

CONTENTS

- › What Is Eclipse IDE?
- › Download and Installation
- › Extensions
- › Productivity Tips
- › Customization... and more!

Eclipse

UPDATED BY MICKAEL ISTRIA & ILYA BUZIU

WHAT IS ECLIPSE IDE?

Eclipse IDE is a cross-platform, multi-purpose, open-source Integrated Development Environment. It was initiated as Java IDE, but quickly became an extensible multi-language development tool that is widely used to develop projects in Java, JavaScript, PHP, C++, Scala, and a lot of other languages. It's highly extensible and supports the vast majority of industrial technologies.

DOWNLOAD, INSTALLATION & GETTING STARTED

The only prerequisite for installation is to have a **Java Virtual Machine** installed (such as OpenJDK or Oracle JVM). To make sure you're ready, just type `java -version` in a terminal, and see that a recent version (`>= 1.8.0`) is installed and that the target architecture (`x86`, `x86_64`...) matches your computer and OS.

You can download Eclipse from eclipse.org/downloads. There is a generic installer, which is recommended. The installer will guide you through installation. Alternatively, you can download one package dedicated to your use case directly. In this case, installation only requires extracting the archive and running the executable.

WORKSPACE 101

When starting the IDE, you're prompted to select a **workspace**. The state of your IDE (directories visible as projects, preferences, workbench arrangement, etc.) is stored in the workspace location. Your projects and code can either be stored in the workspace location or in another directory. The workspace location is often the default directory when creating new projects, but this can always be changed.

In Eclipse, you can simultaneously work on multiple Projects. The term Project in Eclipse can refer to a standalone project, or to a project "module" that has dependency or parency with some other projects in the IDE. You can simultaneously work on totally independent projects, using different technologies inside the same IDE and workspace. A project stores its own configuration in various metadata files, such as `.project`, `.classpath` or `.settings` files. Those project configuration files can usually be shared on an SCM together with the project.

Since the projects in a workspace are often linked one another, sometimes it's best to make sure to clear the **Project** → **Build Automatically** option to stop the project from automatically cascading builds to dependent projects.

WORKBENCH 101

The main user interface of Eclipse is called the **Workbench**. It is highly customizable. The Workbench can contain several **Perspectives**. A perspective is an arrangement of the various available panels of the workbench dedicated to specific activities (debugging, coding, memory analysis, etc.). The panels in the Perspectives are called **Views**. You can switch between perspectives on the right part of the toolbar or via the **Window** → **Perspective** menu. You can also open new preset perspectives that are meant to be optimized for certain use-cases.

The perspectives can be customized through the **Window** → **Customize Perspective...** menu, which allows you to show or hide content from toolbars, the menu, and the (right-click) context menu. As you get familiar with Eclipse, you should spend some time filtering the actions you're interested in.

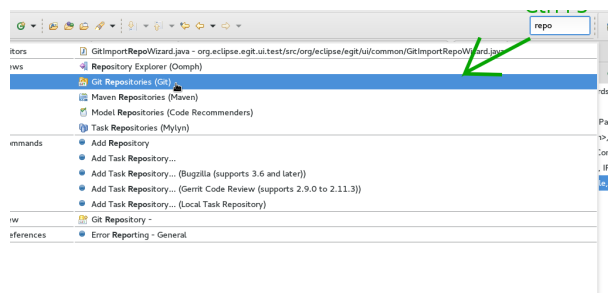
There are views for many things; each view shows an interesting functionality of the IDE. You can browse and open the available view with **Window** → **Show View** (`Alt+Shift+Q` `Q` on Windows; `Cmd+Alt+Q` `Q` on Mac). Each view has its own configuration and settings that you can manage.

If you're about to work with modular projects, we recommend to browse them using the **Project Explorer** view (`Alt+Shift+Q` `Q` or `Cmd+Alt+Q` `Q`, then select **Project Explorer**), and then to get to the View Menu on the top-right of the view and select **Project Layout** → **Hierarchical**. Other options such as **Link With Editor** are available.

The Central part of the workbench is the placeholder for file Editors. When you open a file from the Project Explorer, the IDE will show an editor for that file in this Editor section.

QUICK-ACCESS, CTRL+3, ONE ENTRY-POINT TO RULE THEM ALL

Eclipse has a very powerful **Quick Access** tool towards the top-right part of the workbench (click the text box or use `Ctrl+3` or `Cmd+3` to activate). You can type in a keyword and see all possible related actions. So whenever there's something you want to do, use Quick Access and type a keyword. Most of the time, you'll find what you need!



**RED HAT
DEVELOPERS**

Learn more. Code more. Share more.
 Downloads, cheat sheets and more with
 Red Hat Developers.

Learn more at developers.redhat.com

Take **ECLIPSE** BEYOND your **IDE**

Writing code with Eclipse can be a huge time-saver, but sometimes tough problems can still get in your way. That's when it's time to reach out for a helping hand. As a part of Red Hat Developers you can find answers, try your hand at new solutions, and discover technologies to help you get the job done faster.

Learn more. Code more. Share more.
developers.redhat.com



RED HAT
DEVELOPERS




redhat.


EXTENSIONS

The Eclipse IDE is relying on the Eclipse RCP platform for Rich Applications. This platform is extremely extensible. So the Eclipse IDE is only the beginning of what you can do—many extensions for it are available from multiple sources.


SIMULTANEOUS RELEASE

The “official” Eclipse Community delivers regularly what’s called the *Simultaneous Release*. A number of Eclipse projects conforming to some quality commitment are aggregated in a single extension repository that you can access directly from your IDE. Go to **Help** →  **Install New Software** and look at the available sources. You should get a source called *Neon* (2016-2016) or *Oxygen* (2016-2017). Select this source, and you’ll be able to browse all the extensions offered as part of this simultaneous release.

MARKETPLACE

Eclipse Marketplace (marketplace.eclipse.org) and the Marketplace Client in the Eclipse IDE (**Help** →  **Eclipse Marketplace...**) are the most user-friendly ways to browse and install Eclipse extensions from Eclipse.org or a 3rd party.

INSTALL SITE

Some extensions are not in the Marketplace. Those extensions usually provide a URL that references an update-site, or p2 repository (p2 is the package manager for Eclipse plugins). You can browse and install their content via the **Help** →  **Install New Software...** menu, using the *Add Location* button to add the provided URL as a Source for finding new extensions.

CREATE YOUR OWN EXTENSIONS

You can also write your own Eclipse extensions, or build your own Eclipse RCP-based applications. Although this Refcard doesn’t cover the development of Eclipse plugins, the starting point is to install **Plugin Development Environment (PDE)** from the simultaneous release repo and create a new plugin project. See eclipse.org/pde for more details.

CUSTOMIZATION WITH PREFERENCES

As you’re probably going to spend plenty of time using the Eclipse IDE, it’s worth spending some time looking at the customization possibilities. Open the **Window** → **Preferences** menu, or type *Preferences* in *Quick-Access/Ctrl+3*. Then you can browse available preferences. Changing preferences there apply to the whole workspace. We’ll discuss a few of them here.

SHORTCUTS MAP

The **General** → **Keys** preference page shows all available commands and the shortcuts assigned to each of them. You can edit the shortcuts or switch between different schemes. For example, an Emacs keybinding is available out-of-the-box, and some extensions provide other bindings, such as a Vim-based binding.

UI THEME & FONTS

The default Eclipse IDE theme is a light one, but a dark one is provided if you prefer. From this **General** → **Appearance** preference page, just select the **Dark** theme in the combo, apply and restart the IDE.

Something that can also be convenient to configure, mainly for accessibility purposes, are the fonts and colors used by the IDE. The settings from the Theme can be overridden from the **General** → **Appearance** → **Colors and Fonts** preference page.

If you feel creative, a Marketplace extension called *Jeeyul's Eclipse Themes* adds some more colorful themes for the Eclipse IDE and allows you to customize themes in more detail.

PROJECT PROPERTIES

Projects have specific preferences. You can access them via the **Properties** context-menu on a project or by using the **Alt+Enter** shortcut.

There’s usually a **Save Actions** page in the project and workspace preferences. This allows to configure things such as Code-Style, some automatic refactorings, or other useful operations to trigger when saving a file.

Project Properties also allow you to configure validation, compilation, and everything else for controlling your project behavior in Eclipse.

PRODUCTIVITY TIPS

Have a look at the content of **Help** → **Tips & Tricks** when landing in the Eclipse IDE for the first time, or after an upgrade of the IDE.

Learn some good shortcuts. As explained above, you can see and change these via the Preferences. Here are a few good ones, in addition to the usual system shortcuts such as copy/paste.

SHORTCUT (WIN/LINUX)	SHORTCUT (OSX)	DESCRIPTION
Ctrl+3	⌘+3	Open quick-search allowing you to find any action available
Ctrl+Shift+R	⌘+Shift+R	Open a file/resource in workspace
Ctrl+1	⌘+1	Quick-fix and Refactoring. When in an editor, this shortcut on a piece of code should provide you with the interesting operations to apply at this location.
Ctrl+Space	⌘+Space	Toggle/loop on completion
Ctrl+D	⌘+D	Delete lines
Alt+Up/Down	⌘+Up/Down	Move lines up/down
Alt+Shift+A	⌘+⌘+A	Toggle block/multi-line selection and editing
Ctrl++/Ctrl+-	⌘+=/⌘+-	Zoom in/out on text editor
Ctrl+B	⌘+B	Build All
F3	F3	In code editor, go to declaration
Ctrl+Left Click	⌘+Left Click	In code editor, go to declaration
Alt+Shift+Q Q	⌘+⌘+Q Q	Show view
Ctrl+F	⌘+F	Search/replace in current file
Ctrl+H	⌘+H	Search/replace in current resource, project, or workspace

NEW PROJECTS AND FILES

Eclipse lets you create a large variety of projects and files. The actual list depends on which extensions are installed. So if you want to create a project for some technology, make sure the necessary extensions are installed in order to have the best *New Wizard* possible.

Then it becomes only a matter of picking the best wizard in the list you see when doing **File** → **New** → **Project...** or using the **Ctrl+N** shortcut (or typing *New Project* in *Quick Assist/Ctrl+3*).

Creating new files is also achieved with **File** → **New** → **Others...**

Many of those wizards offer an optional “template” page that can generate a sample project to start more efficiently in your


development. It's recommended that you always have a look at those, and prefer hitting *Next* > rather than *Finish* on wizards to take advantage of the examples and templates.

SCMS

Source Control Managers are usually accessible via the **File** → **Import...** wizard, and once you have a project in a workspace, by the **Team** context-menu on the project. Those 2 entry-points should lead you to every SCM option possible.

GIT

To **clone** a Git repository, show the **Git Repository** view (*Windows* → *Show View/Alt+Shift+Q Q* → *Git Repositories*), or use *Quick-Access/Ctrl+3* and type *Clone*.

Files from a Git repository get decorated with a  in your *Project Explorer* to show whether they're staged for commit or not (prefixed with >). The *Team* context menu offers all useful operation such as **Add to Index** (git add), **Compare, Replace With** (git reset --), **Show history**, **Show annotations** (git blame), and more.

There are 3 very useful views to manage everything in Git:

- the **Git Repositories** view (see above) to view and manage Git repositories in Eclipse.
- The **Git Staging** view allows you to always see and manage the content of your commit. It shows data similar to Git status, and you can drag and drop files between *Staged/Unstaged* to include them or not in your commit; you can prepare a commit message, amend a previous commit, then *Commit and Push* in the same view.
- The **History** view (open when selecting the **Team** → **Show In History** context menu) allows to view history of the project or file (scope can be configured on the top-right buttons of the view), and to easily reset, revert, checkout, compare with... a previous revision.

You can push a commit by selecting the **Team** → **Push...** context-menu on a Project, or via the **Push** context-menu in the *Git Repository* view, the **Commit and Push...** button of the *Git Staging* view, or *Quick Access/Ctrl+3*.

IMPORT EXISTING PROJECTS

FILE → OPEN PROJECTS...

In case you don't know a better way to import your project, you can give a try to the **File** → **Open projects...** wizard. This wizard will take the location you want to work with and will run some analysis to find a good configuration for the project to take advantage of the relevant IDE features.

SPECIALIZED IMPORT WIZARDS

Eclipse also provides some wizards dedicated to specific project types. If you already know some things about the project, and if you're not satisfied with the generic *Open projects...* wizard, then you can go to **File** → **Import...** and find whether there is a dedicated wizard that matches your project.

JAVA DEVELOPMENT 101

NEW JAVA PROJECT

From the *New Wizard* (accessible via **File** → **New** → **Projects...** or **Ctrl+N** or *Quick Access*), you can reach multiple wizards to create projects. The 3 main ones for Java development are:

- Maven project:** Generates a Maven project, creating the pom.xml, the project structure. A page of the wizard allows you to select archetype (project templates/examples). It's often useful to spend time considering the best archetype to use.
- Dynamic Web projects:** This is actually the entry point for a Servlet-based project, usually packaged as a .war and deployed to an application server.
- Plain Java Project:** A simple Java project, suitable for standalone Java libraries or applications. The generated project won't include support for a build system or frameworks.

Once your Java Project is created, you can right-click on it to perform many operations, such as creating a new class, a new interface, a new JUnit test, or some specific classes (Beans, Jax-RS) for certain projects.

JAVA PROJECT SETTINGS

On a project, try the **Properties** context menu (**Alt+Enter**). From there you can tweak many things, such as error reporting, classpath, Java compliance, and more.

NAVIGATION (SHORTCUTS)

Here are the main shortcuts for Java navigation:

SHORTCUT (WIN/LINUX)	SHORTCUT (OSX)	DESCRIPTION
F2	F2	Show Javadoc for selected element
F3	F3	Go to declaration
F4	F4	Show type hierarchy
Ctrl+Shift+T	⌘+Shift+T	Open a type (class, interface, enum)
Ctrl+Click	⌘+Click	Go to... (simply Ctrl+hover shows multiple suggestions when useful)
Alt+Enter	⌘+I	Open Project Properties

As always, many more shortcuts are available in the *General* → *Keys* page of the *Preferences*.

AUTOMATIC BUILD AND ERROR REPORTING

By default, Eclipse is configured to **Build Automatically** (in the *Project* menu, *Build Automatically* is ticked). This enables various analyzers on the project and reports errors and problems directly in the code.

You can at any time force a full build by going to **Project** → **Build All** menu (**Ctrl+B**), or typing "Build All" in *Quick-Assist/Ctrl+3*. It may be useful to sometimes run **Project** → **Clean...** to clean the various caches and work folders if you suspect those are causing some trouble.

Eclipse usually provides resolutions for the errors it reports. Those are named **Quick-Fix** and you can view them by hitting **Ctrl+1** when selection is on a problem. The severity of most problems can be configured in the *Project Properties*.

Some Eclipse extensions such as the **FindBugs** plugin can provide additional error reporting to improve the quality of your code.

At any time, the **Problems** view (*Window* → *Show View/Alt+Shift+Q Q* or "Problems" in *Quick-Access/Ctrl+3*) lists all detected problems in all your projects. Try to fix all issues reported in that view for good productivity and good code quality.

SHORTCUT (WIN/LINUX)	SHORTCUT (OSX)	DESCRIPTION
Ctrl+B	⌘+B	Build All
Ctrl+1	⌘+1	(on an error) Available quick fix

REFACTORING AND CODE FORMATTING

Eclipse comes with a lot of refactoring and other advanced editing operations. All are accessible by right-click under the **Source** (Shift+Alt+S) and **Refactoring** (Shift+Alt+R) context menus. The available operations depend on the current selection. The most common refactorings also have direct shortcuts, and some of them for the current selection are directly shown with **Quick-Assist**/Ctrl+2. As usual, all those operations can also be reached simply by **Quick-Access**/Ctrl+3. Examples of the most useful refactorings include: *Rename*, *Convert Local Variable to Field*, *Extract Method*, *Extract Interface*...


SHORTCUT (WIN/LINUX)	SHORTCUT (OSX)	DESCRIPTION
Shift+Alt+S	⌘+⇧+S	Show advanced editing operations for current selection
Shift+Alt+T	⌘+⇧+T	Show refactorings for current selection
Ctrl+2	⌘+2	Quick-Assist: most usual refactorings for current selection
Ctrl+Shift+C	⌘+⇧+/	Comment selected lines
Shift+Alt+R	⌘+⇧+R	Rename (variable, field, method, class...)

RUN

The **Run As** context-menu on a Java element is populated by what seems to be the best **Run Configurations** for your current selection. Just select one and it will run it.

It's often useful to customize the pre-existing *Run Configurations* or even create your own. Customizing a *Run Configuration* allows to more easily set additional System Properties, Environment Variables, Java settings, etc, that will be used at runtime. Once you have a Run Configuration ready, it is stored and can be run as many times as you want.

The entry points to tweak Run Configurations are:

- **Run As** → **Run Configurations...** context-menu
- *Run Configurations...* under the  toolbar button
- **Run** → *Run Configurations...* menu
- *Run Configurations...* in **Quick Access**/Ctrl+3.


An interesting Run Configuration is accessible via **Run As** → **Run on Server** context menu on Java Web projects. Once you have configured a server in the **Servers** view, you can use it to easily re-deploy your application to the server, usually without requiring for a restart. Note that in the server properties (accessible via context menu from *Server* view), you can set the **Publishing** to **Automatically deploy when resource changed** so your web project associated to this server will get automatically updated on change without requiring you even to use **Run As** → **Run on server**...

Eclipse also has Run Configurations for specific kinds of project, such as Maven, Applet, OSGi, and Gradle. Extensions for frameworks often provide some specific Run Configurations, so if you work on some framework that's not supported out-of-the-box, make sure you spend some time trying to find an extension for it on Marketplace or on the web.


DEBUG

Debugging in Eclipse is just another flavor of running the







application as documented above. The difference is that we now talk about **Debug Configurations** which are just like **Run Configurations**, but with debug enabled.

An interesting Debug Configuration provides the ability to **connect debugger to an external Java application**. To do so, make sure your application to debug is started with the debug flags, usually `-agentlib:jdwp=transport=dt_socket,server=y,address=8000,suspend=n` or using some more specific flags in some context, then get to the *Debug Configurations* menu (click on toolbar button , or under **Run** menu, or with **Quick Access**/Ctrl+3), and create a new **Remote Java Application** debug configuration. Set the debug port and the related workspace project, then you'll be able to fully debug your external application.

Add **breakpoints** in your Java code by double-clicking on the column to the left of line numbers or using the right-click → **Toggle Breakpoints** context-menu.

Manage breakpoints (enablement, grouping...) from the  **Breakpoints** view. A powerful feature is **Conditional Breakpoints**. On a breakpoint, do **right-click** → **Properties** to specify a condition to stop on a breakpoint.






When hitting a breakpoint:

- the  **Debug** view shows threads and method call stack
- the  **Variables** view shows values of the variables for the currently selected stack frame
- the  **Expressions** view allows to define complex expressions that will be evaluated and visible immediately for the selected frame
- The  **Display** view allows you to write some "scratch" pieces of code to run, evaluate, inspect your custom code in the context of the currently selected stack frame.
- The  **Inspect** context-menu (**Shift+Alt+I**) on a variable, field of expression show the value of the selection
- The  **Watch** context-menu on a variable, field or expressions adds it to the *Expressions* view for constant re-evaluation.

As usual, views can be opened via **Window** → **Show View** menu, **Alt+Shift+Q** shortcut, or using **Quick-Access**/Ctrl+3 and typing the view name. The context-menu operations are also available with **Quick-Access**/Ctrl+3.

Most of those views are visible by default in the *Debug* perspective, that will be recommended when Eclipse the debugger notices the target application is suspended by a breakpoint or an error.

DEBUG SHORTCUTS

SHORTCUT (WIN/LINUX)	SHORTCUT (OSX)	ICON	DESCRIPTION
Double-click on left column	Double-click on left column		Add/remove breakpoint on selected line.
F5	F5		Step into
F6	F6		Step over: go to next line
F7	F7		Step return: go back to caller
F8	F8		Resume: continue execution until next breakpoint
Hovering on variable	Hovering on variable		Show variable value

Ctrl+Shift+I ⌘+Shift+I 🔍 Inspect: show selected expression value

TESTING

Eclipse comes with an integration with JUnit. You can create unit tests from the *New Wizard* under **Java** → **JUnit** category, or on the context-menu on a Java element or from *Quick Access*/Ctrl+3. Run those tests with **right-click** → **Run As** → **JUnit test** on a JUnit test class. The test report will be shown in a dedicated view that shows a clear status of your tests, and allowing to do advanced filtering, to navigate inside your production code and to compare expected and actual results.

Running a JUnit test is a regular Run/Debug Configuration, so the steps mentioned above can be used to control execution and debugging of Unit Tests.

On Marketplace and other places on the web, you can find nice additions for Eclipse regarding testing. For example, support for TestNG, test coverage with EclEmma, automatic generation of tests and easier navigation between unit test and class under test with MoreUnit, or continuous test execution in background with Infinitest can be very useful.

EXPORT

When you're done with your code and want to turn your project into a delivery, you can usually open the **Export Wizard** via the **Export...** context-menu on a project. You can select multiple strategies for export. It's up to you to decide which one is the best according to your project. The most common ones, depending on your project, are **Jar File** and **WAR file**.

MAVEN

Maven integration for Eclipse is provided out-of-the-box, it doesn't require any addition.

To import Maven projects, use the **Existing Maven Projects** wizard from the **File** → **Import...** menu or from *Quick-Access*/Ctrl+3. Maven support in Eclipse will run various analysis, and may recommend you to install some extensions to better support your project.

Do **NOT** use the deprecated `mvn eclipse:eclipse` command.

Once your Maven project is imported, you can write code taking advantage of all Eclipse features, including incremental build and error reporting. If you do need to specifically run the Maven build, right-click on the **Run As...** → **m2Maven build** context-menu. A Maven build in Eclipse is a *Run Configuration*, so it can be tweaked or even debugged as mentioned earlier.

GRADLE

Eclipse has an extension for Gradle projects, which is called **BuildShip**. If your IDE doesn't include Gradle support, you can install it from Marketplace.

JAVASCRIPT DEVELOPMENT

NEW JS PROJECT

To create a new JavaScript Project choose **File** → **New** → **Other** → **JavaScript** → **JavaScript Project**.

Once the Finish button is pressed, the JavaScript project will be created and available in the **Project Explorer** view.

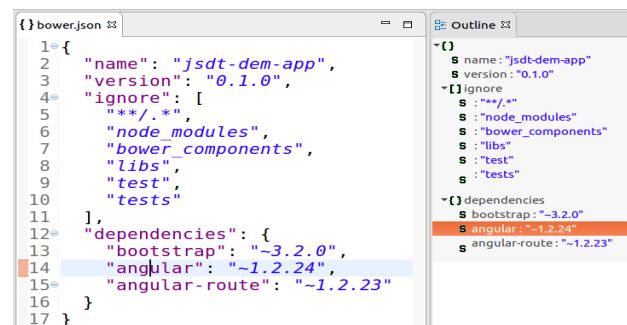
PACKAGE MANAGERS: BOWER, NPM

Eclipse Bower / npm tools use the system installation, hence the following software must be pre-installed:

- Node.js
- npm
- Bower

Getting started with JavaScript package manager is pretty straightforward. In order to start working with Bower one needs to select **File** → **New...** → **Other...** and choose **Bower Init** wizard which helps to create a `bower.json` file depending on a set of preferences.

After pressing the **Finish** button `bower.json` will be created under specified directory. Use this file to add dependencies:



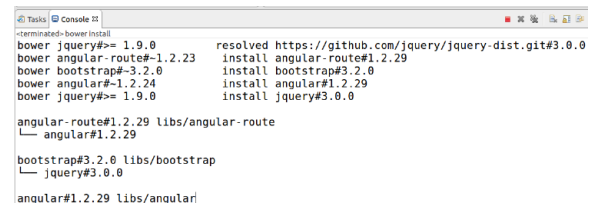
```

1 {
2   "name": "jsdt-dem-app",
3   "version": "0.1.0",
4   "ignore": [
5     "**/*.js",
6     "node_modules",
7     "bower_components",
8     "libs",
9     "test",
10    "tests"
11  ],
12  "dependencies": {
13    "bootstrap": "~3.2.0",
14    "angular": "~1.2.24",
15    "angular-route": "~1.2.23"
16  }
17 }
  
```

The Outline view on the right shows the structure of the file, with the dependencies section expanded.

To install specified dependencies just right-click on the `bower.json` → **Run As...** → **Bower Install**.

The execution output will be available in the **Console View**:



```

bower jquery@= 1.9.0 resolved https://github.com/jquery/jquery-dist.git#3.0.0
bower angular-route@~1.2.23 install angular-route#1.2.29
bower bootstrap@~3.2.0 install bootstrap#3.2.0
bower angular@~1.2.24 install angular#1.2.29
bower jquery@= 1.9.0 install jquery#3.0.0

angular-route#1.2.29 libs/angular-route
└─ angular#1.2.29

bootstrap#3.2.0 libs/bootstrap
└─ jquery#3.0.0

angular#1.2.29 libs/angular
  
```

Then dependencies are now available for development.

npm support can be used the same way: wizard in **File** → **New...** → **Other...** → **npm Init** will create the `package.json` file and **npm Install** / **npm Update** are available under **Run As** for this file.

BUILD SYSTEMS: GRUNT, GULP

Eclipse Grunt / Gulp tools use the system installation, hence the following software must be pre-installed:

- Node.js
- npm
- gulp-cli (npm install -g gulp-cli)
- grunt-cli (npm install -g grunt-cli)

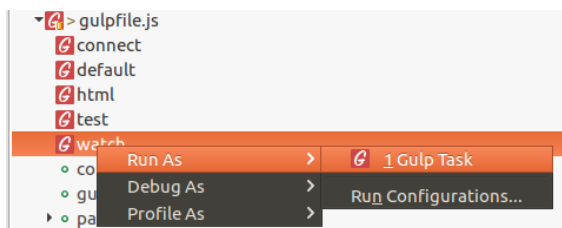
NOTE: gulp-cli/grunt-cli do not install Grunt/Gulp on the system, but rather act as a proxy in charge of running the specific version of Gulp/Grunt requested by the project in `package.json`. This allows using multiple versions of Gulp/Grunt on the same machine.

First thing one should do after creating a project is to define all required dependencies in **package.json**. Both Gulp and Grunt have massive plugin library that can satisfy most, of developer needs:

```
{ "package.json"
1 {
2   "name": "client",
3   "version": "0.1.0",
4   "dependencies": {
5     "gulp": "^3.8.7",
6     "gulp-connect": "^2.0.6"
7   }
8 }
```

After that **npm Install** should be executed (Run As → **npm Install** on package.json). This command will install all the dependencies to the **node_modules** folder.

Now is a high time to start working with **Gruntfile.js / gulpfile.js** and define tasks. All tasks will be available in the Project Explorer view under **Gruntfile.js / gulpfile.js**:



The execution output will be available in the **Console View**:

```
Tasks Console
[19:57:33] Using gulpfile ~/git/jsdt-demo-app/client/gulpfile.js
[19:57:33] Starting 'connect'...
[19:57:33] Finished 'connect' after 24 ms
[19:57:33] Starting 'watch'...
[19:57:33] Finished 'watch' after 24 ms
[19:57:33] Starting 'default'...
[19:57:33] Finished 'default' after 8.32 μs
[19:57:33] Server started http://localhost:2772
[19:57:33] LiveReload started on port 35729
```

DEVELOPING, RUNNING AND DEBUGGING NODE.JS APPS

Eclipse Node.js tools use the system installation, hence the following software must be pre-installed:

- Node.js
- npm

By default, Node.js support in Eclipse will try to use the system-wide Node.js installation that is automatically identified on IDE startup. However, it is also possible to define alternative Node.js binary paths, switch the Node.js used by default to run or debug applications and fully control the Node.js runtimes available in the workspace. To do that, navigate to **Eclipse Preferences → JavaScript → Runtimes**. When there are multiple Node.js installations defined, it is possible to switch the default one used to run the workbench Node.js applications. This can be achieved by clicking on the check box at the left of the desired installation.

NOTE: This change will affect all Node.js launch configurations since this is a workbench-wide setting.

Running and debugging Node.js applications is pretty straightforward. You just need to select **Run → Run Debug**

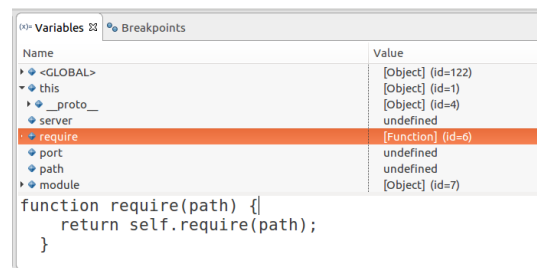
Configurations... or **Debug → Debug Configurations...** and double click **Node.js Application**. This will create a new Node.js Application Run/Debug Configuration where you need to specify a project and a main file to be run / debugged.

Optionally in the **Arguments** tab you can specify Node arguments and working directory.

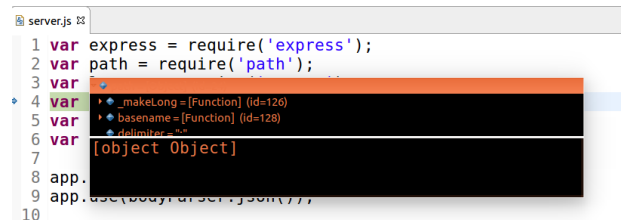
After pressing **Run** or **Debug** you can monitor application's output in the **Console** view and, if debugging, use the various features of the **Debug** perspective.

Node.js Application run/debug actions will be available in context-menu for projects containing a **package.json** file or a **.js** file that is not inside **bower_component** nor **node_modules** folder. To use the shortcut just right click on the project or a **.js** file → **Run As / Debug As → Node.js Application**.

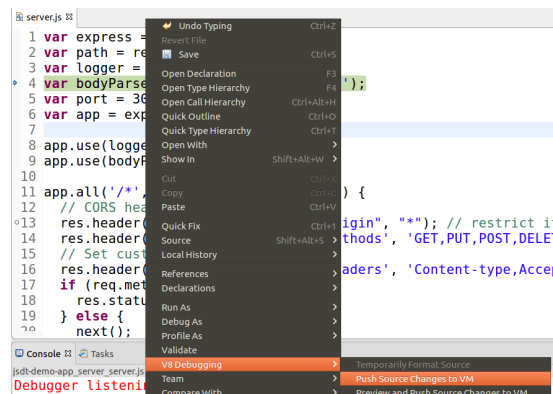
While debugging, all JavaScript variables will be available in the **Variables** view.



Hovering over variables in the editor also shows their value directly:



In order to change the code during debug session just save the file with new changes, and “redeploy” it with right click → **V8 Debugging → Push Source Changes to VM**.



Then new changes will be applied and available in the debug session. Basically, in other aspects the process of debugging Node.

js is the same as for Java. For more details, like debug shortcuts refer to the **Debug** section of the **Java Development 101**.

COMMUNITY

ABOUT THE COMMUNITY AND THE FOUNDATION



The Eclipse Community is an open group of people and involves several hundreds of committers, several thousands of contributors and several millions of users, who contribute to Eclipse and use it either as individuals or on behalf of some organizations. This community aims to deliver good tools and platforms for many things, the Eclipse IDE is only a part of what the whole community produces.

The Eclipse Foundation is a non-profit organization aimed at providing to the community some good infrastructure, community management, communication media, event organization, legal assistance... in order to keep the community innovative and productive.

More info can be found from the eclipse.org site.

IMPORTANT LINKS AND EVENTS

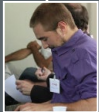
There are a lot of Eclipse-specific events yearly: the community organizes Eclipse Hackathons or DemoCamps or Eclipse Days with the support of the Foundation, and the Foundation organizes 3 major events yearly called EclipseCons.

Stay in the loop with Eclipse, and send feedback thanks to the following links:

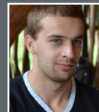
Website	eclipse.org
User forum	eclipse.org/forums/index.php/
Mailing-lists & archives	dev.eclipse.org/mailman/listinfo
Issue tracker	bugs.eclipse.org
Events agenda	events.eclipse.org

Each Eclipse component can be seen as a sub-community, which may use different tools and process. Get in touch with the specific component via its forum or mailing-list to learn more about how to get help and contribute.

ABOUT THE AUTHORS



MICKAEL ISTRIA is working for Red Hat and is an active contributor to various IDE-related projects in the Eclipse community.



ILYA BUZIUK is working for Red Hat and is an active contributor to the JavaScript Development Tools plugins for Eclipse.

BROWSE OUR COLLECTION OF FREE RESOURCES, INCLUDING:

RESEARCH GUIDES: Unbiased insight from leading tech experts

REFCARDZ: Library of 200+ reference cards covering the latest tech topics

COMMUNITIES: Share links, author articles, and engage with other tech experts

JOIN NOW



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

Copyright © 2016 DZone, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

DZONE, INC.

150 PRESTON EXECUTIVE DR.
 CARY, NC 27513
 888.678.0399
 919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
 ISBN-10: 1-936502-77-1



BROUGHT TO YOU IN PARTNERSHIP WITH

