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. Eclipse coupled with GDB provides enhanced code development infrastructure. 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. This functionality makes debugging the experience of your ROOT script or stand-alone program very efficient.

Additionally, the current approach allows users to have ROOT and Geant4 frameworks built in both - Release and Debug modes installed on the same computer for the same user. Debug binaries are great for development, but Release versions of the frameworks installed along with Debug versions are best at running programs with high time and space complexity.

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 different story.

Current approach outlines a setup of an extensive debugging environment for a ROOT script or ROOT/Geant4-based program. We will cover following topics:

- **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.
- **Get a copy of the 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.
- **Convert your ROOT script to a standalone CMake-based C++ project**. Once properly set up in Eclipse, existing ROOT and Geant4 Eclipse projects are referenced in your Eclipse project. This allows for re-building necessary ROOT and Geant4 source files before invoking your project build as well as sharing their indexer databases 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.

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

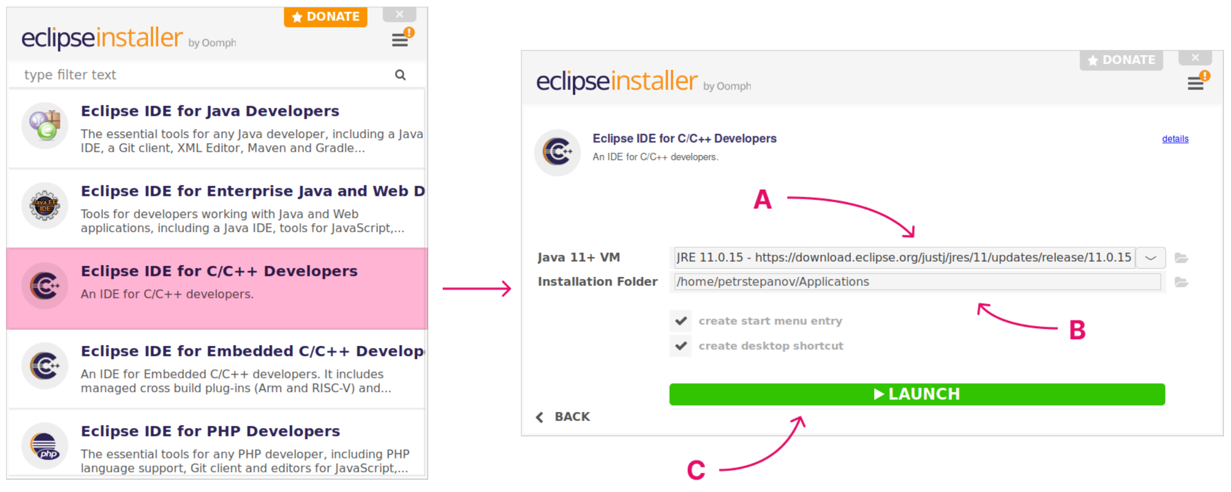
- Using CMake generator to create Eclipse project.
- Using the built-in Eclipse wizard to import an existing CMake project.

Each of these two options have its own downsides that are a subject of a separate discussion. In this article we will outline an approach featuring the CMake4eclipse plugin. For simplicity, we will enumerate the install steps below.

1) **Install Eclipse IDE**. Download Eclipse installer from the official website here, 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 user-specific applications under the "/opt" folder or home folder. In this article we will stick to the latter option and install Eclipse in the “~/Applications/” home folder for consistency with the setup on macOS.



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle...

Eclipse IDE for Enterprise Java and Web D
Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript,...

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

Eclipse IDE for Embedded C/C++ Develop
An IDE for Embedded C/C++ developers. It includes managed cross build plug-ins (Arm and RISC-V) and...

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client and editors for JavaScript,...

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

Java 11+ VM: JRE 11.0.15 - https://download.eclipse.org/justj/jres/11/updates/release/11.0.15
Installation Folder: /home/petrstepanov/Applications

C. 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 1-2GB. 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 half of your computer RAM)

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.

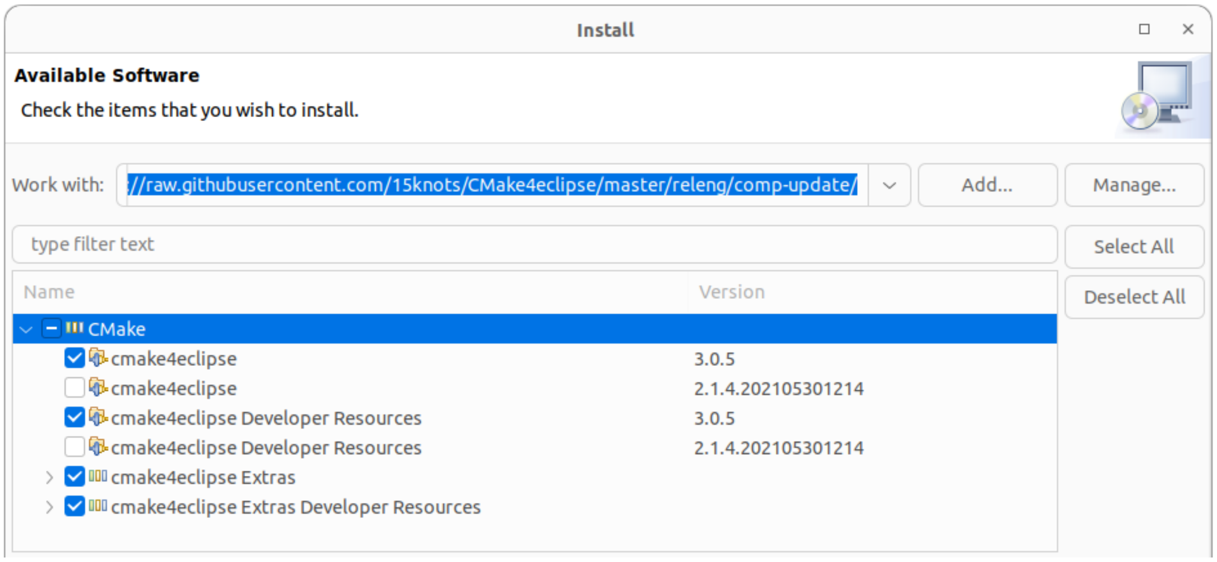
3) **Tweak memory limit for Eclipse indexer**. Launch Eclipse and select default workspace location (e.g. ~Development/eclipse-workspace). In the Eclipse menu open Window → Preferences → C/C++ → Indexer. Under “Cache Limits” set:

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

4) **Update Eclipse** and its CDT plugin. In the menu select Help → Check for updates. Follow the wizard steps.

5) **Install CMake4eclipse plugin**. Project details can be found on GitHub. 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 uncheck the older version of CMake4eclipse (v2). Keep only version v3. Follow the wizard steps and restart Eclipse. Refer to the screenshot below:



Work with: //raw.githubusercontent.com/15knots/CMake4eclipse/master/releng/comp-update/

Name	Version
CMake	
✓ cmake4eclipse	3.0.5
☐ cmake4eclipse	2.1.4.202105301214
✓ cmake4eclipse Developer Resources	3.0.5
☐ cmake4eclipse Developer Resources	2.1.4.202105301214
✓ cmake4eclipse Extras	
✓ cmake4eclipse Extras Developer Resources	

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

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**. 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.

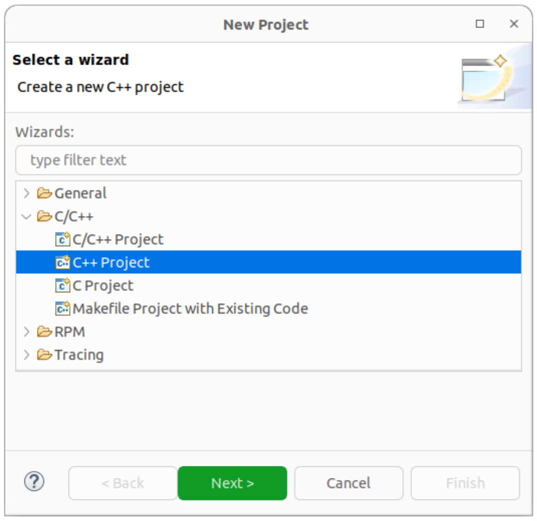
1) **Install dependencies**. Refer to this page on ROOT website to satisfy the dependencies for your system. Below we copy the list of dependencies for ROOT v6.27 for popular linux distributions. In Terminal execute following two commands:

For Ubuntu 22.04 LTS:	For Fedora 36:
`sudo apt -y install build-essential`	`sudo dnf -y groupinstall "Development Tools" "Development Libraries"`
`sudo apt -y install git dpkg-dev cmake g++ gcc binutils libx11-dev libxpm-dev libxft-dev libxext-dev python3 libssl-dev`	`sudo dnf -y install git cmake3 gcc-c++ gcc binutils libX11-devel libXpm-devel libXft-devel libXext-devel python openssl-devel`
`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`	`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`

2) **Obtain the source code**. I recommend cloning the latest ROOT from Git unless for some reason the user needs a particular version. This provides the most up-to-date code with all patches. In this guide we will keep all the source code and Git repositories under the "~/Development" home folder. In Terminal run following commands:

```
mkdir -p ~/Development && cd ~/Development
git clone https://github.com/root-project/root
```

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").

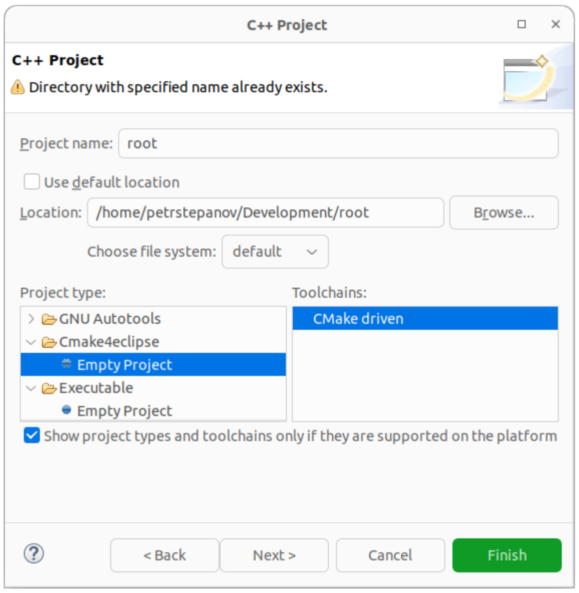

New Project

Select a wizard

Create a new C++ project

Wizards: type filter text

- General
- C/C++
  - C/C++ Project
  - C++ Project
  - C Project
  - Makefile Project with Existing Code
- RPM
- Tracing


C++ Project

C++ Project

Directory with specified name already exists.

Project name: root

Use default location

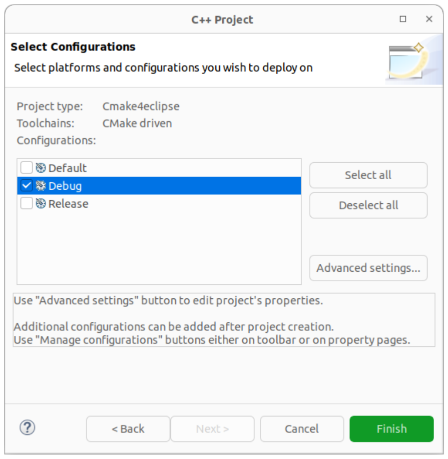
Location: /home/petrstepanov/Development/root

Choose file system: default

Project type:	Toolchains:
GNU Autotools	CMake driven
Cmake4eclipse	
Empty Project	
Executable	
Empty Project	

Show project types and toolchains only if they are supported on the platform

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 >”.


C++ Project

Select Configurations

Select platforms and configurations you wish to deploy on

Project type: Cmake4eclipse

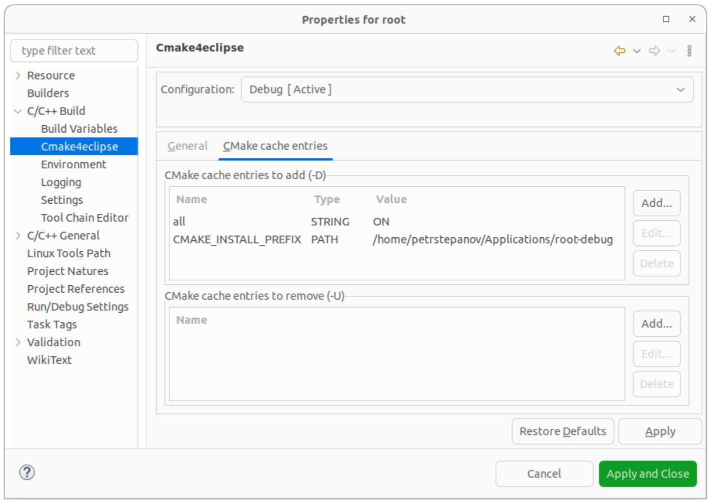
Toolchains: CMake driven

Configurations:

- Default
- 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.


Properties for root

Cmake4eclipse

Configuration: Debug [Active]

General | CMake cache entries

CMake cache entries to add (-D)

Name	Type	Value
all	STRING	ON
CMAKE_INSTALL_PREFIX	PATH	/home/petrstepanov/Applications/root-debug

CMake cache entries to remove (-U)

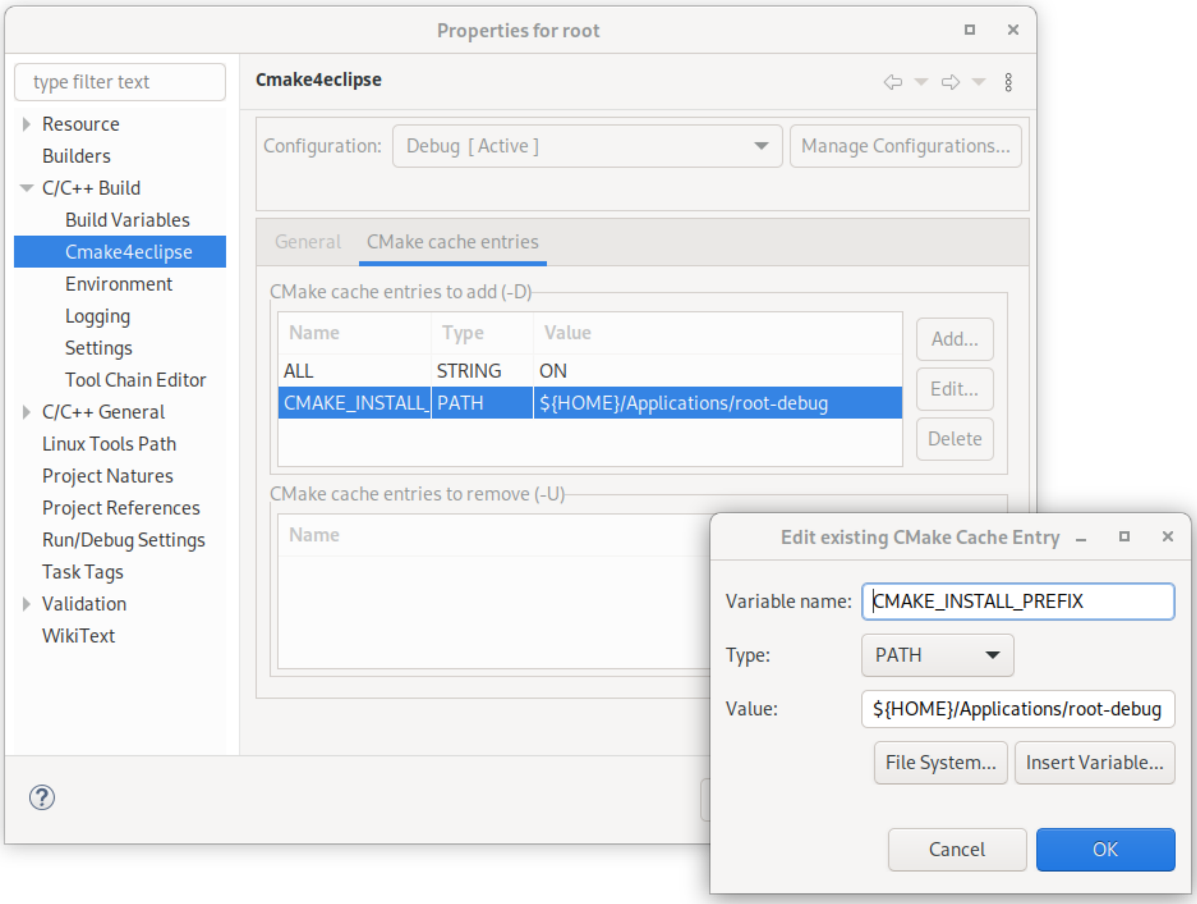
Name

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`	`STRING`	`ON`

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



Configuration: Debug [Active]

Name	Type	Value
ALL	STRING	ON
CMAKE_INSTALL_	PATH	${HOME}/Applications/root-debug

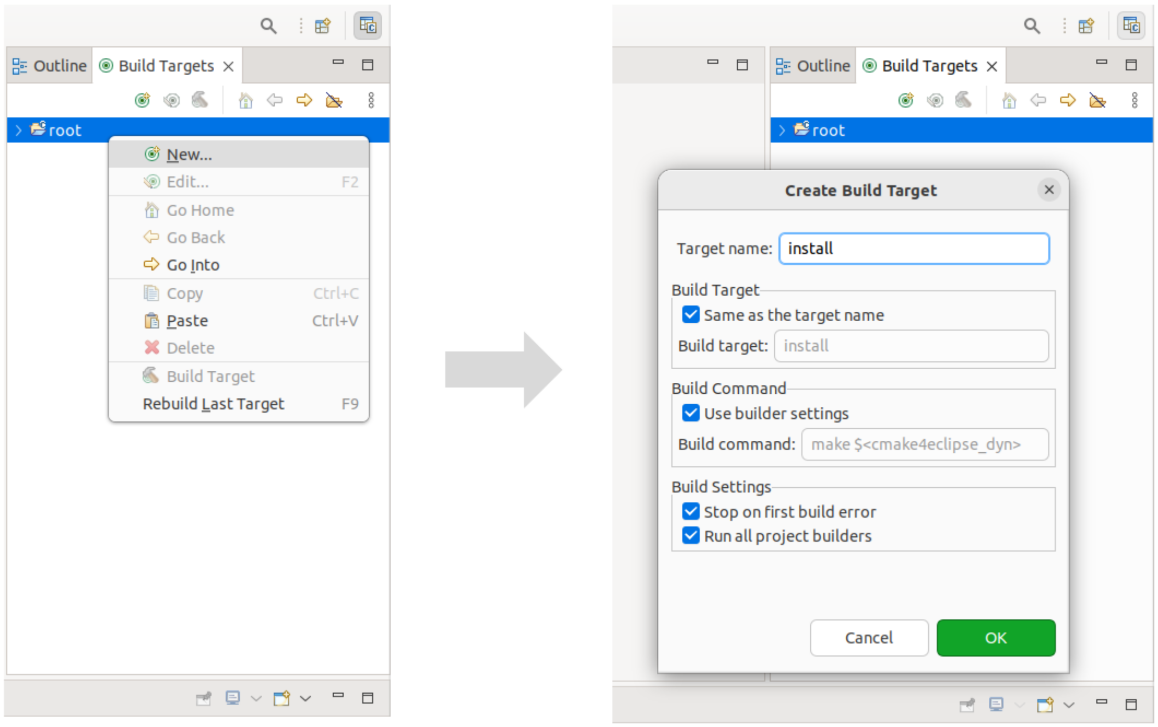
Variable name: CMAKE_INSTALL_PREFIX
Type: PATH
Value: ${HOME}/Applications/root-debug

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 official website and tailor your ROOT build for your needs.

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 (button in the bottom right corner) and 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.



Target name: install

Build Target
- [x] Same as the target name

Build target: install

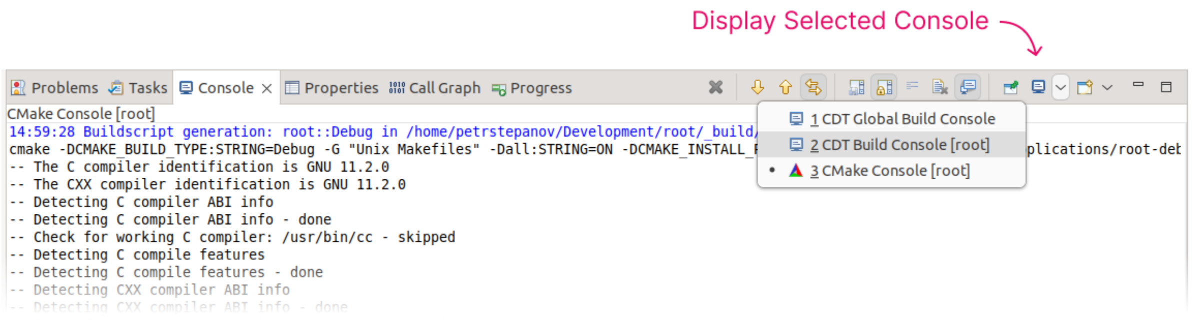
Build Command
- [x] Use builder settings

Build command: make $<cmake4eclipse_dyn>

Build Settings
- [x] Stop on first build error
- [x] Run all project builders

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.



```
CMake Console [root]
14:59:28 Buildscript generation: root::Debug in /home/petrstepanov/Development/root/_build/
cmake -DCMAKE_BUILD_TYPE:STRING=Debug -G "Unix Makefiles" -Dall:STRING=ON -DCMAKE_INSTALL_P                    plications/root-deb
-- The C compiler identification is GNU 11.2.0
-- The CXX compiler identification is GNU 11.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
```

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 the 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 few to parse thousands 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 would 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
`sudo apt install -y qt5* libX11* libXmu* libmotif* expat* libxerces-c*`	`sudo dnf -y install qt5* libX11* libXmu* motif* expat* mesa-libGL-devel`

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 (that has a better backward compatibility) there is a higher chance you will need to download a particular Geant4 version to satisfy a specific project you will be working on.

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`
`GEANT4_BUILD_MULTITHREADED`	`STRING`	`ON`

GEANT4_INSTALL_DATA	STRING	ON
GEANT4_USE_GDML	STRING	ON
GEANT4_USE_OPENGL_X11	STRING	ON
GEANT4_USE_QT	STRING	ON
GEANT4_USE_XM	STRING	ON

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 the Environment Variables for Eclipse

In the above 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:

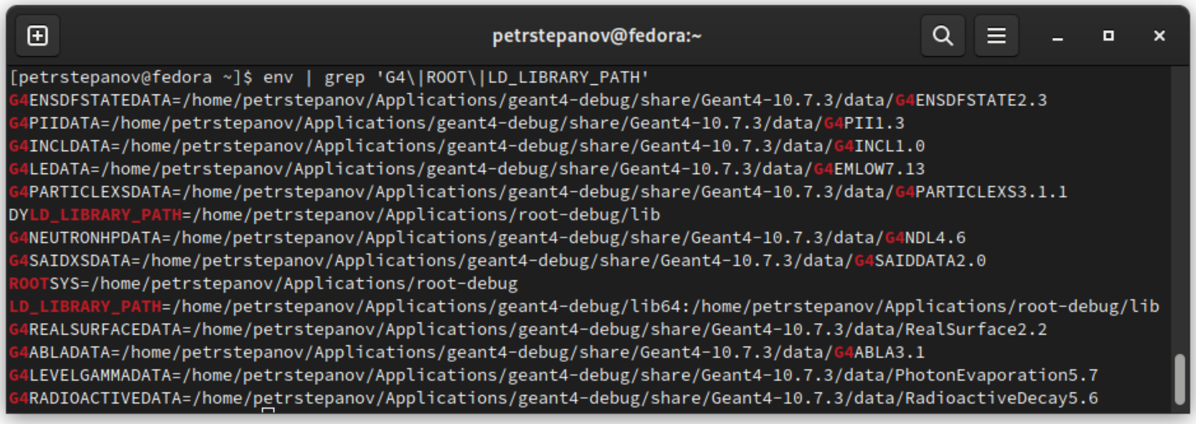
```
source $HOME/Applications/bin/thisroot.sh
source $HOME/Applications/bin/geant4.sh
```

However, we will use a different approach. Frameworks built with debug symbols (`-DCMAKE_BUILD_TYPE=Debug`) are executed notably slower compared to the release (`-DCMAKE_BUILD_TYPE=Release`) versions. Some users may want to have release versions of these frameworks installed on a local computer along with the debug build.

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'
```



```
[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 under the ~/.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 %u"
```

Please modify the path above to match your specific locations of ROOT, Geant4 and Eclipse IDE applications accordingly.

Now Eclipse will start in a BASH shell with correct development environment variables for your ROOT and Geant4 binaries compiled with debug symbols. The beauty of this method is that these environment variables are applied to only one particular shell process and do not affect your system-wide environment.

Before re-launching Eclipse please update your Linux launcher database with following command in Terminal:

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

Alternatively, users may simply logout and login back. Now upon a restart, Eclipse will be able to access ROOT and Geant4 development environment variables.

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 >”.

   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. 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. Therefore, “Add…” following build variable names, types and values:

Name	Type	Value

ROOT_DIR Geant4_DIR	PATH PATH	${HOME}/Applications/root-debug/cmake ${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. Again, it's a good idea to use Eclipse's folder picker ("File System…" button) to make sure the path variables are correct.

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 configuration**. Select your project in the project tree. In 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 (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 1. Convert a ROOT script into Standalone ROOT-based Program

You don't have a CMake based project and want to use my Template GitHub project. Then your ROOT script needs to be properly integrated into the existing project. Generally speaking, this involves following:

- Specify extra libraries you are utilizing in project's CMakeLists.txt.
- Explicitly define all the headers used in your script (Cling interpreter does not require that).
- Indicate certain class names in project's LinkDef.h file for the proper 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.

Appendix 2. Standalone Geant4-based program 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, the CMake file needs to locate both frameworks.

An example project of a Geant4 and ROOT-based program with nested CMake structure is outlined in this repository on GitHub.