Gabriel Staples
Written: Mar. 2018
Updated: 25 Oct. 2020

This PDF version of the document last generated on (newest date on bottom):

11 Apr. 2020
25 Oct. 2020
31 Jan. 2021

Tested on: Eclipse IDE for C/C++ Developers, Versions:

2019-12 (4.14.0)
2020-09 (4.17.0)

This document is part of eRCaGuy_dotfiles:
https://github.com/ElectricRCAircraftGuy/eRCaGuy_dotfiles/tree/master/eclipse.

***Get the latest version of this document on Google Drive online here*.**

Note that this guide has been compiled over time after years of experience with Eclipse. It is focused on setup on a Linux computer, but the majority of the guidance and information herein applies to Eclipse on any operating system.

Table of Contents:

Install & Setup Steps:

1. ***[IMPORTANT TO PREVENT FREEZES!]*** Increase your system's swap file (virtual memory) to at least 32~64 GB. Follow instructions here: https://linuxize.com/post/how-to-add-swap-space-on-ubuntu-18-04/. In short:

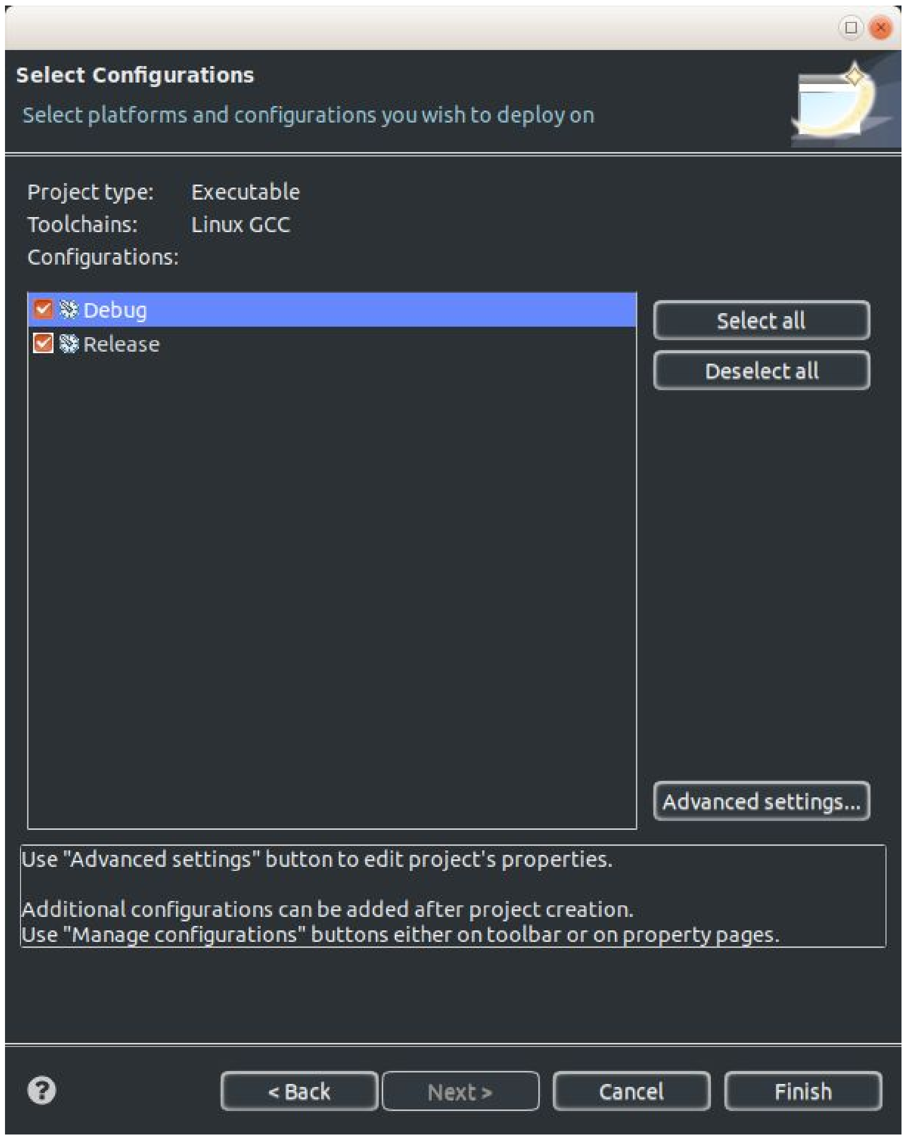
```
sudo fallocate -l 64G /swapfile # create a 64 GB file
sudo chmod 600 /swapfile        # set permissions to rw
for ONLY the user (root!)
sudo mkswap /swapfile
sudo swapon /swapfile
sudo gedit /etc/fstab           # edit the /etc/fstab file
to make these changes persistent (load them each boot)
# ADD this line to bottom (w/out the # comment symbol):
# /swapfile swap swap defaults 0 0
sudo swapon --show              # verify this new 64GB
swap file is now active
cat /proc/sys/vm/swappiness     # not required: verify
your systems "swappiness" value is 60 or so (range is 0 to
100)
```

2. Install Eclipse
    a. Download: https://www.eclipse.org/downloads/
    b. Install by running the latest Eclipse Installer: `eclipse-installer/eclipse-inst`
3. ***[IMPORTANT TO PREVENT FREEZES!]*** Configure the eclipse.ini file as explained below in the "Configure 'eclipse.ini'" section to SIGNIFICANTLY increase the RAM available to Eclipse for its powerful indexer.
4. Choose a workspace. I recommend you keep your project files elsewhere (ie: *outside* the workspace). Read more about my reasoning for this later in this doc.
    a. Ex: project files in "~/dev/my_project" and eclipse workspace in "~/dev/eclipse-workspace"
    b. If the "Welcome" window opens when you enter Eclipse for the first time, feel free to just close that sub-window.
5. Install plugins below, & configure their settings as described below. *If you have multiple workspaces, this must be done *individually* for each workspace.*
6. ***[IMPORTANT TO PREVENT FREEZES!]*** Unset "Build Automatically". *If you have multiple workspaces, this must be done *individually* for each workspace*.
    a. "Project" menu at top of Eclipse → Uncheck "Build Automatically"

    b. See: https://www.benchresources.net/how-to-build-java-project-in-eclipse-ide-automatically/

7. Add a desired project.
    a. Notes: we need to watch out for symbolic links in the project; find symbolic links in it with `` `*find . -type l*` `` or `` `*find . -type l -ls*` ``; see here: https://askubuntu.com/questions/522051/how-to-list-all-symbolic-links-in-a-directory/522059#522059
    b. ***[IMPORTANT TO PREVENT FREEZES!]***
        i. Configure **Advanced Project Settings** to **exclude directories** which:
            1. might have recursive (circular) symbolic links
            2. Are HUGE third-party libraries to index (such as Boost)
                a. Note that some third-party libraries can be so huge they are not practical to index on a normal computer and/or in a normal amount of time! In some cases, it could take up to 64 GB ~ 128 GB of RAM to properly index (and most computers just don't have that!) and as many as several hours to a few days of time. If you ever have a project with a bunch of third party libraries, you are welcome to bump up your swap file to 32 GB ~ 64 GB and give it a shot if you like, but it will potentially make your computer bogged down with every CPU pegged at 100% for many hours, if not days, and you may have to set your "eclipse.ini" file (explained below) to have `-Xmx32000m` or `-Xmx64000m` instead, thereby providing it 32GB or 64GB heap space, respectively.
            3. Are build or bin (binary output) directories
            4. Are bazel (a type of build system) output directories (such as bazel-out or bazel-bin).
        ii. Example directories to *exclude* include the following:
            1. bin
            2. build
            3. bazel-out
            4. bazel-bin
            5. bazel-testlogs
            6. third_party, third-party, or other folders known to contain large external libraries
        iii. ***[IMPORTANT TO PREVENT FREEZES!]*** How to exclude directories:
            1. You must do this BEFORE finishing the creation of this new project or else it will FREEZE while trying to index folders you SHOULD HAVE EXCLUDED. During the Project creation phase, simply click Advanced Project Settings. Once you do that, see the "Project Resource Filters" section below for more details.

c. Click "Apply and Close". For HUGE, pre-existing projects, expect it to take up to ***5 minutes ~ 2+ hrs to complete!***, taking 100% of one of your CPUs the entire time. This is because it's scanning across every file in the repo. A more typical time for this "Apply and Close" process to complete on a huge repo and good computer might be 45 minutes to 1 hr or so. For small projects or new projects, it should be almost instantaneous. Here's the screen you'll get back to once the "Apply and Close" process is complete:



Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Executable
Toolchains: Linux GCC
Configurations:

✓ Debug
✓ Release

Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

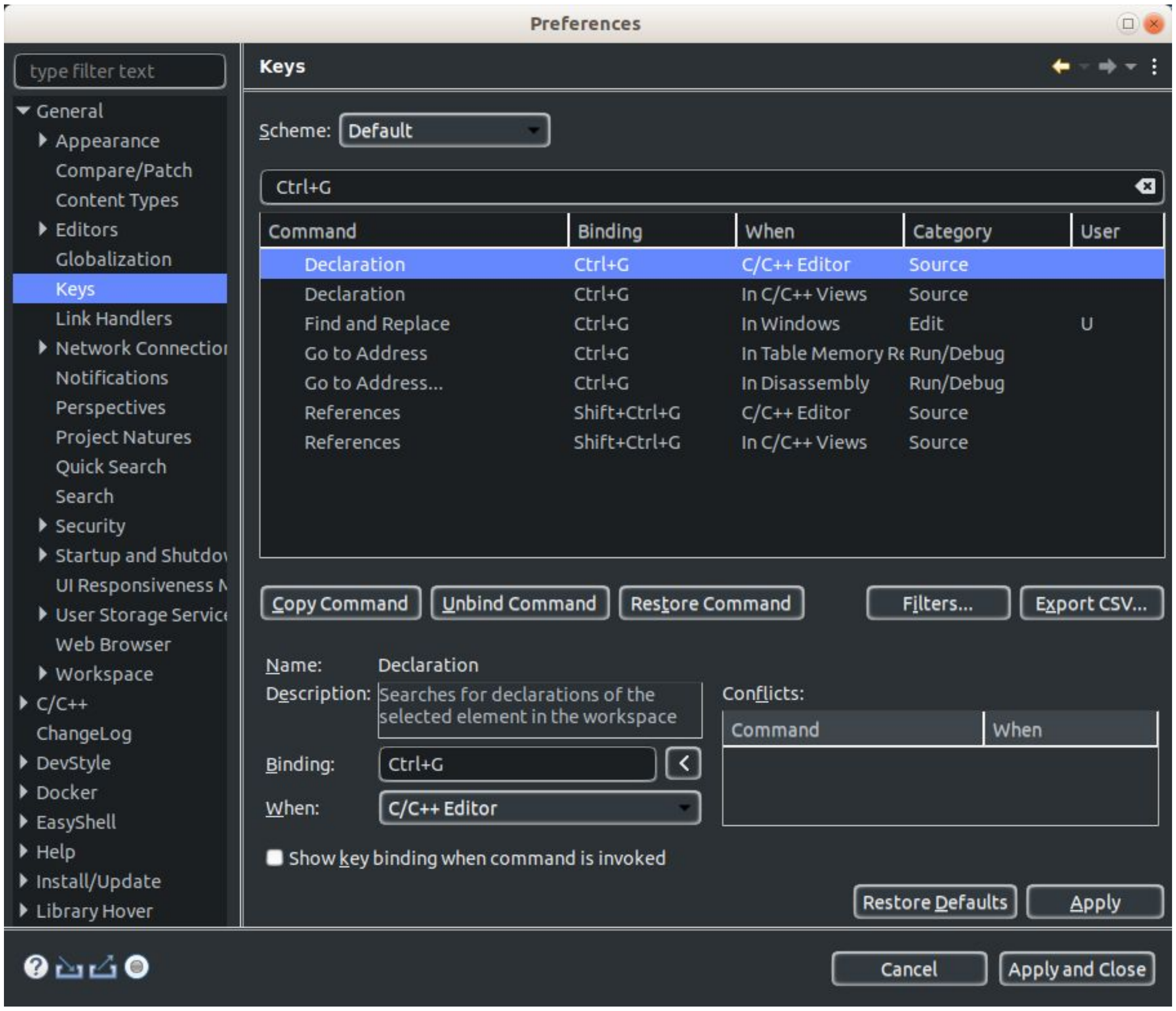
.

d. IF YOU SEE A POPUP WINDOW WHICH SAYS "Refreshing workspace", then something like "The user operation is waiting for background work to complete", [I THINK] you messed up! You forgot to click "Project" at the top of Eclipse then → uncheck "Build Automatically"! YOU NEED TO UNCHECK THE Project → "Build Automatically" setting for this workspace, as described above, before adding a large project. Go ahead and click the little "X" to cancel this process. Then cancel adding this project, fix that setting for this workspace (ensure Project → "Build

Automatically" is NOT checked), and start this process from the beginning again to add this project to the workspace.

   e. Now, click "Finish" to finish adding this project. At this point, the indexer will begin to do its work. You'll see its % complete status in the bottom-right corner of the screen. You can continue working like normal while the indexer indexes the repo. If you forgot to exclude any huge folders above, the indexer will probably exceed the RAM limitations you set previously in the "eclipse.ini" file, and eventually freeze and crash Eclipse. Simply delete the project (or edit the Project Properties right after re-opening Eclipse if you can do so fast enough before it crashes again) and add the project again, this time properly excluding any external directories which are not directly part of your project and are too big to index. Beware that some projects, such as the C++ Boost libraries, may require at least 64 GB of RAM (if not 128 GB?) to get fully indexed.

8. Reset the C++ perspective to default in case it got messed up: Window → Perspective → Reset Perspective → Reset Perspective. Then, click the little arrow next to your project name in the "Project Explorer" pane at the far left to expand and show all files and folders in your project. Click the little "Link with Editor" button (left and right yellow arrows, as shown near the bottom of this document) to cause the Project Explorer to always track to the file currently open. Then press **Ctrl + Shift + R** to search for a file of your choice to edit.
9. *****Change the Ctrl + H keyboard shortcut (binding) from the "Open Search Dialog" command (which defaults to the "C/C++ Search" tab) to the "File Search" command (which opens up to the "File Search" tab):
   a. Window → Preferences → General → Keys → search for "File Search" → click the empty "Binding" box, then press 'Ctrl + H'. Click "Apply and Close".
   b. Done! Now Ctrl + H defaults to the "File Search" search tab! No longer do you have to always remember to click this tab before trying to search!
   c. Source: https://stackoverflow.com/questions/91984/how-do-i-hotkey-directly-to-file-search-tab-in-eclipse/92097#92097
10. Change the Ctrl + F Find/Replace shortcut key ***to Ctrl + G*** since DevStyle takes over the Ctrl + F shortcut key for its "Inline Search and Replace" tool.
   a. Window → Preferences → General → Keys → search for "Find and Replace" → click the "Binding" box which currently has "Ctrl+F" written in it, then erase what's in there, and press 'Ctrl + G'. Click "Apply". Now, in the search box above, where you searched for "Find and Replace", search for "Ctrl+G". You'll see several other functions attached to this shortcut, as shown here:



Keys

Scheme: Default

Ctrl+G

Command	Binding	When	Category	User
Declaration	Ctrl+G	C/C++ Editor	Source	
Declaration	Ctrl+G	In C/C++ Views	Source	
Find and Replace	Ctrl+G	In Windows	Edit	U
Go to Address	Ctrl+G	In Table Memory R	Run/Debug	
Go to Address...	Ctrl+G	In Disassembly	Run/Debug	
References	Shift+Ctrl+G	C/C++ Editor	Source	
References	Shift+Ctrl+G	In C/C++ Views	Source	

Name: Declaration

Description: Searches for declarations of the selected element in the workspace

Binding: Ctrl+G

When: C/C++ Editor

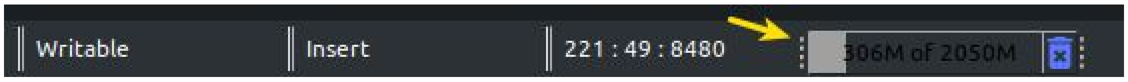
Conflicts:

Command	When

Show key binding when command is invoked

. You need to delete the one highlighted in blue since it will conflict otherwise. It will conflict because it is set to be active when in the “C/C++ Editor”, which is when we also want to use “Find and Replace”. So, select it, erase the “Ctrl+G” in the “Binding” box, and hit “Apply.” Click “Apply and Close”.

11. Now try your 4 find shortcuts (*read more about each one later in this doc*). While editing a file, do:
    a. **Ctrl + F** = DevStyle Inline Search
    b. **Ctrl + G** = [custom shortcut we just set up] Eclipse’s original Find/Replace tool
    c. **Ctrl + H** = File Search [we just set this above] for searching for text WITHIN files, projects, "Working sets", etc.
    d. **Ctrl + Shift + R** = “Open Resource” search tool for searching for any *file* you want to open in your project! Use asterisks (*) for wildcards.
        i. NB: YOU MAY NEED TO BEGIN AND END YOUR SEARCH STRING WITH A WILDCARD (*)! ECLIPSE DOES *NOT* HAVE A FANCY FUZZY SEARCH CAPABILITY LIKE SUBLIME TEXT 3! See my answer here: https://stackoverflow.com/questions/24379092/eclipse-ctrlshiftr-not-showing-all-files-in-the-project/65987607#65987607.

12. Set formatting, tab spaces, vertical column marker at 120 (or 100) chars, & convert tabs to spaces.
    a. Follow these instructions: https://stackoverflow.com/questions/1650652/changing-editor-tab-width-in-eclipse-3-5/58494110#58494110 and these instructions: https://stackoverflow.com/questions/1248895/is-there-an-eclipse-line-width-marker/49372370#49372370.
    b. In short:
        i. Window --> Preferences --> General --> Editors --> Text Editors --> ensure "Displayed tab width" is what you want (ex: 4). Also check the box for “Insert spaces for tabs”, and “Show print margin”. Set the “Print margin column” to 120 (or 100). Many of these settings are overridden by our C/C++ settings, however, so we must set them as well:
        ii. Window --> Preferences --> Java OR C/C++ [depending on which you're using] --> Code Style --> Formatter --> click “New” to create a new, custom profile you can edit. Give your profile a name (ex: “GS_custom”). Select which profile you’d like to initialize these settings with (ex: “K&R [built-in]”). Ensure the “Open the edit dialog now” box is checked. Click “OK”.
            1. Now, you are in your profile “Edit” window. Click "Indentation" tab on top-left --> change “Tab policy” from “Tabs only” to “Spaces only”, set “Indentation size” and “Tab size” to what you want (ex: 4).
            2. Now click the “Line Wrapping” tab at the top and change the “Maximum line width” from 80 to 120 (or 100) characters. THIS OVERRIDES THE “Print margin column” we set previously above, as described by my 2nd link above. Click “OK”, then “Apply and Close”.
            3. You will now see a vertical bar at the 120 (or 100) char column marker as a visual indicator to indicate you shouldn’t make your lines longer than this. Also, your tab size is now 4 (or whatever you set above), and tabs are automatically converted to spaces (if you set this above).
13. Turn on showing the heap status at the bottom of the Eclipse window (if it isn’t already on by default):
    a. Window → Preferences → General → check the box for “Show heap status” → click “Apply and Close”. Here’s what it looks like now at the bottom of the Eclipse window!
    b. 

    c. Source: https://stackoverflow.com/questions/31254187/how-to-view-memory-usage-in-eclipse-beginner/31255323#31255323

14. Tell Eclipse that Arduino `*.ino` files are a type of C++ Source File so that it will index them and open them up in its C/C++ editor with proper syntax highlighting whenever you open them! Following these instructions here: https://stackoverflow.com/questions/33474629/is-ecplise-cdts-indexer-limited-to-the-common-filetypes-for-sources-and-headers/33520998#33520998.
    a. In short: Window --> Preferences --> C/C++ --> File Types --> click "New..." --> type in "*.ino" as the pattern, and set the "Type" to "C++ Source File" --> click "OK" --> "Apply and Close". Eclipse will now automatically re-index the whole project, indexing and treating all Arduino *.ino files as C++ source files! Perfect! Now you can easily use Eclipse to create, edit, and navigate Arduino source code, although I still recommend you build the Arduino code externally, either with the Arduino IDE OR with the Arduino CLI (Command Line Interface) tool (get it on GitHub here).
15. For any Arduino project or library project you are working on, symlink (symbolically link) in the Arduino core source code so that Eclipse will index it within your project and allow you to jump to definitions within it! See full instructions in my answer here: https://stackoverflow.com/questions/9272882/eclipse-cdt-c-c-include-a-header-file-from-another-project/65975086#65975086.
16. Set all "*.inl" files as Type "C++ Header File", since some people like to use .inl as the extension for special header files which contain only "inline" functions or methods, such as those defined for C++ template functions and classes, as templates must be *fully defined* in header files per the C++ specification.
    a. Follow the instructional pattern set just above for the .ino file we just did.
17. Read & study the "Eclipse Usage, Workflow, Help, Tips & Tricks" section below.
18. Done!

Plugins to Install:

1. **Darkest Dark Theme with DevStyle** - *DevStyle functionally replaces the apparently now unmaintained Glance 1.2.1 and Eclipse Color Theme 1.0.0 plugins which used to exist. The Sublime Text 3 (Monokai) color theme by Jeremy Shepherd, mentioned below, was originally created for and hosted by the Eclipse Color Theme plugin website, but is also compatible with the DevStyle plugin. The Ctrl + F DevStyle Inline Search tool which overrides Eclipse's default search tool functionally replaces the Ctrl + Alt + F quick-search and highlight tool that Glance previously offered.*
    a. Once you've installed this DevStyle plugin, download & install the color theme "**Sublime Text 3 (Monokai) - by Jeremy Shepherd**", here: http://www.eclipsecolorthemes.org/?view=theme&id=25808 [DEAD LINK!] ⇐ Link is dead, so instead do one of these two options:
        i. 1) Copy and paste it out of this Stack Overflow question now instead: https://stackoverflow.com/questions/27022313/how-to-quickly-copy-the-current-editing-file-name-or-full-file-path-in-eclipse/59626187#59626187.

        ii. OR 2) Download it from this Github project here: https://github.com/ElectricRCAircraftGuy/eRCaGuy_dotfiles/tree/master/eclipse/color_themes → "Sublime Text 3 (Monokai) - by Jeremy Shepherd--theme-25808.xml".
    b. Install this Sublime Text 3 theme: Window → Preferences → DevStyle → Color Themes → click the "Import…" Link under "Editor theme", and choose the Jeremy Shepherd xml file you just downloaded above. Also check the box for "Theme background" and (optional) "Enable Breadcrumb" [not sure what this one does yet] under "Editor theme". Click "Apply and Close", then restart Eclipse.
2. **EasyShell** - *EasyShell functionally replaces the apparently now unmaintained StartExplorer, Path Tools, and Copy as Path plugins which used to exist.* For screenshots & usage information see here: https://stackoverflow.com/questions/27022313/how-to-quickly-copy-the-current-editing-file-name-or-full-file-path-in-eclipse/59626187#59626187.
3. **Bash Editor 2.2.3**
4. **PyDev - Python IDE for Eclipse 7.5.0**
5. **AnyEdit Tools** - Allows you to sort lines of text or code! See my comment under this answer on Stack Overflow (https://stackoverflow.com/questions/3632206/does-eclipse-have-a-way-to-alphabetically-sort-lines-within-a-selection-of-text/14899278#14899278):
    a. "AnyEdit works great! More info here: https://marketplace.eclipse.org/content/anyedit-tools. To use after installing: ***select a bunch of lines*** → ***right click*** → ***Sort*** → ***Case-Sensitive A-z***. Done!"
6. **Eclipse Web Developer Tools 3.16** - "Includes the HTML, CSS, and JSON Editors, and JavaScript Development Tools from the Eclipse Web Tools Platform project…". Also includes a text editor with syntax highlighting for *.yaml files!

Configure "eclipse.ini":

Back up "eclipse.ini", found within the same folder as the eclipse executable file (ex: inside "~/eclipse/cpp-2019-12/eclipse/eclipse.ini"):

```
cd ~/eclipse/cpp-2019-12/eclipse
cp eclipse.ini eclipse.ini.bak
```

Then edit eclipse.ini as follows:

Change from:

-Xms256m
-Xmx1024m

To:

(for a system with 64 GB RAM):
-Xms10000m
-Xmx32000m

(for a system with 32 GB RAM)
[NB: <u>at least</u> this much RAM is recommended for any decent professional development machine, although for small home projects, anything > 4GB RAM is technically enough!]:
-Xms10000m
-Xmx24000m
OR
-Xms4096m
-Xmx12288m

(for a system with 16 GB RAM):
-Xms10000m
-Xmx12000m
OR
-Xms10000m
-Xmx10000m
OR
-Xms8000m
-Xmx10000m
OR
-Xms8000m
-Xmx8000m
OR
-Xms6000m
-Xmx8000m
OR
-Xms2048m
-Xmx8192m
OR
-Xms1024m
-Xmx4096m

(for a system with 8 GB RAM):
-Xms4000m
-Xmx4000m
OR
-Xms512m
-Xmx2048m

(for a system with 4 GB RAM):
-Xms2000m
-Xmx2000m

What does this do? “-Xms” is the Java Virtual Machine (JVM) memory setting to set the *default heap memory (default RAM usage)* size, and “-Xmx” is the JVM memory setting to set the *maximum heap memory (max RAM usage)* size. To read more about these JVM “-Xms” and “-Xmx” values, see here: https://stackoverflow.com/questions/3571203/what-are-runtime-getruntime-totalmemory-and-freememory.

Using “-Xms10000m” and “-Xmx32000m”, for example, gives Eclipse a HUGE 10GB of heap memory by default, with the option of using up to 32GB of heap space! You might think this is huge, but for extremely large projects, even this isn’t enough, as indexing can grow in memory size exponentially. Note that for a machine with 64 GB RAM, “-Xms10000m” and “-Xmx32000m” is a good starting setting, although it’s possible you’ll still need to increase “-Xmx” up to “-Xmx54000m” or so. If you install this on a computer with less RAM, it is recommended to set Xms to something more reasonable (<= ~1/2 of the total available RAM), and Xmx to ~1/2 of the total available RAM or so, as shown above. Note that “-Xms” should be <= “-Xmx”.

Save the file and close and reopen Eclipse.

Freezes, & Clearing Eclipse’s Cached .pdom Indexer File For Your Project:

[IMPORTANT TO PREVENT FREEZES!]
Freezes:
After using Eclipse for a long time, the indexer .pdom file may become extremely large, causing Eclipse to periodically freeze while editing due to problems it’s having while indexing. This can manifest itself as slow syntax-highlighting or failure to go to a declaration when you Ctrl + Click on it, etc. It appears that the index file gradually increases in size until Eclipse just constantly freezes. This may be more likely to happen on extremely large projects/mono-repos since they have so much to index and produce such huge index files. Doing many branch changes with `git checkout` may also contribute to this problem since each time you switch to a different branch the indexer needs to go back to work. **To fix this problem, delete your huge .pdom indexer file as follows:**

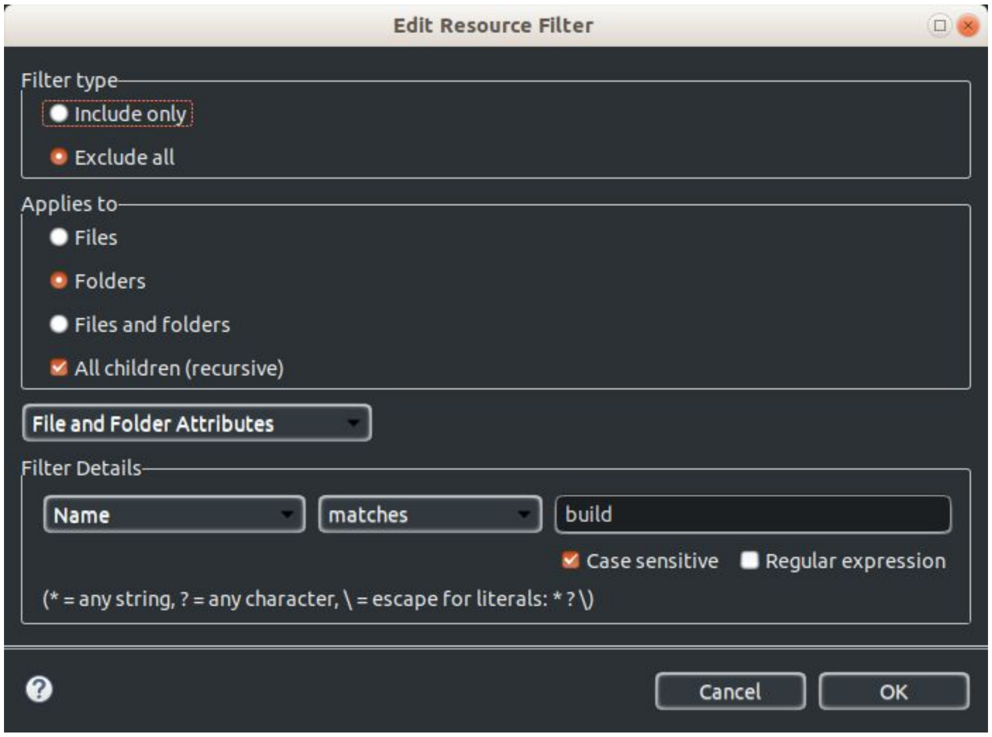
1. Close Eclipse.
2. `cd` to **<your workspace folder>/.metadata/.plugins/org.eclipse.cdt.core** and delete any super large **.pdom** files.

3. Re-open Eclipse. The indexer will now have to start over completely from scratch for any project whose indexer .pdom file you deleted. **This may take from several hours for a large project to overnight or even 1~2 days for a humongous mono-repo.**
4. Notes:
   a. Some people recommend doing this as often as once a week to keep their system running smoothly.
   b. **If you haven't edited your "eclipse.ini" file, however, as well, to increase your RAM granted to Eclipse, DO THIS AS WELL, AS IT MAY HELP A BUNCH TOO! See the *Configure "eclipse.ini"* section in this document.**

Project Resource Filters *(adding resources, linked resources, excluded resources, virtual folders, etc)*:

1. Resource Filters ("Include only" or "Exclude all")

You may need to include or exclude certain resources, like including a "src" directory but excluding its "build" or "bin" directory. Here's **how to exclude a certain directory and all of its contents:**
Right-click the project → Properties → Resource → Resources Filters → Add Filter → select "Exclude all" for "Filter type", "Applies to" "Folders", check the box for "All children (recursive)", Set the "Filter Details" to "Name" "matches" [directory name, ex: "build"], check the box for "Case sensitive" if desired, then click "OK". Here's what that looks like. Note that these filters are relative to your project directory, so if your project directory is "~/dev/my_project", then the "build" folder being excluded below would be located at "~/dev/my_project/build". Be sure to select the "All children (recursive)" option!


Edit Resource Filter

Filter type
- Include only
- [✓ Exclude all]

Applies to
- Files
- [✓ Folders]
- Files and folders
- [✓ All children (recursive)]

File and Folder Attributes

Filter Details

Name | matches | build

[✓ Case sensitive] Regular expression

(* = any string, ? = any character, \ = escape for literals: * ? \)

Note: Eclipse's help menu says to use "Include only" OR "Exclude all" Resource Filters, but NOT both at once really. If you use both at once, a file or folder MUST be in the Included list and NOT in the Excluded list in order to be included.

So, what if you want to Exclude an entire "build" directory, for instance, EXCEPT a couple files or folders inside it. How do you do that!?

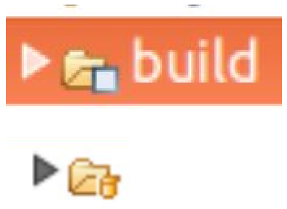
Answer:

2. "Virtual" Folders and "Links" to files or folders (ie: Linked Folders or Linked Files)

First, since the "build" folder is completely excluded, let's add a "Virtual" version of it as a folder NOT on the file-system, but rather, only "virtually" shown in Eclipse!

Right click a folder in the Project Explorer bar on the left, in which you'd like to add a Virtual folder. In the right-click menu, go to **New → Folder → click the "Advanced" button → select the radio button for "Folder is not located in the file system (Virtual Folder)" → add a "Folder name", such as "build" in this case since it's excluded above, and click "Finish"**. You now have a virtual "build" folder with nothing in it.

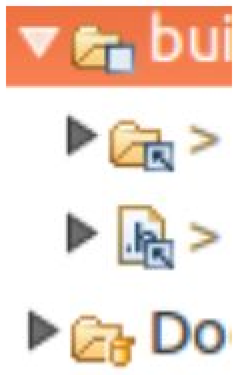
Once you've added a Virtual Folder like this, you'll see it has a different icon, with a *tiny little square* in the bottom right of the folder, as opposed to a tiny little cylinder shape. A Virtual Folder icon looks like this. Here a virtual folder icon is on top, with a regular folder icon underneath it. The virtual "build" folder is selected here, hence why it's highlighted.



Then, you can right-click any folder (such as this virtual "build" folder) you'd like to add a **Linked Folder** or **Linked File** into and do so like this: **Right click → New → select "File" OR "Folder" → click the "Advanced" button → select the option for "Linked Folder" or "Link to file in the file system", as appropriate → click the "Browse" button and browse to the resource, or type its path into the text bar → click Finish.**

Done! Now you can have Linked files or folders within any other folders, including Virtual Folders. This lets you therefore *manually* ADD BACK IN files or folders which are otherwise EXCLUDED by your Resource Filters you previously set above!

The icon for **linked resources** (whether a linked folder or a linked file) has a tiny *little arrow in the bottom-right of the icon*, like this. Here you can see, from top to bottom, a Virtual Folder, Linked Folder, Linked File, Normal Folder.



Eclipse Usage, Workflow, Help, Tips & Tricks:

1. Workspaces, Projects, Perspectives, and Working Sets:

a. Eclipse has the concept of Workspaces, Projects, Perspectives, and Working Sets, which can be very confusing for anyone new to using Eclipse.

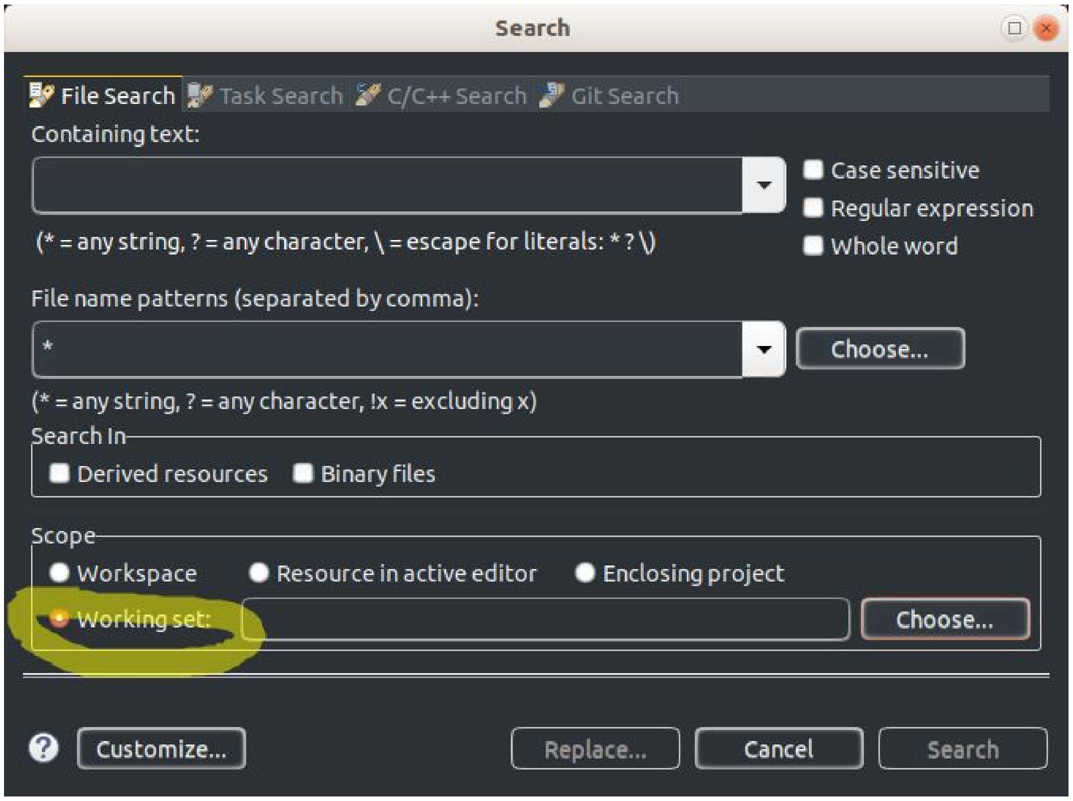
b. A “**workspace**” is a folder where Eclipse saves all configuration information for your entire IDE in a given configuration. When you go to Window → Preferences, all settings you change there are part of your “workspace” settings and get saved to your workspace folder. *This includes individual settings for each of your installed plugins, so if you have multiple workspaces, you must configure the plugin settings individually for each workspace.* ***[IMPORTANT TO PREVENT FREEZES!]:*** additionally, you must uncheck the “Build Automatically” option in the “Project” menu for each new workspace you create *before* adding a new project to it! You can have only one workspace open per instance of the IDE open, but you can have multiple workspaces to choose from when you first start the IDE, and you can have multiple instances of the IDE running simultaneously and in different workspaces. It is common for a developer to have only a single workspace on their computer, although using multiple workspaces to separate groups of projects is not uncommon either.

c. A “**project**” is a given chunk of software you are working on. Each workspace can have many projects. Projects are shown in the “Project Explorer” window on the left-hand side. Eclipse defaults to storing new projects inside your workspace folder, but many people advise against this and recommend keeping your project folders separate and outside of your workspace folder. This is what I recommend too, so that if you ever upgrade your IDE to a later version, you don’t have to worry about corrupted workspaces leading to accidentally-deleted projects. It is common for a developer to have many projects on their computer, all in a single workspace. If a developer has many many projects, they might group them together in a way that makes sense into multiple workspaces. Remember, the actual contents and files/folders of projects *can* reside inside a physical workspace folder as well, but don’t have to. I prefer to keep my projects in “~/dev/project_1”, “~/dev/project_2”, etc, and my workspaces organized separately as “~/dev/eclipse_workspace_1”, “~/dev/eclipse_workspace_2”, etc. Again, I prefer to keep my project folders physically outside of my workspace folders, but you can do what you like.

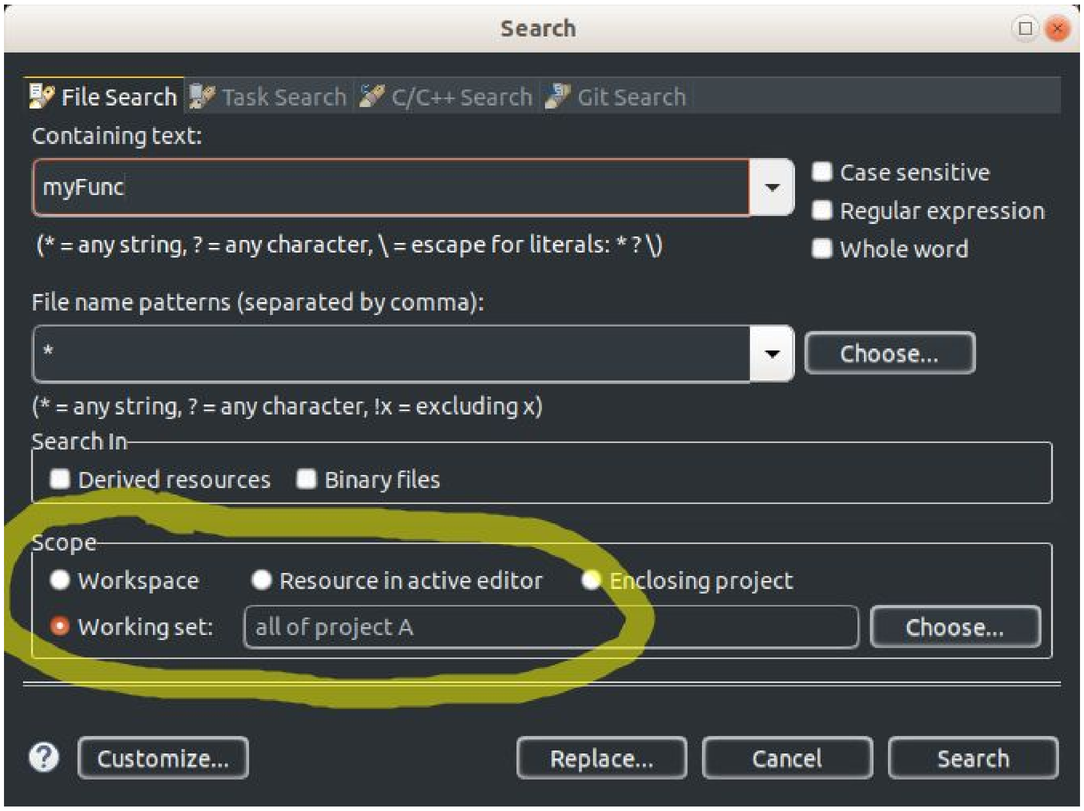
d. A “**perspective**” is like a view, or set of windows and their positions on the screen. If you ever wonder where your windows went or why the heck they got moved, resized, or otherwise messed up somehow, you can reset your perspective to default by going to Window → Perspective → Reset Perspective. If you ever build or debug in Eclipse, a new perspective will automatically open for debugging or whatever. Just be ready for that. I prefer, however, to use Eclipse just as an editor and do all my building from the command-line. If doing heavy microcontroller programming, however, it is *extremely common* for the editor to be Eclipse-based, such as the STM32 Cube IDE, in which case building is integrated into Eclipse and for personal projects I might as well just use the built-in build tools ST provides inside Eclipse as well.

e. A “**working set**” is a custom search filter. It allows you to set up custom filters for searching when working on different aspects of a project. Ex: you might set up a "module A" "working set" for easily searching for functions and files just within Module A (where each "module" is just a sub-folder or set of folders within your project), and a different "working set" for searching just within Module B. Working sets can be very intricate and detailed, have a ton of options, and allow searching across multiple projects, files, resources, etc, within a workspace. It is common for a developer to have many working sets in their workspace, some of which might draw from multiple projects or directories in order to capture source code, generated files, and library files all from a single piece of software they are working on. Working sets allow you to more quickly do *targeted searches* for files with Ctrl + Shift + R or search for *targeted file content* with Ctrl + H. Just choose the "working sets" of interest in the respective search tool you're using.

i. Here is one way to create a working set: use Ctrl + H to open the "File Search" tool to look for text within a file. Here's what that tool looks like:



. At the bottom of it, select "Working set", then "Choose" → New → "Resource" → Next → expand any projects you see you want to choose from, and click the check boxes next to any folders you want to include, then type in something in the "Working set name" box, such as "all of project A", then click "Finish". This brings you to the previous window. Select this working set ("all of project A"). If you have multiple working sets you can select as many or few of them as you want here, to include in your search. Click "OK". Now you are back in the File Search tool and can search for any text within resources contained in this Working set, as shown here:



Containing text: myFunc

(* = any string, ? = any character, \ = escape for literals: * ? \)

[ ] Case sensitive
[ ] Regular expression
[ ] Whole word

File name patterns (separated by comma): *

(* = any string, ? = any character, !x = excluding x)

Search In
[ ] Derived resources [ ] Binary files

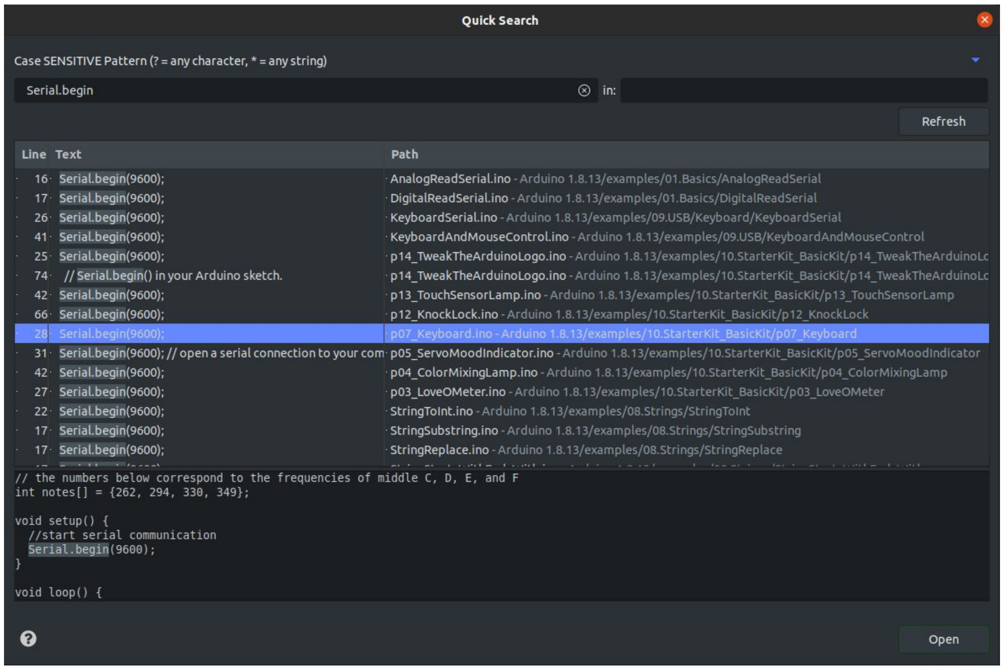
Scope
○ Workspace ○ Resource in active editor ○ Enclosing project
● Working set: all of project A

.

2. Common Eclipse Usage & Shortcuts:

a. **Window → Perspective → Reset Perspective** = reset your perspective (window arrangement) view to default
b. **Window → Preferences** = adjust workspace settings
c. **Help → Eclipse Marketplace** = the place you download and install plugins and 3rd-party tools such as DevStyle and EasyShell.
d. Right click project in Project Explorer
    i. →Properties = all project settings; some of these may override workspace settings
    ii. → Refresh = search for new folders or files; do this if you ever rename, add, or delete a file or folder from an external file manager or the command-line, to tell Eclipse about the change
    iii. → Index → Freshen all Files = ensure indexing of all files is up-to-date
    iv. → Index → Rebuild = start indexing over from scratch (could take many minutes to hours depending on the size of your project)
e. **Ctrl + Tab** = jump between corresponding source (.c/.cpp) and header (.h/.hpp/.hh) files.
f. **Ctrl + Click** = jump to implementation (try it on functions, variables, header file `#include`s, etc); SUPER USEFUL! Eclipse has an excellent indexer to make this possible!
g. **Alt + Left Arrow** = navigate back

h. **Alt + Right Arrow** = navigate forward
i. **Ctrl + Space** = autocomplete: open the auto-complete menu. This is particularly helpful when you remember the start of a long variable name and want to autocomplete the rest of it: just start typing the name, then press Ctrl + Space to open up a menu you can choose the variable from. It also works very well on class or struct objects: type the name of a class or struct instance variable, then the dot (.) or arrow (->) operator (depending on whether it is the object directly or a pointer to it), and press Ctrl + Space to see what members are available in this object.
j. **Ctrl + Shift + L** = bring up the list of all shortcuts (see here). You can type to search for a shortcut, but the search is by alphabetical order only (not a good fuzzy search), and therefore very poor. Note that once you've selected a shortcut key sequence from the popup list, pressing Enter on it OR double-clicking the mouse on it will activate it as if you had pressed the correct shortcut key sequence for it.
k. **Ctrl + Z** = undo
l. **Ctrl + Shift + Z** = redo
m. **Ctrl + D** = delete a line
n. **Ctrl + L** = Go to Line
o. **Ctrl + Shift + P** = find matching brace 'p'air
p. The various *search tools*:
    i. **Ctrl + F** = [overridden by DevStyle] *DevStyle's Inline Search*; see below for how to use DevStyle
    ii. **Ctrl + G** = [custom shortcut you should have set up above] Eclipse's original Find/Replace tool; very handy!
    iii. **Ctrl + H** = "File Search" tool [assuming you set it to this above; otherwise it defaults to the less-useful "C/C++ Search"]; very handy! Note this is NOT for searching for files--that's what Ctrl + Shift + R is for! This is for searching for text, functions, variables, etc, WITHIN files, projects, "Working sets", etc! *This is a VERY POWERFUL TOOL! Learn how to use it to do mass Find-Replace (while being very granular or selective if desired) across multiple files at once here:* https://stackoverflow.com/questions/6800799/replace-string-in-all-files-in-eclipse/50283848#50283848.
    iv. **Ctrl + Shift + Alt + L** (or: Search → Quick Search) = "Quick Search" tool: a very handy tool to quickly do a search in the code base for matching patterns, *live*. This lets you rapidly search for one string after another, in the code base, seeing the results update live as you type. Double-click any search result line to jump to it in the code base. Click the little tiny down-arrow in the top-right of the window to toggle the "Case sensitive" check-mark on or off. I'm not really sure what the "Keep Open" option does there.

1. 

Case SENSITIVE Pattern (? = any character, * = any string)

Serial.begin in:

Line	Text	Path
16	Serial.begin(9600);	AnalogReadSerial.ino - Arduino 1.8.13/examples/01.Basics/AnalogReadSerial
17	Serial.begin(9600);	DigitalReadSerial.ino - Arduino 1.8.13/examples/01.Basics/DigitalReadSerial
26	Serial.begin(9600);	KeyboardSerial.ino - Arduino 1.8.13/examples/09.USB/Keyboard/KeyboardSerial
41	Serial.begin(9600);	KeyboardAndMouseControl.ino - Arduino 1.8.13/examples/09.USB/KeyboardAndMouseControl
25	Serial.begin(9600);	p14_TweakTheArduinoLogo.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p14_TweakTheArduinoLc
74	// Serial.begin() in your Arduino sketch.	p14_TweakTheArduinoLogo.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p14_TweakTheArduinoLc
42	Serial.begin(9600);	p13_TouchSensorLamp.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p13_TouchSensorLamp
66	Serial.begin(9600);	p12_KnockLock.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p12_KnockLock
28	Serial.begin(9600);	p07_Keyboard.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p07_Keyboard
31	Serial.begin(9600); // open a serial connection to your com	p05_ServoMoodIndicator.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p05_ServoMoodIndicator
42	Serial.begin(9600);	p04_ColorMixingLamp.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p04_ColorMixingLamp
27	Serial.begin(9600);	p03_LoveOMeter.ino - Arduino 1.8.13/examples/10.StarterKit_BasicKit/p03_LoveOMeter
22	Serial.begin(9600);	StringToInt.ino - Arduino 1.8.13/examples/08.Strings/StringToInt
17	Serial.begin(9600);	StringSubstring.ino - Arduino 1.8.13/examples/08.Strings/StringSubstring
17	Serial.begin(9600);	StringReplace.ino - Arduino 1.8.13/examples/08.Strings/StringReplace

```
// the numbers below correspond to the frequencies of middle C, D, E, and F
int notes[] = {262, 294, 330, 349};

void setup() {
  //start serial communication
  Serial.begin(9600);
}

void loop() {
```

v. **Ctrl + Shift + R** = Open Resource (VERY USEFUL! This is how you find and open files!; use * for wildcards); see how to use it here: https://stackoverflow.com/questions/91984/how-do-i-hotkey-directly-to-file-search-tab-in-eclipse/49437058#49437058. Note that you can specify which *working set* you want to search in while using this tool simply by clicking the 3 vertical dots at the top-right of the window, followed by → "Select Working Set…". Working sets were discussed previously in this document.

vi. **Ctrl + E** = "Quick Switch Editor" (jump to another already-open file): a little search box which pops up to search for file names of *all open files shown in the file tabs*, to quickly switch to another one of these open files. I learned about it in this question here while attempting to find a "fuzzy search" utility in Eclipse. You can use the `*` char (asterisk) to act as a wild-card string when searching for file names of open files.

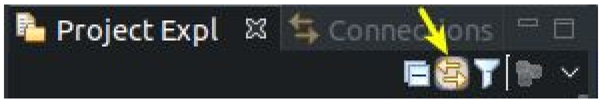
q. **Ctrl + Shift + Y** = change to lowercase (see here)

r. **Ctrl + Shift + X** = change to UPPERCASE (see here)

s. **Right Click** (on variable name, function name, etc)
   i. → Refactor → Rename = rename all usages of this symbol
   ii. → Open Call Hierarchy = show all callers of a given function

t. Eclipse also has a really nice **"Outline" view (sometimes colloquially called a "function list" or "function browser") in the right-hand pane** which shows all symbols, such as functions, classes, variables, enums, etc.

u. "Link with Editor" button:
   i. Near the top of your "Project Explorer" pane on the left is this little "Link with Editor" button:



. It is VERY handy! *Select it to make the Project Explorer file tree automatically jump to whatever file you have open in the Editor*. Deselect it to disable this feature so you can view a file in the Editor and navigate the Project Explorer file tree independently.

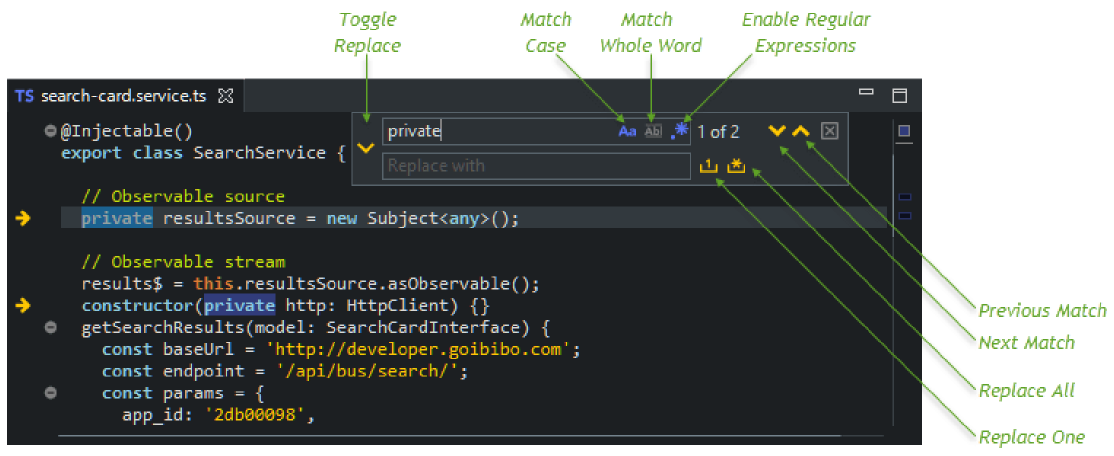
   v. **Multi-view/split view/independent window view:**
      i. To get the view you desire, simply grab a file open in the editor by its tab and drag it to where you want it. Drag to the right to get a vertical split screen view, or drag to the bottom to get a horizontal split view. Once a split view is achieved, you can arbitratrily open files on either side of the view and drag from one view to the other. Dragging a tab entirely away from the application creates a stand-alone "island" type view of this window all by itself, which can be useful to drag a window to another monitor, for instance.
   w. **Window** → **Editor** → **Clone** = clone a copy of your currently-opened file in the editor to a new view so you have two instances of the same file open at once. Doing this then dragging one of the cloned tabs to a split view makes it *extremely easy* to reference and look at one part of a file in one location while working on separate code in the same file at another location.
   x. **File** → **Revert** = revert back to the previously-saved version of your open file by discarding all unsaved changes and reopening the file; useful if you made some changes you do NOT want to keep and have not yet saved, and want to undo them without repeated uses of Ctrl + Z to undo your last actions.

3. EasyShell Plugin Usage:

   a. **Right-click anywhere in an editor or the Project Explorer on a file →EasyShell →**
      i. → Open with default Application = open with your computer's default editor (*ex: Sublime Text 3*, which has excellent multi-cursor editing!)
      ii. → Copy full path to clipboard = self-explanatory; can be really useful when documenting where files are and stuff

4. DevStyle Plugin Usage:

   a. The real benefit DevStyle offers is the ability to set the dark theme and import the Sublime Text 3 syntax highlighting color scheme. However, it also takes over a bunch of other parts of Eclipse and adds some new features, menus, colors and styling, and effects or tools.
   b. Read more from the creator here: https://www.genuitec.com/products/devstyle/
   c. **Ctrl + F** = *DevStyle Inline Search*; VERY HANDY since it quickly highlights all matches of a given string in your open file; read more here, & also see the image below: https://www.genuitec.com/docs/power-ups/using-inline-search/; click the little down-arrow at the left to get a "Replace with" box.



```
@Injectable()
export class SearchService {

  // Observable source
  private resultsSource = new Subject<any>();

  // Observable stream
  results$ = this.resultsSource.asObservable();
  constructor(private http: HttpClient) {}
  getSearchResults(model: SearchCardInterface) {
    const baseUrl = 'http://developer.goibibo.com';
    const endpoint = '/api/bus/search/';
    const params = {
      app_id: '2db00098',
```

i. TO OPEN THE ECLIPSE CLASSIC FIND/REPLACE you must now manually go to Edit → Find/Replace, OR have manually set Eclipse's built-in Find/Replace tool to a new shortcut key (such as Ctrl + G) as described above.

Now you should hopefully have a good glimpse of the power of Eclipse, including its tremendously helpful and powerful indexer, as well as how to set it up and what some of its best shortcuts, plugins, and features are.

Changelog:

(Newest on TOP: YYYYMMDD)

1. 20210131:
   a. ADDED:
      i. Note and link about Ctrl + Shift + R file search in Eclipse NOT having a fancy fuzzy search capability, and maybe requiring an asterisk wildcard before and/or after your search string.
      ii. Note about how to symlink Arduino source code and core code into your project with OS-level symlinks.
2. 20201025:
   a. ADDED:
      i. Link to share this Google Document directly, to top of this document.
      ii. Ctrl + Shift + Alt + L (or: Search → Quick Search) = "Quick Search" tool shortcut.
      iii. Ctrl + E = "Quick Switch Editor" shortcut.
      iv. (I can't remember if it was in this round of changes, but I think it was) info about making Eclipse treat Arduino *.ino files at C++ source files so it will index them as such.

3. 20200428:
    a. ADDED:
        i. More shortcut keys: Ctrl + Shift + L (list shortcuts), Ctrl + Shift + Y (change to lowercase), Ctrl + Shift + X (change to UPPERCASE)
4. 20200412:
    a. CHANGED:
        i. Made “Eclipse Usage, Workflow, Help, Tips & Tricks” main numbered bullets into sub-headings so that they show up in the Table of Contents.
    b. ADDED:
        i. Ctrl + Space autocomplete shortcut key and description.
5. 20200411:
    a. ADDED:
        i. Installation step 14 near the end of “Install & Setup Steps” section which explains how to “tell Eclipse that Arduino `*.ino` files are a type of C++ Source File.” This way Eclipse can be used to edit Arduino source code.
        ii. This Changelog

END.