowed/disallowed SHA-256 checksums of Gradle Wrappers.
- `java.project.importOnFirstTimeStartup`: Specifies whether to import the Java projects, when opening the folder in Hybrid mode for the first time. Supported values are `disabled` (never imports), `interactive` (asks to import or not), `automatic` (always imports). Default to `automatic`.
- `java.project.importHint`: Enable/disable the server-mode switch information, when Java projects import is skipped on startup. Defaults to `true`.
- `java.import.gradle.java.home`: Specifies the location to the JVM used to run the Gradle daemon.
- `java.project.resourceFilters`: Excludes files and folders from being refreshed by the Java Language Server, which can improve the overall performance. For example, ["node_modules",".git"] will exclude all files and folders named 'node_modules' or '.git'. Defaults to ["node_modules",".git"].
- `java.templates.fileHeader`: Specifies the file header comment for new Java file. Supports configuring multi-line comments with an array of strings, and using ${variable} to reference the predefined variables.
- `java.templates.typeComment`: Specifies the type comment for new Java type. Supports configuring multi-line comments with an array of strings, and using ${variable} to reference the predefined variables.
- `java.references.includeAccessors`: Include getter, setter and builder/constructor when finding references. Default to true.
- `java.configuration.maven.globalSettings` : Path to Maven's global settings.xml.
- `java.eclipse.downloadSources` : Enable/disable download of Maven source artifacts for Eclipse projects.

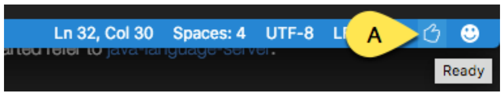
- `java.recommendations.dependency.analytics.show` : Show the recommended Dependency Analytics extension.
- `java.references.includeDecompiledSources` : Include the decompiled sources when finding references. Default to true.
- `java.project.sourcePaths`: Relative paths to the workspace where stores the source files. `Only` effective in the `WORKSPACE` scope. The setting will `NOT` affect Maven or Gradle project.
- `java.typeHierarchy.lazyLoad`: Enable/disable lazy loading the content in type hierarchy. Lazy loading could save a lot of loading time but every type should be expanded manually to load its content.
- `java.codeGeneration.insertionLocation`: Specifies the insertion location of the code generated by source actions. Defaults to `afterCursor`.
  - `afterCursor`: Insert the generated code after the member where the cursor is located.
  - `beforeCursor`: Insert the generated code before the member where the cursor is located.
  - `lastMember`: Insert the generated code as the last member of the target type.
- `java.settings.url` : Specifies the url or file path to the workspace Java settings. See Setting Global Preferences
- `java.symbols.includeSourceMethodDeclarations` : Include method declarations from source files in symbol search. Defaults to `false`.
- `java.quickfix.showAt` : Show quickfixes at the problem or line level.
- `java.configuration.workspaceCacheLimit` : The number of days (if enabled) to keep unused workspace cache data. Beyond this limit, cached workspace data may be removed.
- `java.import.generatesMetadataFilesAtProjectRoot` : Specify whether the project metadata files(.project, .classpath, .factorypath, .settings/) will be generated at the project root. Defaults to `false`.
- `java.inlayHints.parameterNames.enabled`: Enable/disable inlay hints for parameter names. Supported values are: `none`(disable parameter name hints), `literals`(Enable parameter name hints only for literal arguments) and `all`(Enable parameter name hints for literal and non-literal arguments). Defaults to `literals`.

Semantic Highlighting

Semantic Highlighting fixes numerous syntax highlighting issues with the default Java Textmate grammar. However, you might experience a few minor issues, particularly a delay when it kicks in, as it needs to be computed by the Java Language server, when opening a new file or when typing. Semantic highlighting can be disabled for all languages using the `editor.semanticHighlighting.enabled` setting, or for Java only using language-specific editor settings.

Troubleshooting

1. Check the status of the language tools on the lower right corner (marked with A on image below). It should show ready (thumbs up) as on the image below. You can click on the status and open the language tool logs for further information in case of a failure.



2. Read the troubleshooting guide for collecting informations about issues you might encounter.
3. Report any problems you face to the project.

Contributing

This is an open source project open to anyone. Contributions are extremely welcome!

For information on getting started, refer to the CONTRIBUTING instructions.

Continuous Integration builds can be installed from http://download.jboss.org/jbosstools/jdt.ls/staging/. Download the most recent `java-<version>.vsix` file and install it by following the instructions here. Stable releases are archived under http://download.jboss.org/jbosstools/static/jdt.ls/stable/.

Also, you can contribute your own VSCode extension to enhance the existing features by following the instructions here.

Feedback

- Have a question? Start a discussion on GitHub Discussions,
- File a bug in GitHub Issues,
- Chat with us on Gitter,
- Tweet us with other feedback.

License