

Setting Up ROOT (and Geant4) Development Environment on Linux with Eclipse IDE

ROOT and Geant4 frameworks are written in C++, a language with complete manual control over the memory. Therefore, development and execution of your ROOT script (or Geant4 program) may sometimes lead to a crash providing minimal information in the stack trace. ROOT framework does not provide out-of-the-box solutions for debugging scripts. Hence, a question about debugging ROOT scripts now and then arises in the ROOT community.

Generally speaking, one does not need a special development environment to invoke a debugger on a ROOT script. Users can simply invoke the GNU Debugger (GDB) on the debug the root.exe binary:

```
gdb --args root.exe -l -b -q yourRootMacro.C
```

Similarly, GDB can be used for debugging stand-alone ROOT and Geant4-based programs. However, this debugging experience is carried out in the Terminal and lacks user interface and many useful features.

In this article, we outline an **approach for robust debugging of CERN ROOT scripts**, ROOT and Geant4-based programs. We will utilize Eclipse CDT (C/C++ Development Tooling) Integrated Desktop Environment (IDE), a free software coupled with the GNU debugger (GDB):

- Eclipse indexer scans the object-oriented hierarchy of the library classes, allowing easy navigation between C++ sources and headers, quick lookup of method overrides, code highlighting and many more.
- GDB allows pausing program execution at any time. Computer memory, object instances, variable values .

Additionally, the current approach allows users to have ROOT and Geant4 frameworks built in both - Release and Debug modes installed on the same computer. Debug binaries are great for development, allowing memory analysis and efficient development. Release builds - on the other hand - can be optimized for robust execution of the program and may work up to 10 times faster.

A few words about the operating system (OS). In this post, we will consider the setup on Linux-based systems. A similar approach may be replicated to macOS with GNU toolchain, but will require a [code signing procedure](#). Windows is a totally different story.

Following milestones are required to complete the setup of the development environment:

- **Install Eclipse IDE on your computer.** Eclipse is an all-in-one development solution that automates many things: source highlighting and formatting, invokes the CMake build, lets users set breakpoints in code, attaches the debugger to the executable, and many more.
- **Obtain ROOT/Geant4 source code on your computer.** Once attached to the project, this allows easy inspection and navigation between your script (or program) and ROOT/Geant4 source files within the IDE user interface. It also allows modification of the frameworks' source files while debugging your program, which makes it easy to fix bugs and issue Pull Requests to the ROOT/Geant4 open-source code.
- **Compile ROOT/Geant4 with debug symbols.** This provides the ability to set up breakpoints in your code and original ROOT (or Geant4) source files, inspect variables, access data types and object members in the program source code.
- **Transform ROOT script into a standalone program.** ROOT scripts designed to run with C++ interpreter (executed line-by-line), need to be transformed into a compiled C++ program - with an entry point in "main()" function.
- **Set up your ROOT-based program in Eclipse IDE.** During the main project setup, ROOT and Geant4 projects are marked as its references. This automatically triggers the rebuild and re-install of corresponding ROOT (and Geant4) components prior to the main project build. Additionally, ROOT and Geant4 indexer databases become shared with your project.

Eclipse IDE Setup

In this section we demonstrate how to install Eclipse IDE on a personal Linux computer. We will use Eclipse with the CMake4eclipse plugin - a powerful tool for use with CMake-based projects. Cmake4eclipse automates the project setup and allows for automatic rebuild of the frameworks' libraries (ROOT and Geant4) once their source code was changed.

Today (July 2022) **CMake4eclipse plugin** provides better integration of CMake-based projects in Eclipse compared to other options e.g:

- Using CMake generator to create Eclipse project.
- Using the built-in Eclipse wizard to import an existing CMake project.

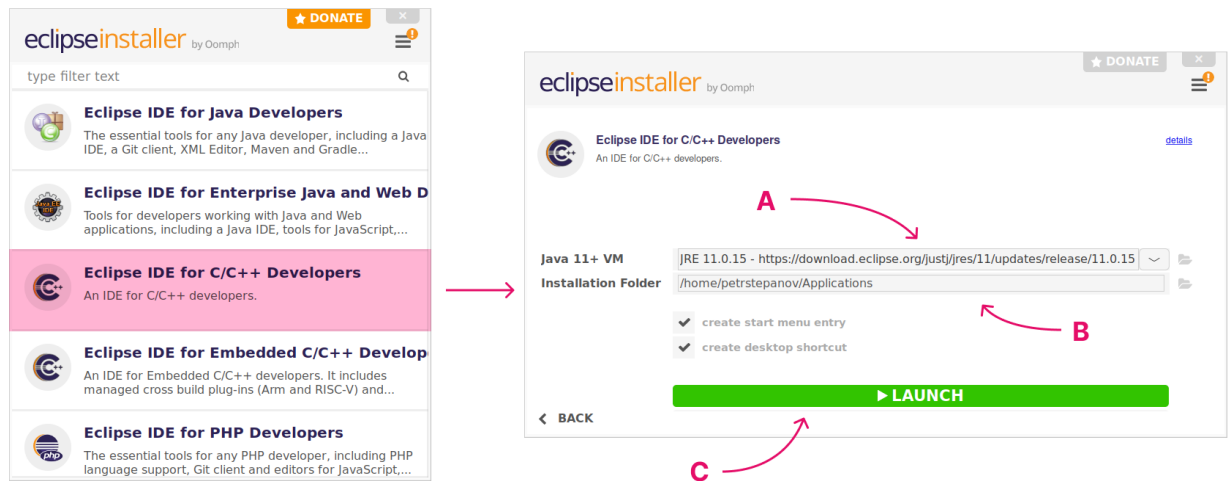
Each of above options have its own downsides that are a subject of a separate discussion.

Following steps are required to set up the IDE workflow.

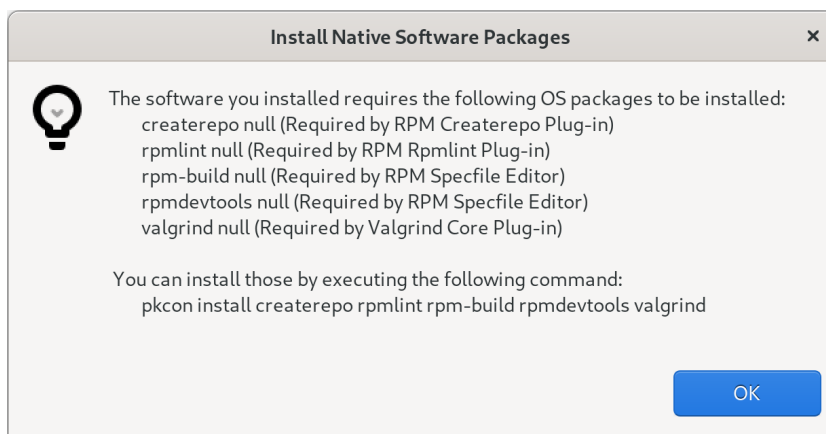
- 1) **Install Eclipse IDE.** Download the Eclipse installer from the [official website](#), extract it and run. Select "Eclipse IDE for C/C++ Developers". Refer to the screenshot and instructions below:

A. Recent Eclipse versions come with bundled Java Runtime Environment (JRE). As of July 2022, **specify the built-in JRE version 11**. Otherwise there will be an error accessing Eclipse help. This may be fixed in later Eclipse releases.

B. On Linux it is a good practice to install software that is not included in your distribution under the “/opt”, “/usr/local/” folder or home folder. In this article we will stick to the latter option and **install Eclipse in the home folder** under “~/Applications/” for consistency with the setup on macOS.



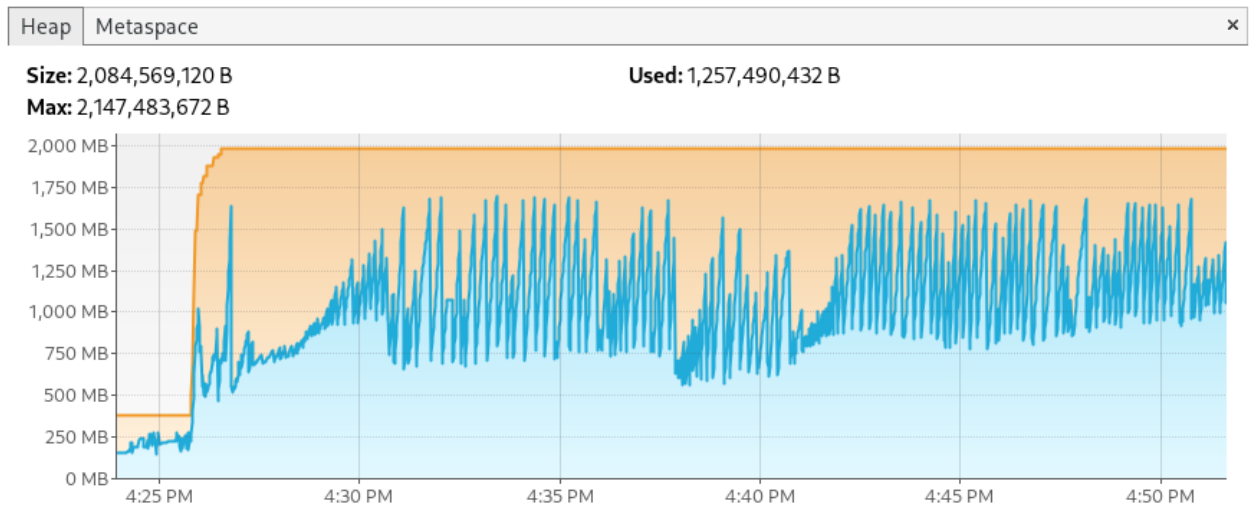
C. Wizard will provide a required list of packages to be installed on your system. Ensure all of the package dependencies are installed on your system.



D. Exit the wizard. There is no need to launch Eclipse right away. We will tweak its configuration file first.

- 2) **Increase Eclipse memory limits.** ROOT and Geant4 libraries contain thousands of source files. Usually, when indexing a ROOT and Geant4 based project, memory use fluctuates

around 2GB. This can be observed via the [VisualVM application](#).



Memory limits are specified in the “eclipse.ini” file located inside the Eclipse install folder. Use text editor to update following lines:

```
-Xms512m  
-Xmx4096m (set to 2048m minimum or higher if available)
```

Here the “-Xms” value corresponds to the initial heap size used at the Eclipse startup. The latter “-Xmx” value is to the maximum available memory limit. It is reasonable to set the “-Xmx” value to about half of the Random Access Memory (RAM) installed on the computer. Indexing speed of the ROOT and Geant4 framework source files will be higher with more available RAM.

- 3) **Fix Eclipse launcher** to comply with the FreeDesktop specifications (bug report submitted [here](#)).

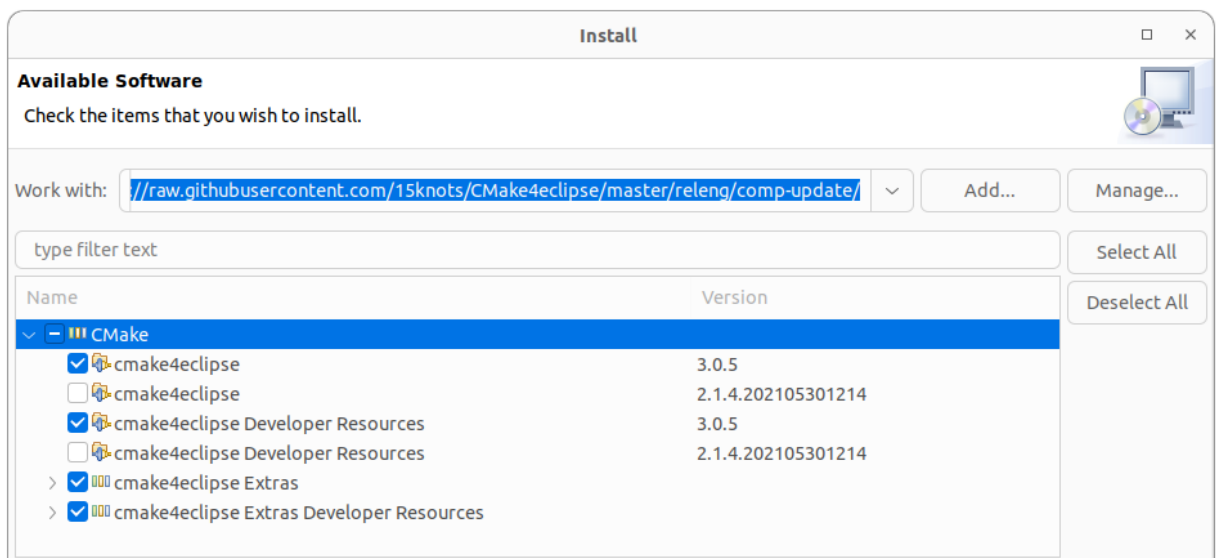
```
echo StartupWMClass=Eclipse >>  
~/.local/share/applications/epp.package.cpp.desktop
```

- 4) **Tweak memory limit for Eclipse indexer.** Launch Eclipse and select default workspace location (e.g. ~Development/eclipse-workspace). In the Eclipse menu open Window → Preferences → C/C++ → Indexer. Under “Cache Limits” set:

Limit relative to maximum heap size: 75%
Absolute limit: 4096 MB (same as for -Xmx value in eclipse.ini)

- 5) **Update Eclipse** and its CDT plugin. In the menu select Help → Check for updates. Follow the wizard steps. Restart Eclipse if required.
- 6) **Install CMake4eclipse plugin**. Project details can be found [on GitHub](https://github.com/15knots/CMake4eclipse). In the Eclipse menu select Help → Install new software. Enter following URL in the “Work with” field:
<https://raw.githubusercontent.com/15knots/CMake4eclipse/master/releng/comp-update/>

In the modal dialog select everything but uncheck the older version of CMake4eclipse (v2). Keep only version v3. Follow the wizard steps and restart Eclipse. Refer to the screenshot below:



- 7) **Tweak cmake4eclipse settings**. Set default workbench for CMake4eclipse. In the Eclipse menu select Window → Preferences → C/C++ → Cmake4eclipse → Default build system → Set "**Unix Makefiles**".

Additionally, in the same dialog check "**Force re-creation with each build**". This will trigger the CMakeLists.txt update and re-generation of the Unix makefile at every build, reflecting possible changes in the CMake cache entries (variables) and ROOT components' source code.

Optionally apply following tweaks to the Eclipse user interface:

- **Hide the Launch Bar**. It indeed takes quite some space on smaller screens. In the Eclipse menu go to: Window → Preferences → Run/Debug → Launching → Launch Bar. Uncheck "Enable the Launch Bar".
- **Display line numbers**. In the Eclipse menu go to: Window → Preferences → General → Editors → Text Editors. Check "Show line numbers".

We successfully installed and set up the Eclipse with CMake4eclipse plugin and are now ready to set up ROOT and Geant4 projects in Eclipse IDE.

Building ROOT with Debug Symbols

In this section we address the setup of ROOT libraries as a project in Eclipse IDE. Framework will be built with debug symbols. This allows for setting breakpoints in the ROOT code, inspecting memory and variable values during the program run.

- 1) **Install dependencies.** Refer to [this page on ROOT website](#) to satisfy the dependencies for your system. Below we copy the list of dependencies for ROOT v6.27 for popular linux distributions. In Terminal execute following two commands:

<p>For Ubuntu 22.04 LTS:</p> <pre>sudo apt -y install build-essential sudo apt -y install git dpkg-dev cmake g++ gcc binutils libx11-dev libxpm-dev libxft-dev libxext-dev python3 libssl-dev sudo apt -y install gfortran libpcre3-dev xlibmesa-glu-dev libglew1.5-dev libftgl-dev libmysqlclient-dev libfftw3-dev libcfitsio-dev graphviz-dev libavahi-compat-libdnssd-dev libldap2-dev python3-dev libxml2-dev libkrb5-dev libgsl0-dev qtwebengine5-dev</pre>	<p>For Fedora 36:</p> <pre>sudo dnf -y groupinstall "Development Tools" "Development Libraries" sudo dnf -y install git cmake3 gcc-c++ gcc binutils libX11-devel libXpm-devel libXft-devel libXext-devel python openssl-devel sudo dnf -y install redhat-lsb-core gcc-gfortran pcre-devel mesa-libGL-devel mesa-libGLU-devel glew-devel ftgl-devel mysql-devel fftw-devel cfitsio-devel graphviz-devel avahi-compat-libdns_sd-devel openldap-devel python-devel python3-numpy libxml2-devel gsl-devel libuuid-devel readline-devel R-devel R-Rcpp-devel R-RInside-devel</pre>
--	---

- 2) **Obtain the source code.** There are a few options here.

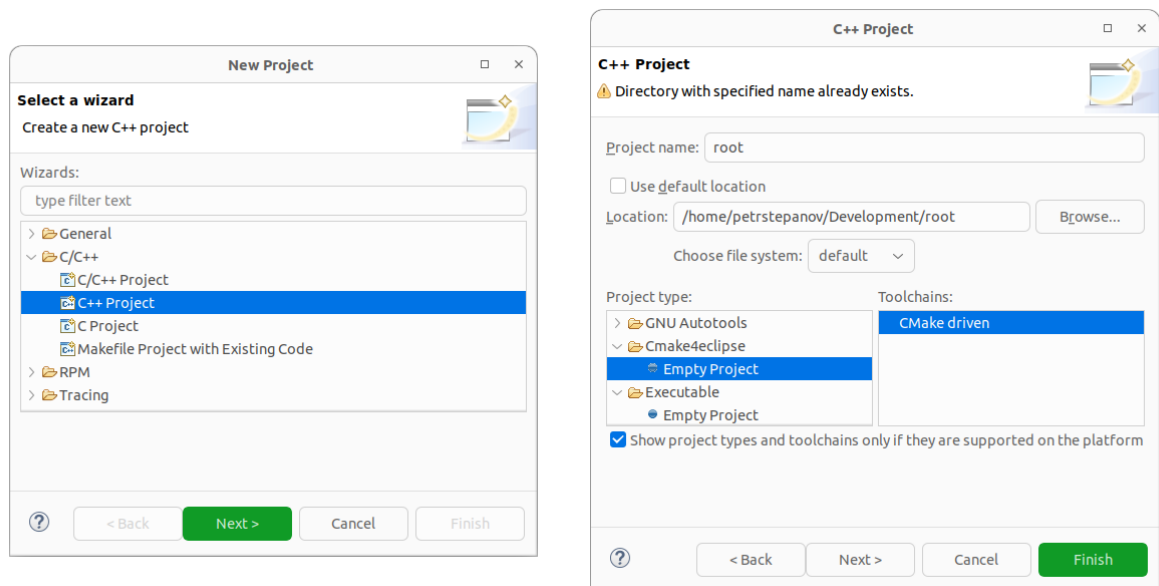
The straightforward way is downloading ROOT sources for a specific release [from the ROOT website](#). Extract ROOT sources under the "~/Development" home folder. We will keep all the source code and Git repositories in this folder for consistency purposes.

Alternatively, if a user plans on contributing towards the ROOT repository it is recommended to fork the latest “master” branch [on GitHub](#), create a new branch in your forked repository and check it out:

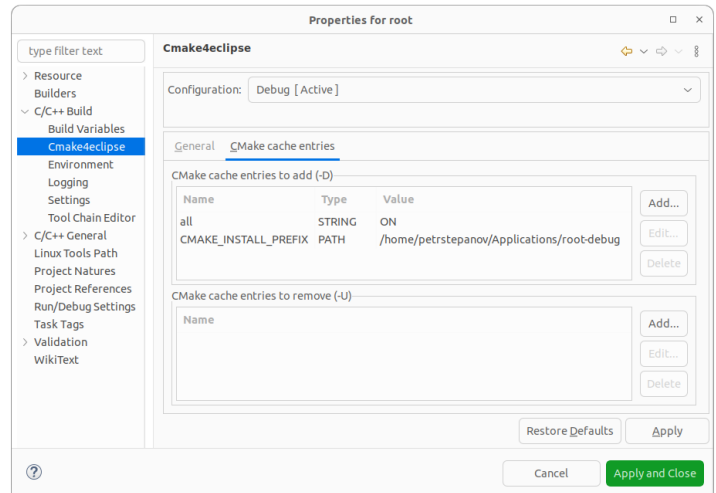
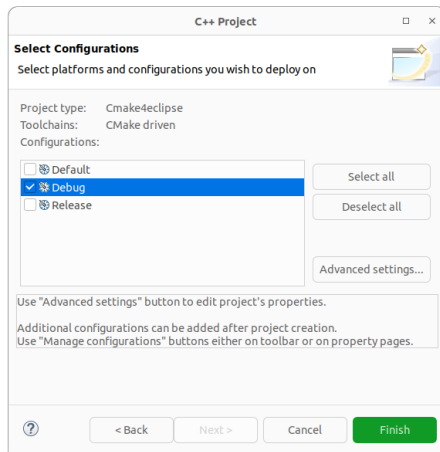
```
mkdir -p ~/Development && cd ~/Development
git clone https://github.com/<your-username>/root
git checkout -b <your-feature-branch>
```

This allows for issuing Pull Requests to the original repository. More details can be found [on ROOT website](#).

- 3) **Set up a project in Eclipse.** Launch Eclipse. In the menu open File → New → Project... Expand "C/C++" and select "C++ Project" (not "C/C++ Project").



On the next dialog, specify “root” as the project name. Uncheck “Use default location” and “Browse...” for ROOT sources location (e.g. ~/Development/root). In “Project Type” expand “Cmake4eclipse” and select “Empty Project”. In “Toolchains” select “CMake driven”. Click “Next >”.

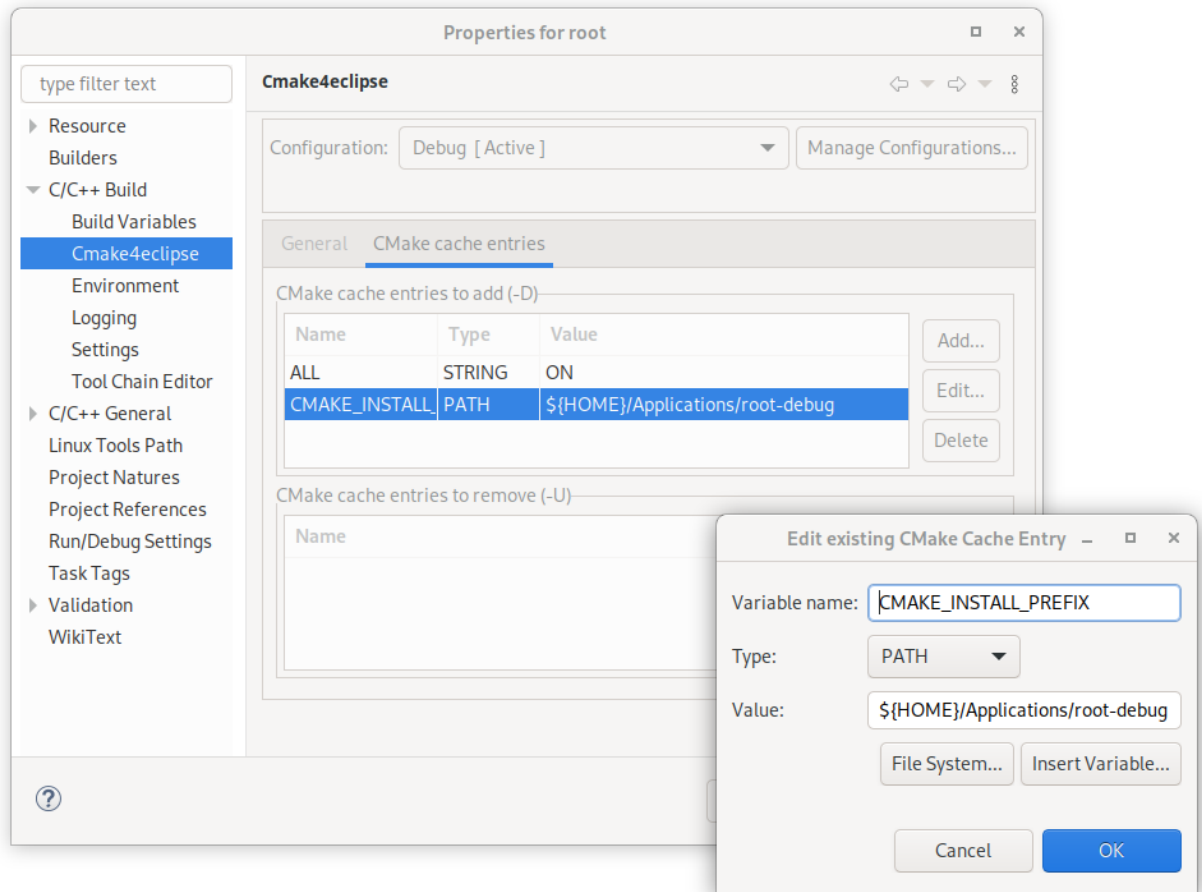


We are building ROOT with debug symbols. Therefore, uncheck "Default" and "Release" build options and only keep the "Debug". Essentially this dialog box specifies the CMake -DCMAKE_BUILT_TYPE variable.

Next we provide the CMake plugin with ROOT build options. Click "Advanced Settings...". Go to C/C++ Build → Cmake4eclipse. Open the "CMake cache entries" tab. Add following variable names and values. Use "Add..." button on the right to input following variable names, types and values:

Name	Type	Value
CMAKE_INSTALL_PREFIX	PATH	\${HOME}/Applications/root-debug
All	BOOL	ON
CMAKE_CXX_STANDARD	STRING	17

When specifying variables of a PATH type, it is handy to use the "File System..." button. It will display the folder picker dialog and minimize the chance of specifying a wrong path. Refer to the screenshot below:



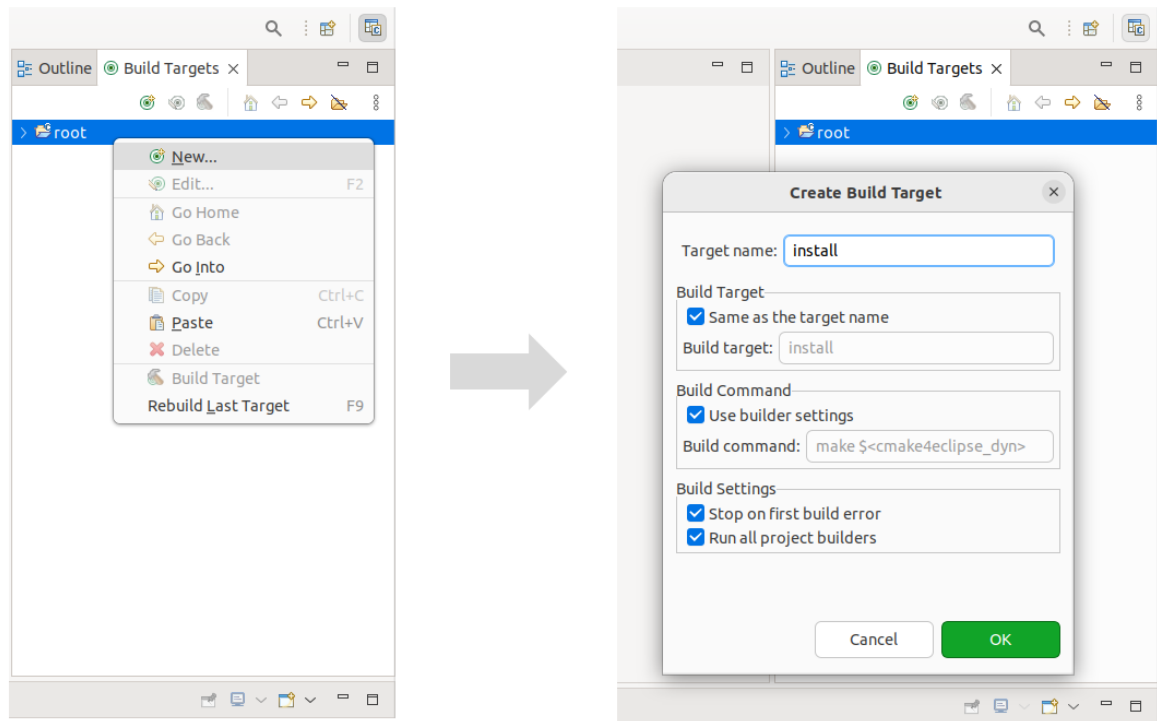
In this tutorial we build ROOT with all optional components turned on (-Dall=ON). Please find a complete list of the [ROOT CMake build variables](#) on the ROOT website and tailor the build for your needs.

It is important to keep in mind that ROOT-based programs need to be compiled with the same C++ standard as the ROOT libraries. Therefore, in this guide we will explicitly set the C++ standard for ROOT, Geant4 and their based programs. This is achieved via setting the CMAKE_CXX_STANDARD CMake variable. [More info here](#).

Click "Apply and Close". Click "Finish".

Notice that Eclipse will start indexing the project. However, we will reschedule this operation after the build is completed. Reveal the "Progress" panel (tiny scrollbar animation in the very bottom right corner). Stop the indexer operation.

- 4) **Build framework in Eclipse.** Reveal the "Build Targets" tab (on the right side) and select "root" project. Right-click and select "New..." build target. Name target "install". Click "Ok". Expand "root" in the "Build Targets" tab and double-click the "install" target.



Build process speed depends on your computer speed and provided build variables. It may take up to a few hours to finish.

Tip: to switch between the CMake console and Linux make console, locate the “Display Selected Console” dropdown on the bottom actions panel.



- 5) **Exclude build folder from indexing.** Cmake4eclipse plugin performs a so-called in-source build. Meaning that the build folder is located within a project file tree. During the build ROOT header files are copied and duplicated inside the “_build” folder. To avoid indexing duplicate sources and headers, right click “root” project → Properties → C/C++ General → Paths and Symbols → Source Location. Expand the “/root” folder. Select “Filter”. Click “Edit filter...”. Add CMake4eclipse “_build” folder to the filter. Click “Apply and

Close”.

- 6) **Run Eclipse indexer.** We are now ready to index all ROOT source files and headers. This will create an Object-Oriented Programming (OOP) database of all ROOT object types, their methods and inheritance relations. Right click “root” project → Index → Rebuild.

Tip 1: sometimes Eclipse indexer may freeze while parsing the “./interpreter/...” sub-folders. If this happens, exclude the “interpreter” folder from the build (this also excludes folders from the index). Highlight the “interpreter” folder in the project tree. Right click, and select Resource Configurations → Exclude from build... Check “Debug” configuration. Click “Ok”. Now right click “root” project → Index → Rebuild.

Tip 2: Indexer usually takes about an hour or two to parse all of the ROOT framework source files. Computers with fast NVMe hard drives will perform this task the best. For older computers I strongly recommend keeping ROOT (and Geant4) sources on the RAMDisk. Please refer to my RAMDisk implementation [on GitHub](#).

Apparently, if users want to issue pull requests to the ROOT GitHub repository, the CMake4eclipse “_build” child folder needs to be added to the “.gitignore” of the local ROOT Git tree.

At this point ROOT libraries are compiled with debug symbols and Eclipse has indexed all the framework source files.

Building Geant4 with Debug Symbols

In this section we will set up Geant4 as an Eclipse project. Skip to the next section if Geant4 install is not required. Generally speaking, the process is similar to the ROOT install since both frameworks use the CMake build system.

- 1) **Install dependencies.** Refer to [the Geant4 documentation](#) to satisfy its dependencies for your system. Below we provide the list of dependencies for Geant4 v10 for popular linux distributions. These packages were tested on top of the ROOT installation. Therefore make sure to install ROOT dependencies from the previous section first.

For Ubuntu 22.04 LTS	For Fedora 36
<pre>sudo apt install -y qt5* libX11* libXmu* libmotif* expat* libxerces-c*</pre>	<pre>sudo dnf -y install qt5* libX11* libXmu* motif* expat* mesa-libGL-devel xerces-c*</pre>

- 2) **Obtain the source code.** Download required Geant4 version [from official website](#) or clone recent codebase [from GitHub](#) under the “~/Development/geant4” folder. Please note that compared to ROOT (better backward compatibility) there is a higher chance that an existing Geant4-based project may require a specific Geant4 version.
- 3) **Set up a project in Eclipse.** Similarly to the ROOT project setup, in the Eclipse menu open File → New → Project... Expand "C/C++" and select "C++ Project" (not "C/C++ Project").

On the next dialog, specify “geant4” as the project name. Uncheck “Use default location” and “Browse...” for Geant4 sources location (e.g. ~/Development/geant4). In “Project Type” expand “Cmake4eclipse” and select “Empty Project”. In “Toolchains” select “CMake driven”. Click “Next >”.

Next we provide the CMake plugin with Geant4 build options. Click “Advanced Settings...” and navigate to C/C++ Build → Cmake4eclipse. Open the “CMake cache entries” tab. Full list of build variables and their descriptions can be found [on the Geant4 website](#). We will specify a common set of CMake variables that works for most projects:

Name	Type	Value
CMAKE_INSTALL_PREFIX	PATH	\${HOME}/Applications/geant4-debug
CMAKE_CXX_STANDARD	STRING	17
GEANT4_BUILD_MULTITHREADED	BOOL	ON
GEANT4_INSTALL_DATA	BOOL	ON
GEANT4_USE_GDML	BOOL	ON
GEANT4_USE_OPENGL_X11	BOOL	ON
GEANT4_USE_QT	BOOL	ON
GEANT4_USE_XM	BOOL	ON

Please make sure that the Geant4 is built with the same C++ standard as the ROOT version earlier in this guide. Here, we are using the [C++17](#) standard.

Click "Apply and Close". Click "Finish".

Notice that Eclipse will start indexing the project. Reveal the “Progress” panel and stop the indexer operation.

- 4) **Build framework in Eclipse.** Reveal the “Build Targets” tab (on the right side) and select "geant4t" project. Right-click and select “New...” build target. Name target "install". Click "Ok". Expand “root” in the “Build Targets” tab and double-click the "install" target.

- 5) **Exclude build folder from indexing.** Right click "geant4" project → Properties → C/C++ General → Paths and Symbols → Source Location. Expand the "/geant4" folder. Select "Filter". Click "Edit filter...". Add CMake4eclipse "_build" folder to the filter. Apply changes.
- 6) **Run Eclipse indexer.** Right click "geant4" project → Index → Rebuild. Please note that the indexer will take a while to finish.

Now Geant4 libraries are compiled with debug symbols and Eclipse has indexed all the framework source files.

Specifying Environment Variables for Eclipse

In previous sections we compiled ROOT and Geant4 libraries with debug symbols and installed their binaries under the ~/Applications folder. However, we did not set up the required environment variables for the frameworks. Usually, one would source the environment in ~/.bashrc file:

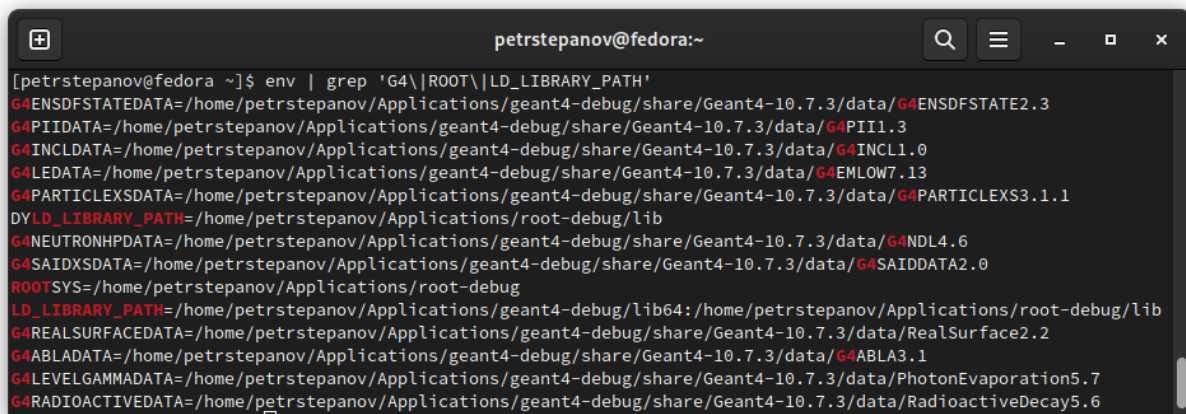
```
source $HOME/Applications/bin/thisroot.sh
source $HOME/Applications/bin/geant4.sh
```

However, we are using a different approach. We built ROOT and Geant4 frameworks with debug symbols (-DCMAKE_BUILD_TYPE=Debug) are executed notably slower compared to the release (-DCMAKE_BUILD_TYPE=Release) versions. Users may want to have faster release versions of these frameworks installed on a local computer (possibly from system repositories) along with the debug build set up in the IDE.

To isolate ROOT and Geant4 frameworks built with debug symbols from potentially system-wide installed release versions we will source the development environment in a separate shell process where Eclipse is launched. There are a few ways to do this.

First option. Source ROOT and/or Geant4 variables in Terminal and manually plug them into the Eclipse settings. Open Terminal and execute following commands:

```
source $HOME/Applications/root-debug/bin/thisroot.sh
source $HOME/Applications/geant4-debug/bin/geant4.sh
env | grep 'G4\|ROOT\|LD_LIBRARY_PATH'
```

A terminal window titled 'petrstepanov@fedora:~' with search, menu, and window control icons. It displays the output of the command 'env | grep 'G4\|ROOT\|LD_LIBRARY_PATH'', showing various environment variables for Geant4 and ROOT. The variables include paths to data files and libraries for different Geant4 processes like G4ENSDFSTATE2.3, G4PII1.3, G4INCL1.0, G4LELOW7.13, G4PARTICLEXS3.1.1, G4NDL4.6, G4SAIDDATA2.0, G4REALSURFACE2.2, G4ABLA3.1, G4LEVELGAMMADATA, and G4RADIOACTIVEDATA. The LD_LIBRARY_PATH is set to include /home/petrstepanov/Applications/geant4-debug/lib64 and /home/petrstepanov/Applications/root-debug/lib.

```
[petrstepanov@fedora ~]$ env | grep 'G4\|ROOT\|LD_LIBRARY_PATH'
G4ENSDFSTATEDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4ENSDFSTATE2.3
G4PIIDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4PII1.3
G4INCLDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4INCL1.0
G4LEDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4EMLOW7.13
G4PARTICLEXSDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4PARTICLEXS3.1.1
DYLD_LIBRARY_PATH=/home/petrstepanov/Applications/root-debug/lib
G4NEUTRONHPDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4NDL4.6
G4SAIDXSDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4SAIDDATA2.0
ROOTSYS=/home/petrstepanov/Applications/root-debug
LD_LIBRARY_PATH=/home/petrstepanov/Applications/geant4-debug/lib64:/home/petrstepanov/Applications/root-debug/lib
G4REALSURFACEDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/RealSurface2.2
G4ABLADATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/G4ABLA3.1
G4LEVELGAMMADATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/PhotonEvaporation5.7
G4RADIOACTIVEDATA=/home/petrstepanov/Applications/geant4-debug/share/Geant4-10.7.3/data/RadioactiveDecay5.6
```

Required environment variables are output in the Terminal window. Now in the Eclipse menu open: Window → Preferences → C/C++ → Build → Environment. Manually “Add...” each environment variable’s name and value into the Eclipse Environment settings.

Second option. I consider this more elegant. We will modify the Eclipse launcher. Locate the Eclipse *.desktop launcher located in “~/.local/share/applications” folder and modify the “Exec=...” line as follows:

```
Exec=bash -c "source \\$HOME/Applications/root-debug/bin/thisroot.sh && source \\$HOME/Applications/geant4-debug/bin/geant4.sh && \\$HOME/Applications/eclipse/eclipse"
```

Please note that the paths above may need to be modified depending on your particular locations of ROOT, Geant4 frameworks and Eclipse application accordingly. Eclipse launcher database needs to be updated (alternatively, re-login into your session):

```
xdg-desktop-menu forceupdate
```

Now Eclipse will start in a separate shell process with ROOT (and Geant4) environment variables specified for the build with debug symbols.

The beauty of this method is that the development environment variables in Eclipse are isolated from the system-wide environment variables.

Convert ROOT script into CMake ROOT-based Program

ROOT scripts are originally designed to run through the Cling, a modern C++ interpreter based on the LLVM and Clang. To debug a ROOT script with native Linux GNU development tools - "gcc" compiler and "gdb" debugger - we need to convert a ROOT script into a ROOT-based program. Having it compiled into an executable with debug symbols, we will be able to invoke a debugger on it.

Skip to the next section if you already have a ROOT- or Geant4-based program code ready.

Next we elaborate how to convert a ROOT script into a CMake ROOT-based program. Generally speaking, this involves following:

1. Compose CMakeLists.txt file containing a set of instructions that:
 - Locate necessary libraries (ROOT, Geant4).
 - Find and compile source and header files in your project.
 - Create a ROOT dictionary and shared library for your program.
 - Link all object files and shared libraries into an executable.
2. Explicitly define all the headers used in your script (Cling interpreter does not require that).
3. If using the Object-Oriented-Programming (OOP) approach, certain class names need to be listed as directives in a special "LinkDef.h" file for the dictionary and shared library generation.

A detailed instructions elaborating each item above can be found in this [template repository on GitHub](#). Please refer to the repository README file.

Setting up a ROOT or Geant4-Based CMake Program

In this section we will set up a ROOT or Geant4-based CMake project in Eclipse IDE. This applies to stand-alone ROOT-based projects, custom Geant4 programs and examples.

All Geant4 examples are CMake based. Therefore this section is useful for the users that start learning Geant4 and may want to compile and debug Geant4 examples in Eclipse IDE.

- 1) **Obtain the source code.** Place your ROOT or Geant4-based project (Geant4 example folder) into a desired location, e.g. ~/Development.
- 2) **Set up a project in Eclipse.** Similarly to the ROOT project setup, in the Eclipse menu open File → New → Project... Expand "C/C++" and select "C++ Project" (not "C/C++ Project").

On the next dialog, specify your project name. Uncheck "Use default location" and "Browse..." for your project "CMakeLists.txt" location. In "Project Type" expand "Cmake4eclipse" and select "Empty Project". In "Toolchains" select "CMake driven". Click "Next >".

For the development purpose we uncheck "Default" and "Release" build configurations, keep the "Debug" option only.

Next we need to provide the CMake plugin with Geant4 build variables. Click "Advanced Settings...". Go to C/C++ Build → Cmake4eclipse. Open the "CMake cache entries" tab.

Since we compiled and installed ROOT and Geant4 not system-wide, but in ~/Applications home directory, we need to provide the CMake with the locations of "RootConfig.cmake" and "Geant4Config.cmake" files respectively. Use "Add..." button on the right to input following variable names, types and values:

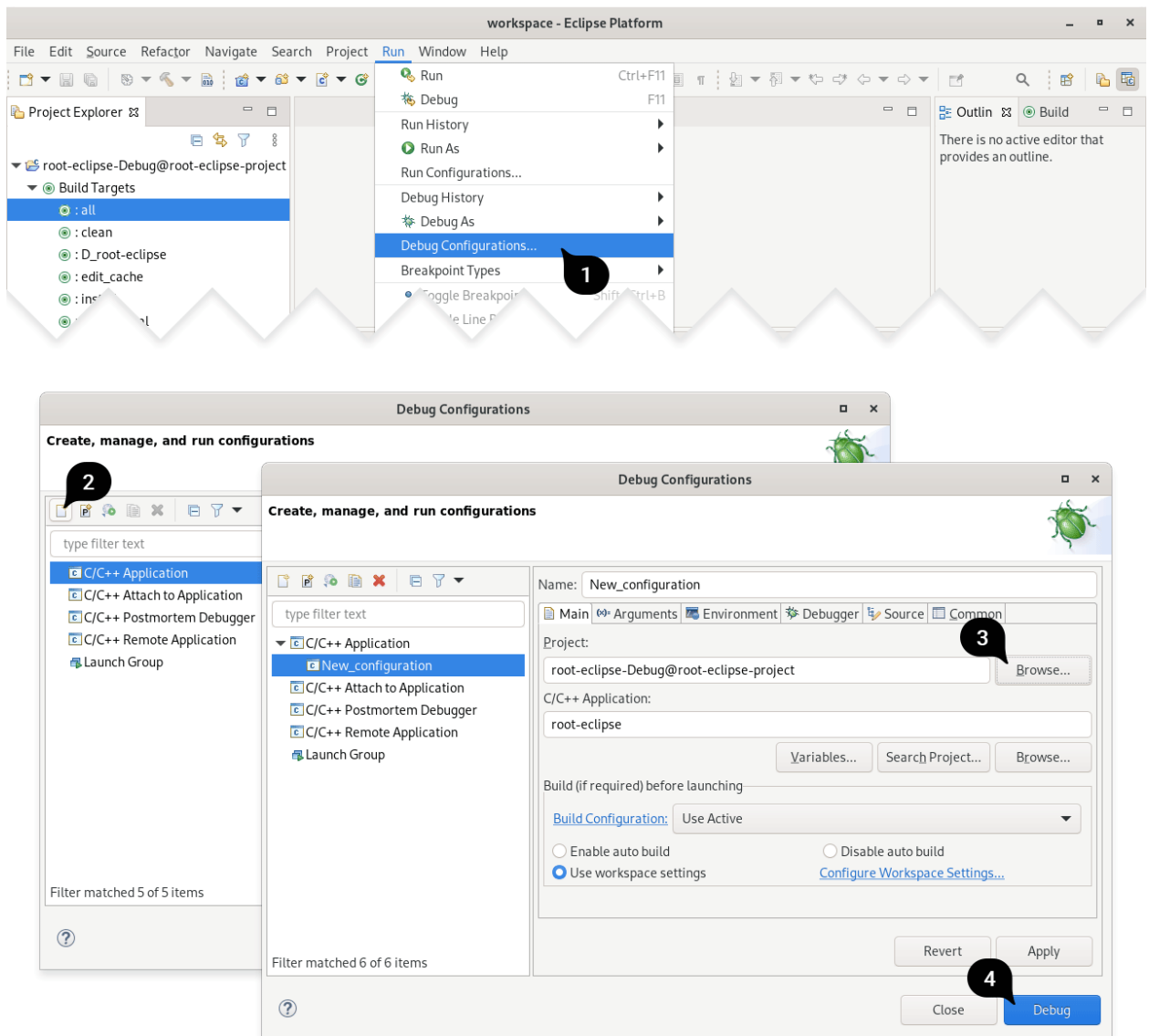
Name	Type	Value
CMAKE_CXX_STANDARD	STRING	17
ROOT_DIR	PATH	\${HOME}/Applications/root-debug/cmake
Geant4_DIR	PATH	\${HOME}/Applications/geant4-debug/CMake/lib64/Geant4-10.7.3

Make sure that the paths specified above are accurate. Some path discrepancies across frameworks' versions may occur. Also, ensure the consistency in C++ standard with your program and ROOT (and Geant4) versions used.

- 3) **Build project in Eclipse.** Highlight your project in Project Explorer. Right click → build.
- 4) **Reference ROOT and/or Geant4 project.** Select your project in Project Explorer. Right click properties → Project References → Check "root" and/or "geant4", depending on if your project requires . This allows following:
 - Sharing ROOT and/or Geant4 indexer database with your project.
 - Rebuild of ROOT and/or Geant4 libraries prior to your project build in case any of ROOT and/or Geant4 source files were changed.
- 5) **Create a run (debug) configuration.** Select your project in the project tree. In the Eclipse menu, open Run → Debug Configurations...

Select "C/C++ Application". Press the "New launch configuration" button (on the very top

left). Click the “Search Project...” button and locate the corresponding executable file.



If necessary, specify any command-line parameters (e.g. macro files for Geant4 etc) on the “Arguments” tab. Click “Debug”.

This is it. Now you can enjoy full-scale debugging of your ROOT or Geant4 based applications in Eclipse IDE.

Appendix. Geant4- and ROOT-based program with nested CMake structure

Geant4 programs may often refer to some ROOT data types especially when saving output data in ROOT files. It is a good practice for a Geant4-based standalone application to have a nested CMake structure including some ROOT scripts or ROOT-based programs that analyze and plot Geant4 output data. Therefore, CMake needs to locate both frameworks.

An example Geant4 project with ROOT scripts visualizing the output data located in a child project folder is implemented in [this repository on GitHub](#).