OpenDSS-X Build Guide

C++ Build Guide

OpenDSS-X Version 1.1

Prepared by

Davis Montenegro

Electric Power Research Institute, Inc.

1325 G St. NW #530

Washington, DC 20005

Last update 08-23-2022

Table of Contents

List of Figures

List of Tables

EPRI | ELECTRIC POWER RESEARCH INSTITUTE

1 Purpose

This guide documents the process for building the OpenDSS – C++ version (henceforth referred to as OpenDSS-X) for the Microsoft Windows 10 and Red Hat Enterprise Linux 8 (RHEL 8) Operating System (OS). Build instructions are provided for the Integrated Development Environments (IDEs) listed in Table 1 and using Clang/CMake in a command line environment (Table 2). Throughout this guide the IDEs will typically be referred to by their acronyms as listed in Table 1. Software developers are welcome to contribute instructions to this guide for building OpenDSS-X for other IDEs and OS.

Table 1. IDEs supported by OpenDSS-X

IDE	OS	Acronym
Visual Studio Community 2019	Windows 10	VS Code 2019
Visual Studio Community 2022	Windows 10	VS Code 2022
Visual Studio Professional 2019	Windows 10	VS Pro 2019
Visual Studio Professional 2022	Windows 10	VS Pro 2022

Table 2. OpenDSS-X Compiler Requirements

Item	Description
C++	GCC 4.8.5 (Red Hat 4.8.5-4) Microsoft Visual Studio Community 2022 (64-bit) - Version 17.1.3 (Windows 10)
Build Engine	Clang 12.0.1 C++ compiler (Red Hat 12.0.1-4) CMake 3.20.2 for build process

The Visual Studio IDEs were selected because of their compatibility with CMake. Information on CMake can be found on the distribution site for the software application (CMake, 2022). The OpenDSS-X project files do not include a Visual Studio project because Visual Studio operates as an IDE around CMake.

2 Prerequisites

OpenDSS-X has been implemented and tested on Windows 10 and RHEL 8 operating systems. The hardware requirements of computer platforms for which OpenDSS-X has been tested to date are listed in Table 3. These provide the current baseline requirements for OpenDSS-X.

- Computer platform that meets specifications in Table 3.
- COTS products in Table 1.
- General knowledge of Visual Code.

Table 3. Minimum Computer Platform Requirements

Item	Description
Processor	RHEL: Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz Windows: i7 8-core @ 1.90 GHz
Random Access Memory	RHEL: 32 GB Windows: 16 GB
Storage Space	286 MB
Operating System	Red Hat Enterprise Linux (RHEL) version 8.5 (Ootpa) Microsoft Windows 10 Pro

3 Software Repository

This section describes the directory tree structure used within the git code repository and the contents of these directories to help developers identify the relevant portions of the code they wish to modify.

3.1 OpenDSS-X

OpenDSS-X[1] is an electric power distribution system simulator (DSS) designed to support distributed energy resource (DER) grid integration and grid modernization. It enables engineers to perform complex analyses using a flexible, customization, and easy to use platform intended specifically to meet current and future distribution system challenges and provides a foundation for understanding and integrating new technologies and resources. This power system analysis software tool was translated from the Delphi computer language (Electric Power Research Institute, 2022) into C++ to make it accessible to a larger community of software developers and used as a core library upon which to build new capabilities. More information on OpenDSS-X can be found in the Reference Guide (Dugan & Montenegro, 2020). OpenDSS (the Delphi version) that runs on a Windows operating system is available (Electric Power Research Institute, 2022) along with a PowerPoint tutorial (Fu, 2019).

The OpenDSS-X software repository has a file distribution as shown in Figure 1 with the directory contents as described in Table 4. The contents of the klusolve folder are described in the next section.

[1] OpenDSS is an EPRI software tool (open source) that utilizes network solution algorithms that have been proven and extensively used by EPRI, academic researchers and the electricity utility industry.

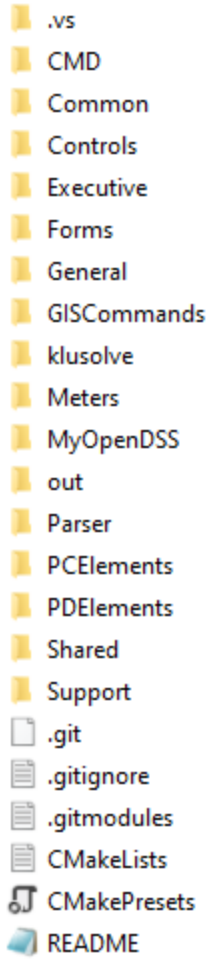


Figure 1. Snapshot of the OpenDSS-X copy obtained from the repository

Table 4. OpenDSS-X directory structure

Folder name	*Description*
CMD	Contains the main project files. The project was designed for compatibility with CMake, facilitating the cross-platform adoption without losing some of the C++ semantics introduced by other vendors. In this folder the user will find the main project file (OpenDSSX.cpp), the CMake configuration files and the external libraries such as KLUSolve.dll and its Linux equivalents.
Common	Contains the routines and objects commonly used across the program for simulation purposes. Utilities, Solution and Circuit are some examples of the units that can be found in this folder.
Controls	Contains the objects implemented as controls in the simulation. Controls are objects that can modify the behavior of a set of elements (it can be 1) using the feedback from a monitored part of the circuit model. Regulators, capacitor controls, smart inverters are some examples of the control that can be found in this folder.

Executive	Contains all the routines and objects that interpret commands given using the OpenDSS-X scripting language, redirecting the program internally to perform the actions commanded by the user using this language.
Forms	Contains the routines for returning messages to the user. The messages can be confirmation, warnings, or errors. This is the equivalent for command line of the VCL forms implemented in the original OpenDSS-X written in Delphi.
General	Contains the objects representing libraries that can be used to simplify the elements' description within a script. LineCodes, LineGeometries, TransformerCodes and XYCurves are examples of these objects.
GISCommands	Contains the commands for interacting with OpenDSS-GIS. Not available in OpenDSS-X but is in the repository to mimic the file distribution in the original version (Delphi).
klusolve	A complex sparse matrix library (see Section 3.2).
Meters	Contains the code for the objects representing meters, monitors and sensor like elements that can be deployed across the circuit model for capturing simulation values.
MyOpenDSS	Contains a set of files that can be used as template for external user models.
Parallel_Lib	Implements the routines and objects needed for performing parallel processing related operations.
Parser	Contains the objects implementing parsers and RPN for interpreting numbers and mathematical expressions when processing commands in OpenDSS-X scripting language.
PCElements	Contains the code for the objects describing Power Conversion Elements (PCEs). These are normally connected in shunt across the model. Loads, capacitors, energy storage are some examples of PCEs.
PDElements	Contains the code for the objects describing Power Delivery Elements (PDEs). These can have 2 terminals and are normally connected in series across the model for energy transport. Transformers, lines, and series connected capacitors are some examples of PDEs.
Shared	Contains routines shared by multiple objects across the program. They are vital for the program operation.
Support	Contains the routines implemented for reproducing the operation of Delphi specialized commands in C++. These routines are fundamental for the program operation since many of the commands that can be fund across the code are not C++ native.

3.2 KLUSolve

KLUSolve is a complex sparse matrix library tailored to electric power systems and is licensed under the GNU Lesser General Public License (LGPL) version 2.1 (McDermott, 2013; Davis, 2006). KLUSolve is based on the KLU, CSparse, and supporting libraries developed by Timothy A. Davis and his students. The KLUSolve library has a top-level directory structure as shown in Figure 2 with the contents as described in Table 5.

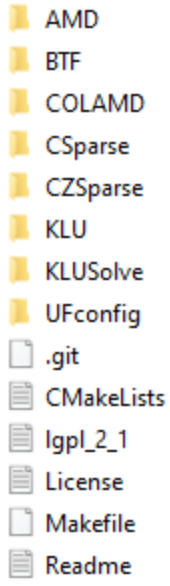


Figure 2. Snapshot of the KLUSolve library obtained from the repository

Table 5. KLUSolve directory structure

Folder name	Description
AMD	Contains a set of routines for permuting sparse matrices prior to factorization. Includes a version in C, a version in Fortran, and a MATLAB mexFunction.
BTF	BTF is a software package for permuting a matrix into block upper triangular form. It includes a maximum transversal algorithm, which finds a permutation of a square or rectangular matrix so that it has a zero-free diagonal (if one exists); otherwise, it finds a maximal matching which maximizes the number of nonzeros on the diagonal. The package also includes a method for finding the strongly connected components of a graph. These two methods together give the permutation to block upper triangular form.
COLAMD	The COLAMD column approximate minimum degree ordering algorithm computes a permutation vector P such that the LU factorization of A (:,P) tends to be sparser than that of A. The Cholesky factorization of (A (:,P))'*(A (:,P)) will also tend to be sparser than that of A'*A. SYMAMD is a symmetric minimum degree ordering method based on COLAMD, available as a MATLAB-callable function. It constructs a matrix M such that M'*M has the same pattern as A, and then uses COLAMD to compute a column ordering of M. Colamd and symamd tend to be faster and generate better orderings than their MATLAB counterparts, colmmd and symmmd.
CSparse	The algorithms contained in CSparse have been chosen with five goals in mind: (1) they must embody much of the theory behind sparse matrix algorithms, (2) they must be either asymptotically optimal in their run time and memory usage or be fast in practice, (3) they must be concise to be easily understood and short enough to print in the book,

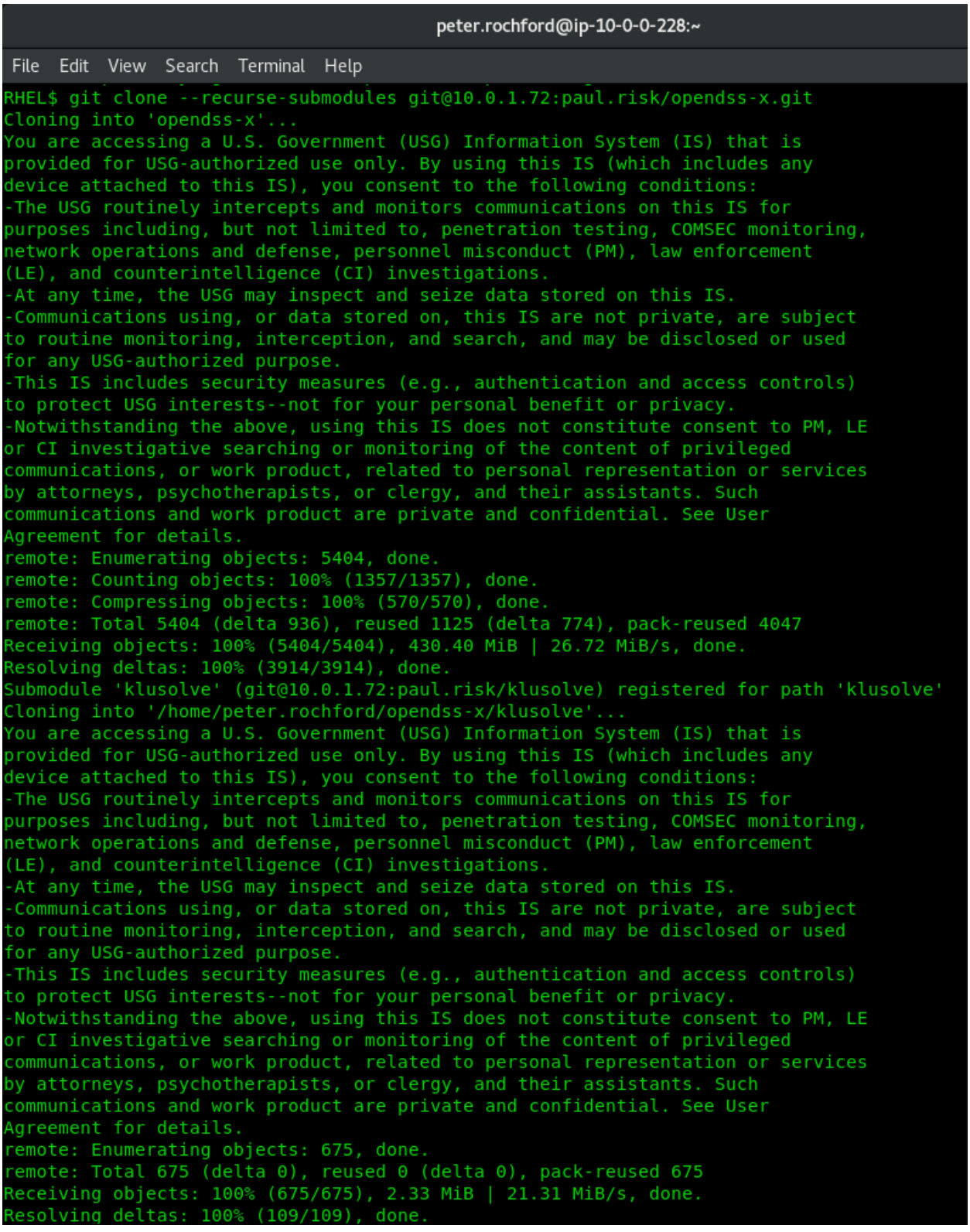
	(4) they must cover a wide spectrum of matrix operations, and (5) they must be accurate and robust. The focus is on direct methods; iterative methods and solvers for eigenvalue problems are beyond the scope of this package.
CZSparse	These are complex versions of selected functions from the CSparse package by Timothy A. Davis. The purpose is to compile on Windows without a C99 compiler. A->x stores alternate real/imaginary values.
KLU	Contains the C code that factorizes P*A into L*U, using the Gilbert-Peierls algorithm with optional symmetric pruning by Eisenstat and Liu.
KLUSolve	Contains the C++ code that defines the entry point for the DLL application.
UFconfig	Contains configuration settings for the software packages.

3.3 Get OpenDSS-X via Git

Skip this step if you have been provided the OpenDSS-X code base. Perform this step if you need to obtain the OpenDSS-X code base from a git repository. Open a terminal window in the Linux operating system and run the command:

$ git clone --recurse-submodules git@XX.XX.XX.XX:firstname.lastname/opendss-x.git

where XX.XX.XX.XX is the internet protocol (IP) address for the git server and firstname.lastname is the name of the owner of the git repository. Note that "$" is used above to indicate the Linux command prompt. You will get output such as that shown below. When the opendss-x repository is cloned it will automatically clone the klusolve repository using git submodules commands and architecture.


peter.rochford@ip-10-0-0-228:~
File Edit View Search Terminal Help
RHEL$ git clone --recurse-submodules git@10.0.1.72:paul.risk/opendss-x.git
Cloning into 'opendss-x'...
You are accessing a U.S. Government (USG) Information System (IS) that is
provided for USG-authorized use only. By using this IS (which includes any
device attached to this IS), you consent to the following conditions:
-The USG routinely intercepts and monitors communications on this IS for
purposes including, but not limited to, penetration testing, COMSEC monitoring,
network operations and defense, personnel misconduct (PM), law enforcement
(LE), and counterintelligence (CI) investigations.
-At any time, the USG may inspect and seize data stored on this IS.
-Communications using, or data stored on, this IS are not private, are subject
to routine monitoring, interception, and search, and may be disclosed or used
for any USG-authorized purpose.
-This IS includes security measures (e.g., authentication and access controls)
to protect USG interests--not for your personal benefit or privacy.
-Notwithstanding the above, using this IS does not constitute consent to PM, LE
or CI investigative searching or monitoring of the content of privileged
communications, or work product, related to personal representation or services
by attorneys, psychotherapists, or clergy, and their assistants. Such
communications and work product are private and confidential. See User
Agreement for details.
remote: Enumerating objects: 5404, done.
remote: Counting objects: 100% (1357/1357), done.
remote: Compressing objects: 100% (570/570), done.
remote: Total 5404 (delta 936), reused 1125 (delta 774), pack-reused 4047
Receiving objects: 100% (5404/5404), 430.40 MiB | 26.72 MiB/s, done.
Resolving deltas: 100% (3914/3914), done.
Submodule 'klusolve' (git@10.0.1.72:paul.risk/klusolve) registered for path 'klusolve'
Cloning into '/home/peter.rochford/opendss-x/klusolve'...
You are accessing a U.S. Government (USG) Information System (IS) that is
provided for USG-authorized use only. By using this IS (which includes any
device attached to this IS), you consent to the following conditions:
-The USG routinely intercepts and monitors communications on this IS for
purposes including, but not limited to, penetration testing, COMSEC monitoring,
network operations and defense, personnel misconduct (PM), law enforcement
(LE), and counterintelligence (CI) investigations.
-At any time, the USG may inspect and seize data stored on this IS.
-Communications using, or data stored on, this IS are not private, are subject
to routine monitoring, interception, and search, and may be disclosed or used
for any USG-authorized purpose.
-This IS includes security measures (e.g., authentication and access controls)
to protect USG interests--not for your personal benefit or privacy.
-Notwithstanding the above, using this IS does not constitute consent to PM, LE
or CI investigative searching or monitoring of the content of privileged
communications, or work product, related to personal representation or services
by attorneys, psychotherapists, or clergy, and their assistants. Such
communications and work product are private and confidential. See User
Agreement for details.
remote: Enumerating objects: 675, done.
remote: Total 675 (delta 0), reused 0 (delta 0), pack-reused 675
Receiving objects: 100% (675/675), 2.33 MiB | 21.31 MiB/s, done.
Resolving deltas: 100% (109/109), done.

4 Visual Studio Professional (VS Pro)

4.1 Opening the project

After launching Visual Studio Professional, select the option “Open a local folder” from the options provided as shown in Figure 3.

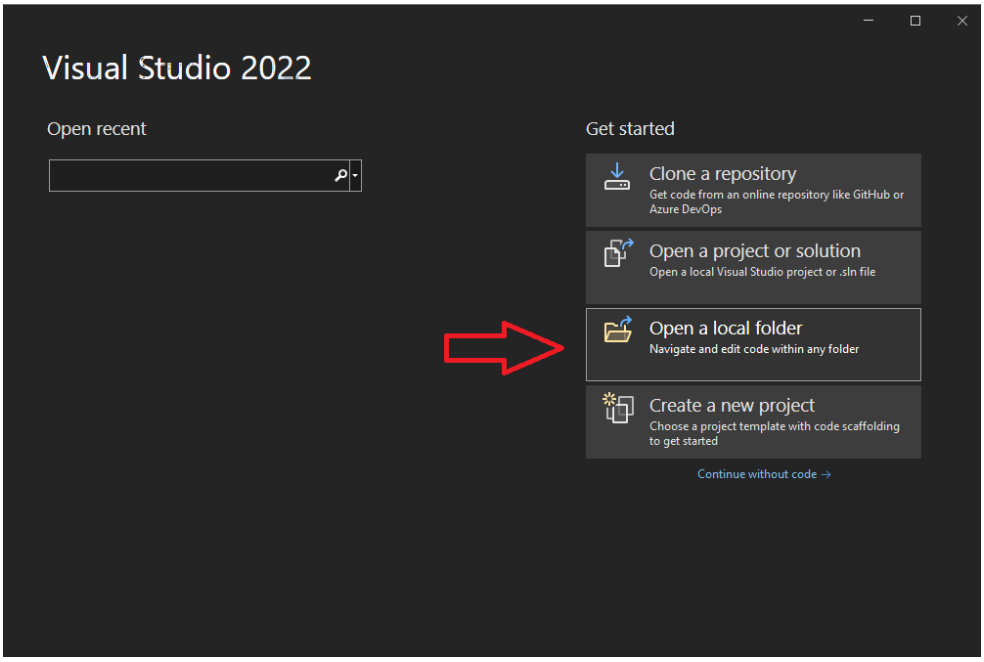


Figure 3. Starting Visual Studio Professional

A new browsing window will open for the user to select the project’s folder. Please redirect that window to the folder CMD within the OpenDSS-X clone created in Section 3. Once you are within that folder select the option “Select folder” as shown in Figure 4.

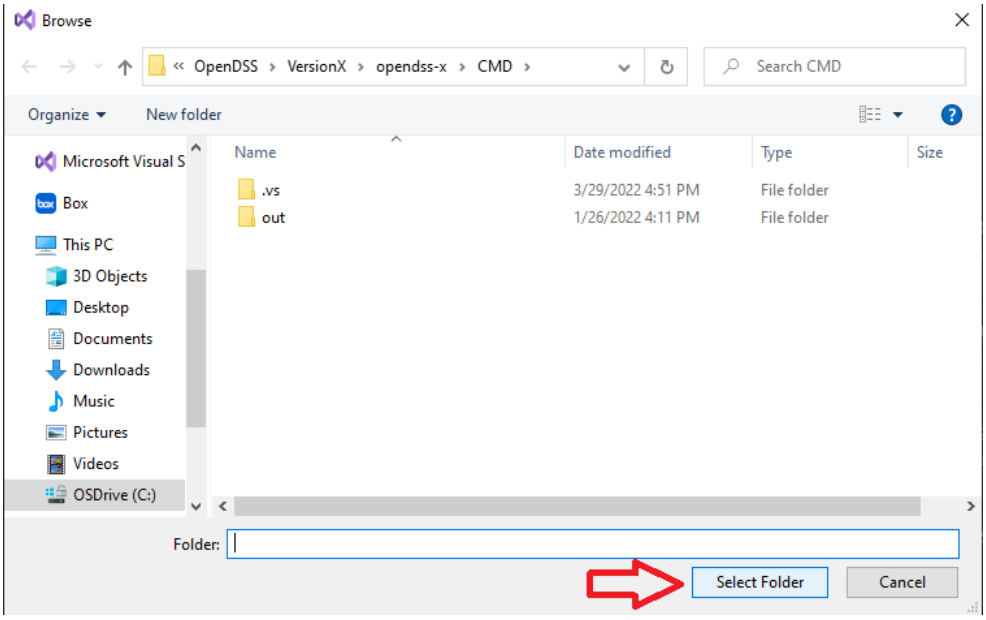


Figure 4. Find the project's source folder

With that, Visual Studio will setup the environment using the CMake files within that folder. The environment should look as shown in Figure 5.

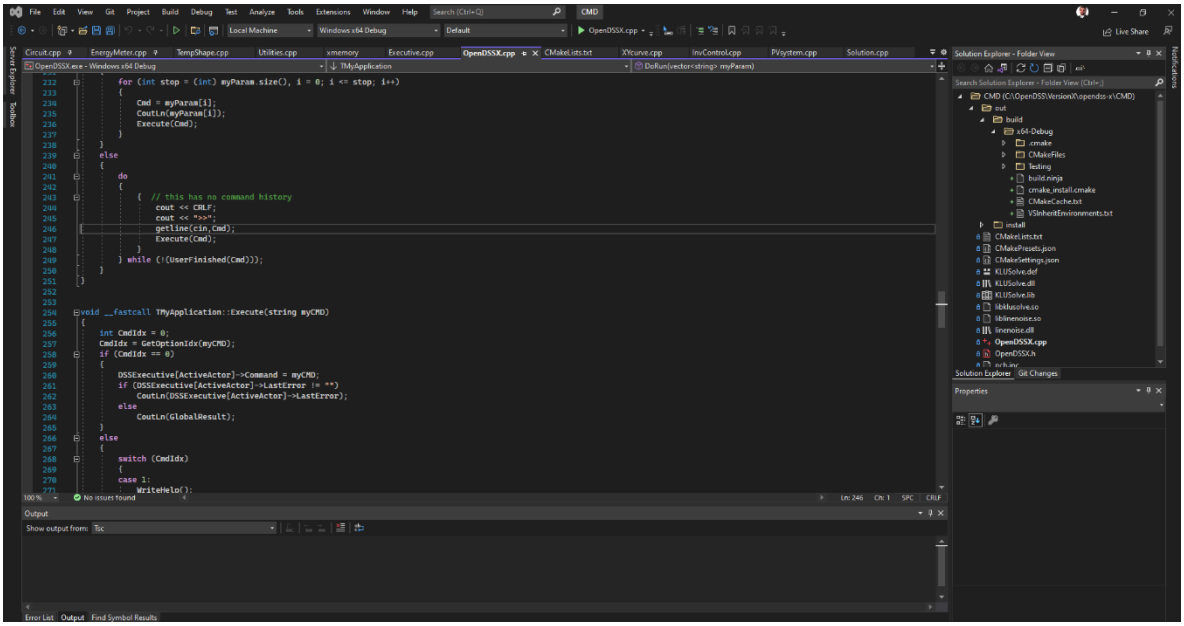


Figure 5. Environment ready

At the right of the screen (normally), the user will find the project's tree, which will display only the files contained within the CMD folder. The first step will be to define the startup item. To do so, at the project's tree right click over the file called OpenDSSX.cpp and select "Set as startup item" from the pop-

up menu displayed as shown in Figure 6. That action will tell Visual Studio what's the starting point in the project when compiling using CMake.

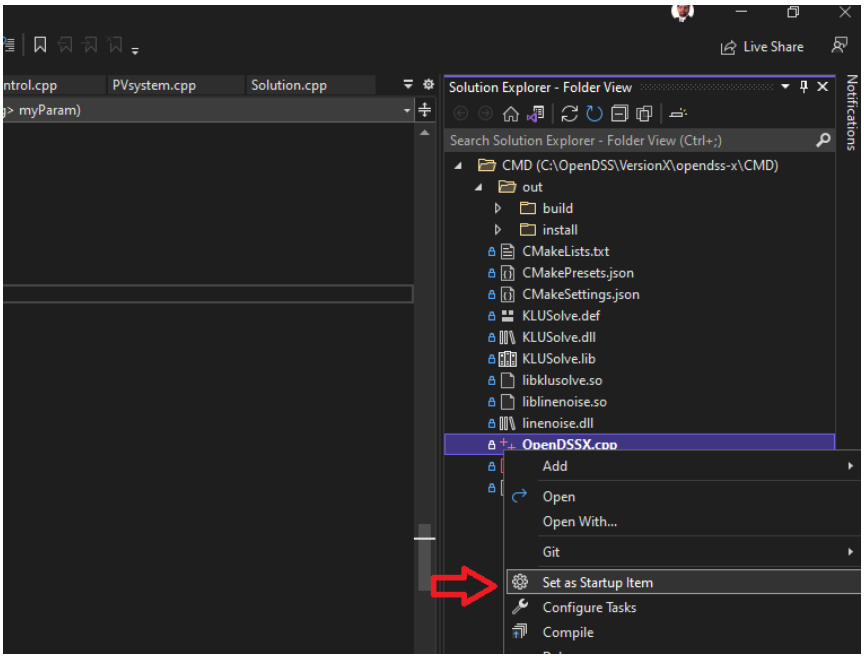


Figure 6. Setting the startup item

Next, compile the project. To do so go to the menu item Build → Build All as shown in Figure 7. That should build the project and throw no errors. If there is any error, it should be notified at the output window at the bottom of the IDE.

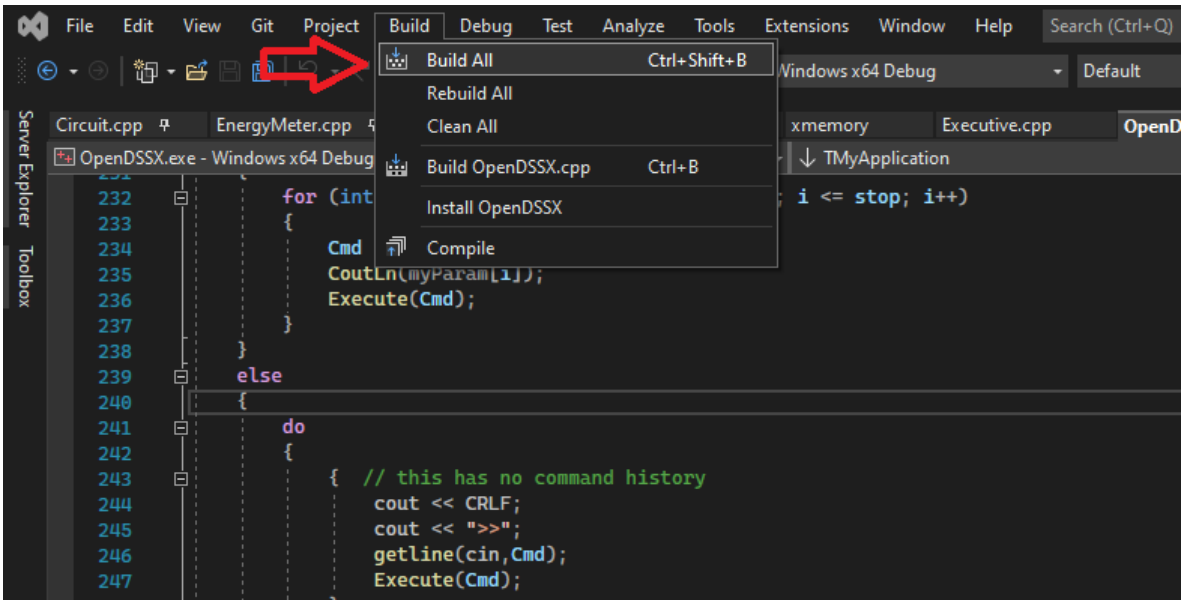


Figure 7. Building the Project using the Visual Studio IDE

The program comes in Debug mode by default, to change this setting double click on the file called CMakeSettings.json at the project's tree. This will open the configuration menu as shown in Figure 8. There, the user can decide to use Debug or Release compilations.

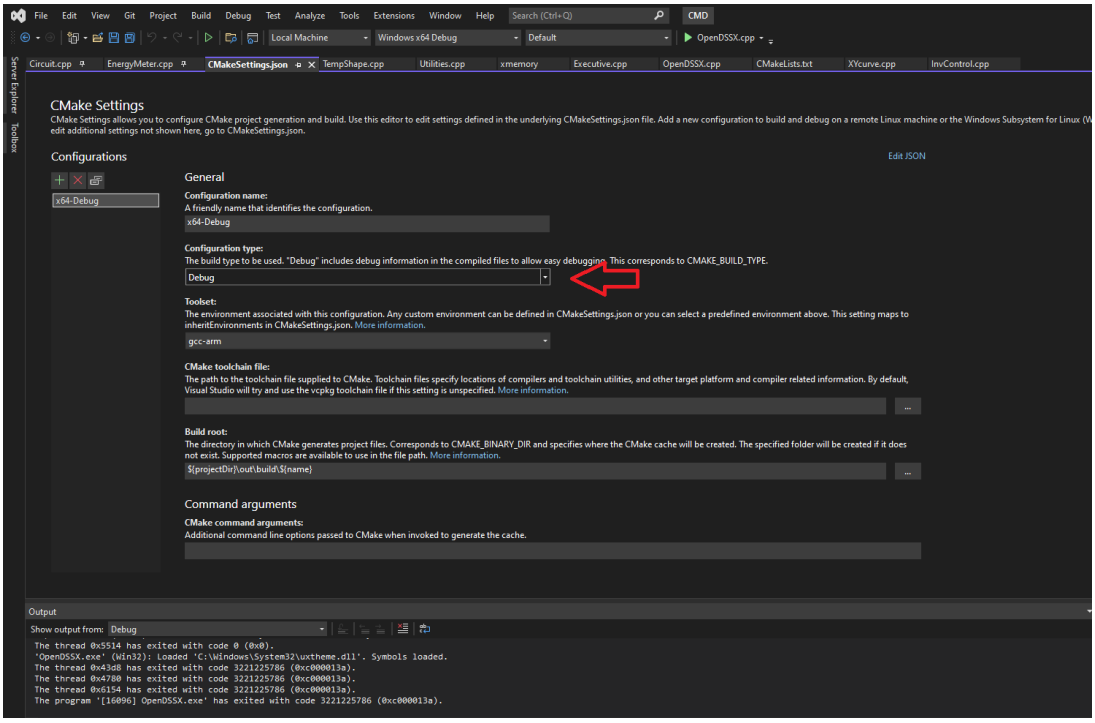


Figure 8. Configuring the release type for the project.

To use the default (Debug), and start the program, press the key F5 or go to the menu Debug → Start debugging as shown in Figure 9. The result will be the compilation of files that need to be compiled and the start of the program as shown in Figure 10.

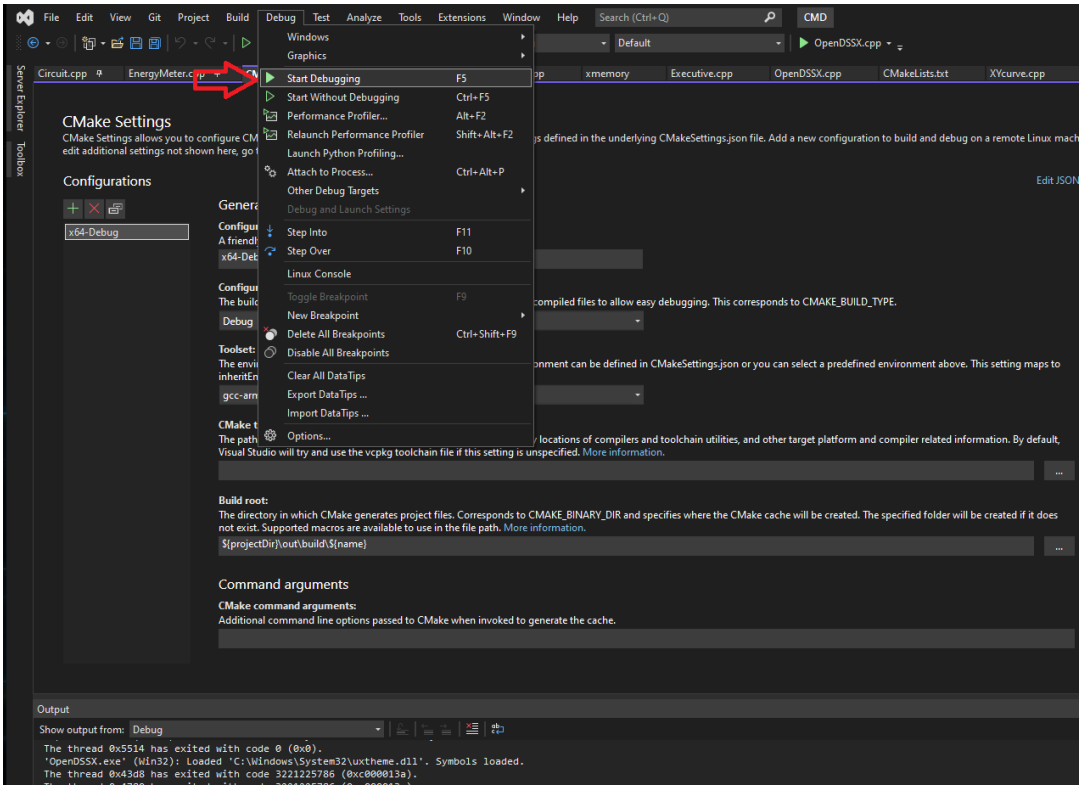


Figure 9. Starting debugging

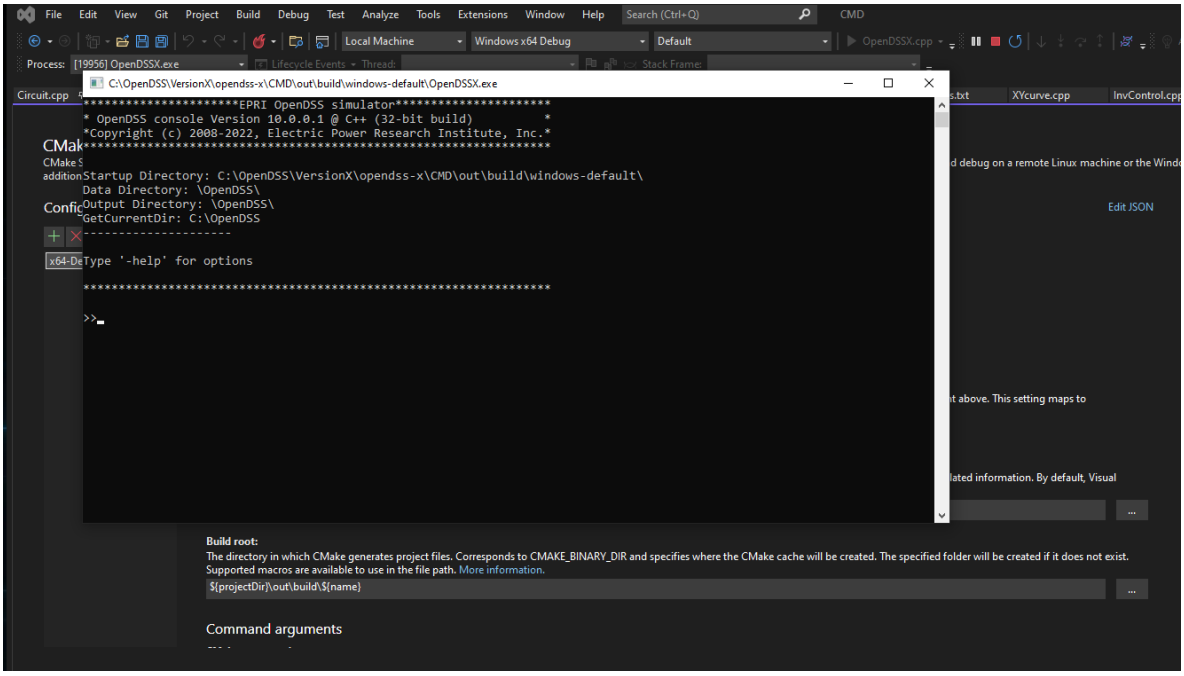


Figure 10. OpenDSS-X running

4.2 Troubleshooting

4.2.1 Cannot open file KLUSolve.lib

In the first attempt of running the program you may get an error message telling you that a link file is missing as shown in Figure 11. This is because the output directory where the executable generated by CMake is located cannot find some dependencies. To solve it, go to the folder CMD within the project's directory and copy the files KLUSolve.dll and KLUSolve.lib into the folder …CMD\out\build\windows-default.

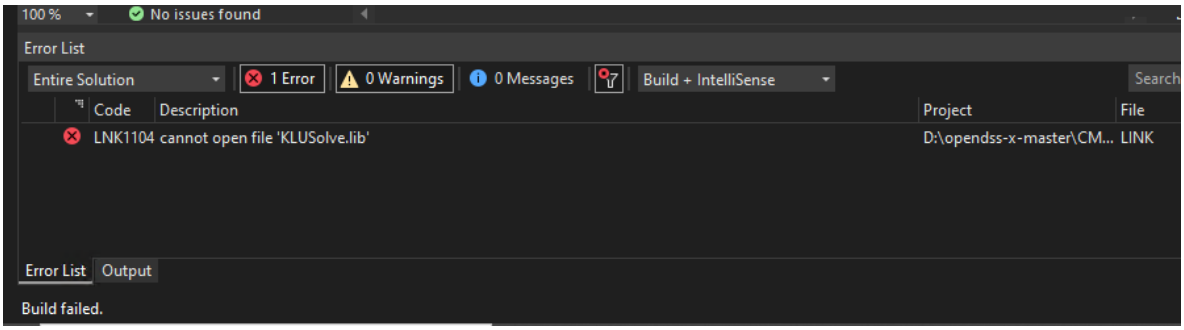


Figure 11. Possible error in the first run attempt.

Then, try to run the program again, the error message should disappear, and you should be seeing something like the image displayed in Figure 10.

5 Visual Studio Community (VS Code)

5.1 Linux

This section describes how to set up VS Code for Linux. This OS is less commonly used than Windows, and therefore requires the user be experienced in using Linux within a terminal environment running a bash shell.

5.1.1 Setting Up Project

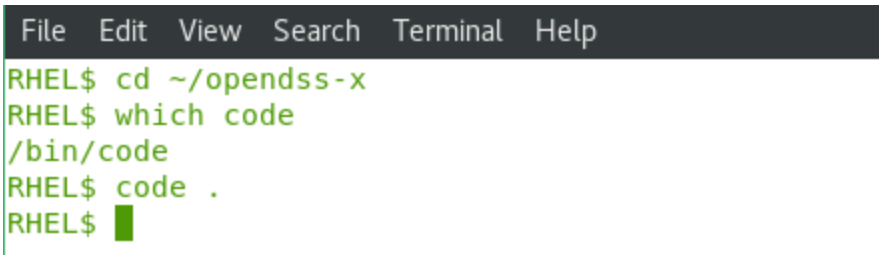
Start by creating a copy of a VS Code project inside the same directory where the code is stored, i.e., the location to which the OpenDSS-X git repository was cloned. Open a terminal window in the Linux operating system and navigate to the using the command:

```
$ cd ~/opendss-x
```

where ~/opendss-x in the above is the directory location for this example. Then enter the command

```
$ code .
```

which should automatically launch VS Code in that location as shown below. This will create an auxiliary .vscode folder that will host many of the json files used by VS Code to control various functions within the IDE, such as debugging and build an executable.

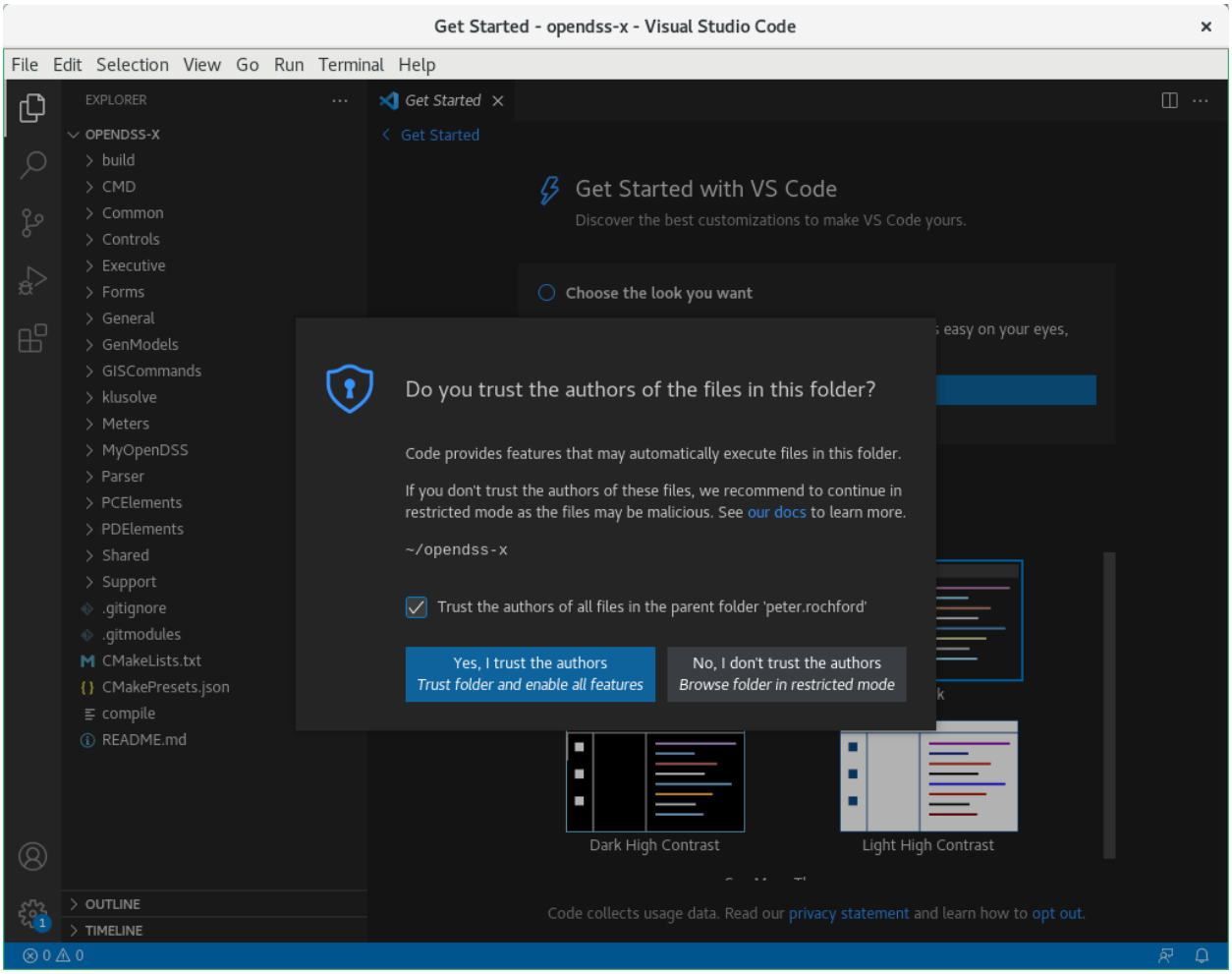


Note: if you have opened VSCODE before, there may already be a .vscode folder under you home directory, which prevents VSCODE from creating one in your project folder. If this is the case, it can be resolved by closing VSCODE, deleting the .vscode directory out of the home directory, and typing

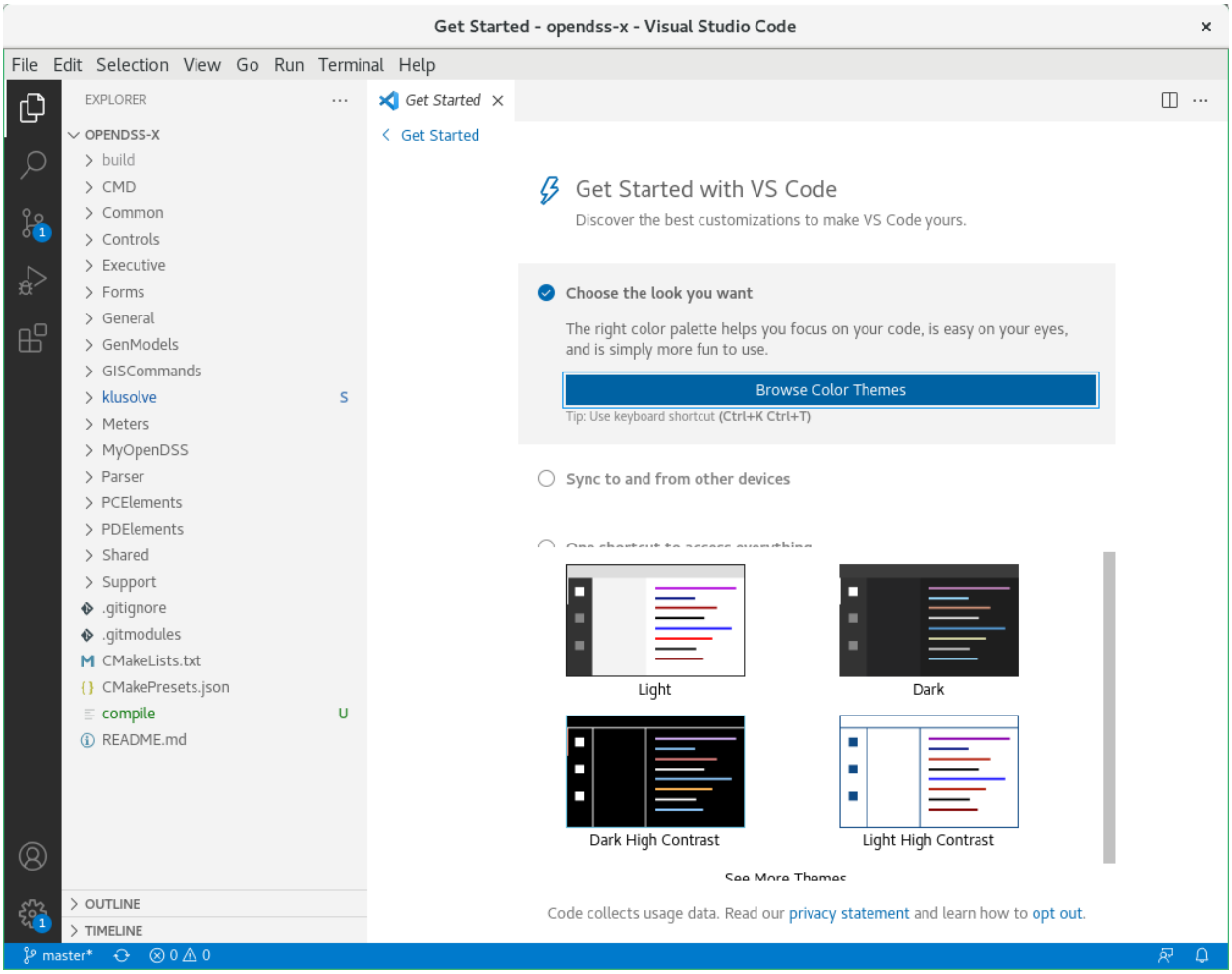
```
$ code .
```

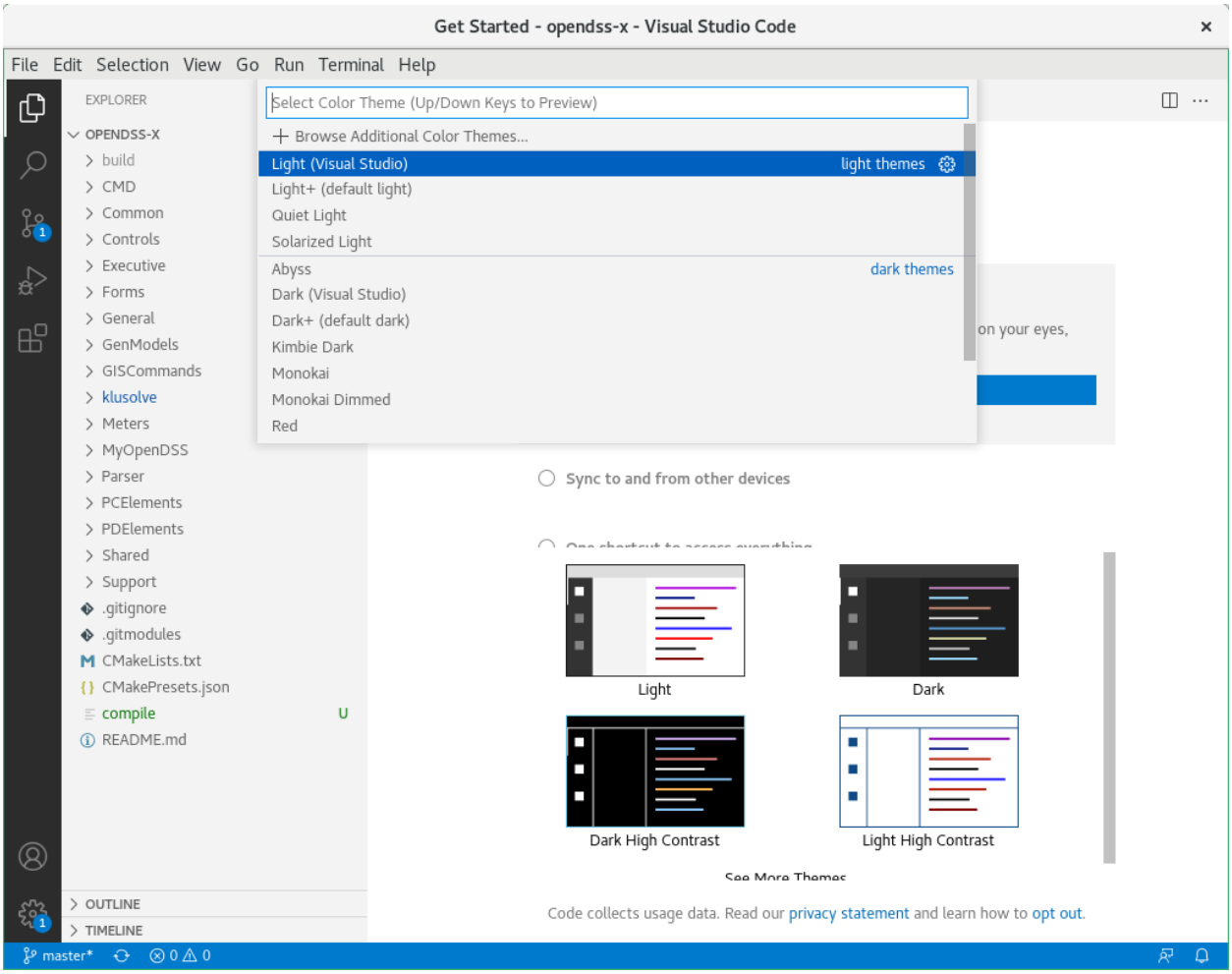
If you're trying to preserve existing settings or other projects, temporarily renaming it instead of deleting it may also work.

VS Code will launch, and the user may be prompted on first use about trusting the files in the folder. Check the "Trust the authors" box and then click on the "Yes, I trust the authors…" box.

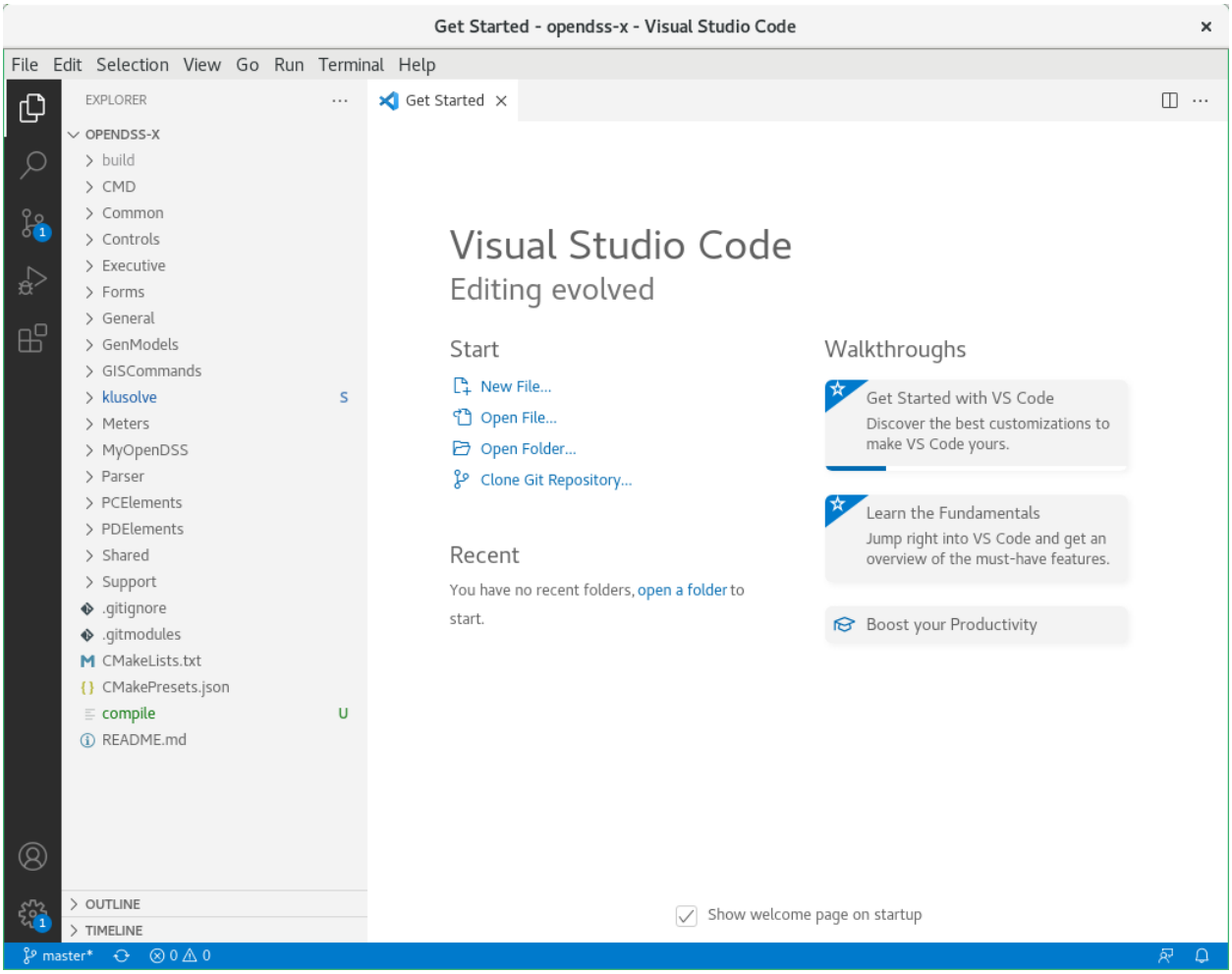


After the trust prompt window disappears, you can then select to choose the look you want for VS Code by clicking on the Browse Color Themes button. In this section, we've elected to use Light (Visual Studio).

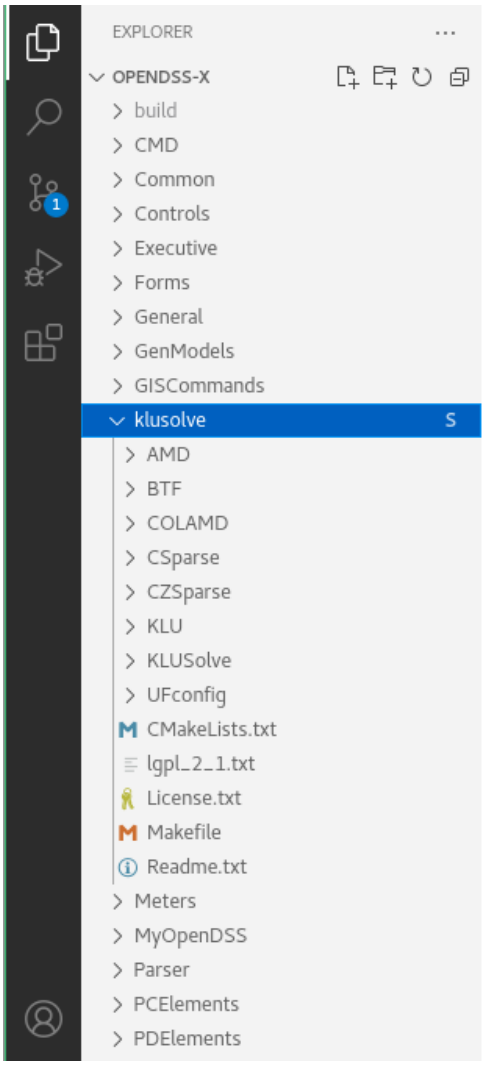

Get Started - opendss-x - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPENDSS-X
build
CMD
Common
Controls
Executive
Forms
General
GenModels
GISCommands
klusolve
Meters
MyOpenDSS
Parser
PCElements
PDElements
Shared
Support
.gitignore
.gitmodules
CMakeLists.txt
CMakePresets.json
compile
README.md
OUTLINE
TIMELINE
Get Started
Get Started
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Choose the look you want
The right color palette helps you focus on your code, is easy on your eyes, and is simply more fun to use.
Browse Color Themes
Tip: Use keyboard shortcut (Ctrl+K Ctrl+T)
Sync to and from other devices
Light
Dark
Dark High Contrast
Light High Contrast
See More Themes
Code collects usage data. Read our privacy statement and learn how to opt out.
master*



Once you've selected the color theme, click on the "< Get Started" in the top left to get the VS Code window below. You should see the OPENDSS-X folder organization structure on the right side as shown below.



The user can inspect the folder structure to ensure that all files have been copied correctly. As described in Section 3, the opendss-x project will clone the klusolve project via git submodule commands. The user doesn't have to individually clone the individual klusolve repository. After a quick inspection, the user should be able to see the two repos in the solution explorer as shown in the next figure. Note how opendss-x is the top repository, with the klusolve repository a subfolder of the opendss-x folder.



5.1.2 Installing Plugins

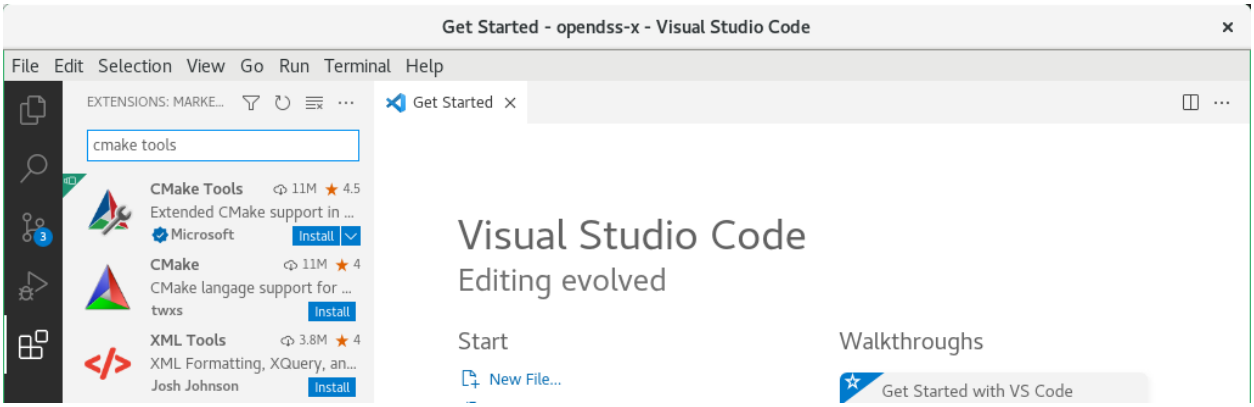
A few plugins are required to successfully build OpenDSS-X within VS Code as listed in Table 6. They are identified according to whether they are required or desirable. This section can be skipped if they are already installed in your VS Code implementation.

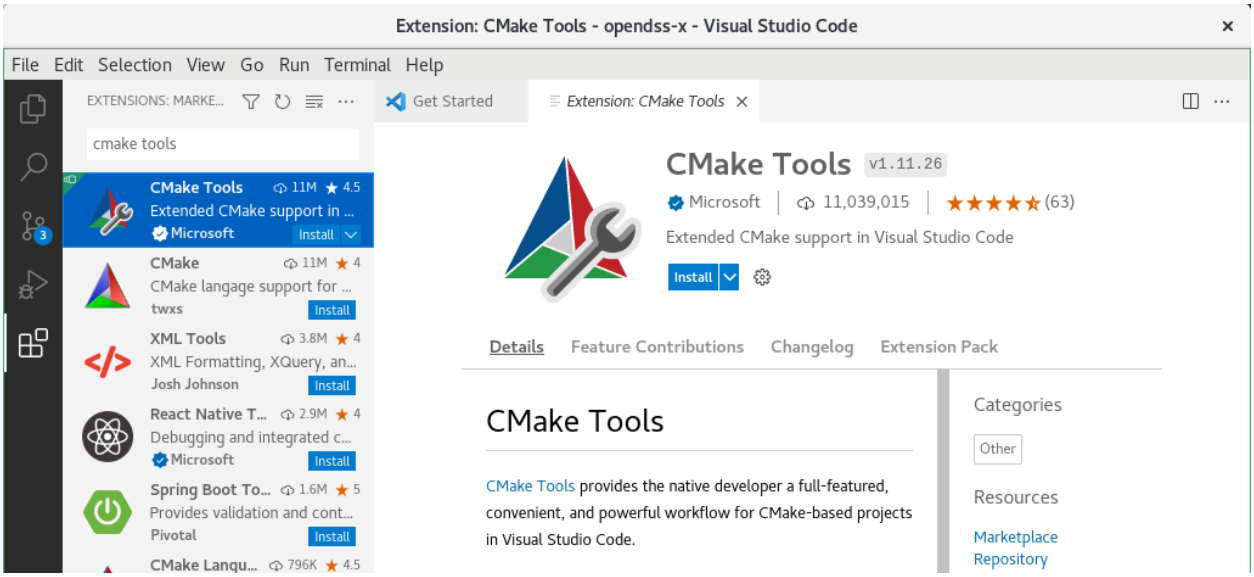
Table 6. VS Code Extensions

Extension	Description	Priority
CMake	Language support for Visual Studio Code	Required
CMake Tools	Extended CMake support in Visual Studio Code	Required
C/C++ Extension Pack	Popular extensions for C++ development in Visual Studio Code.	Desired

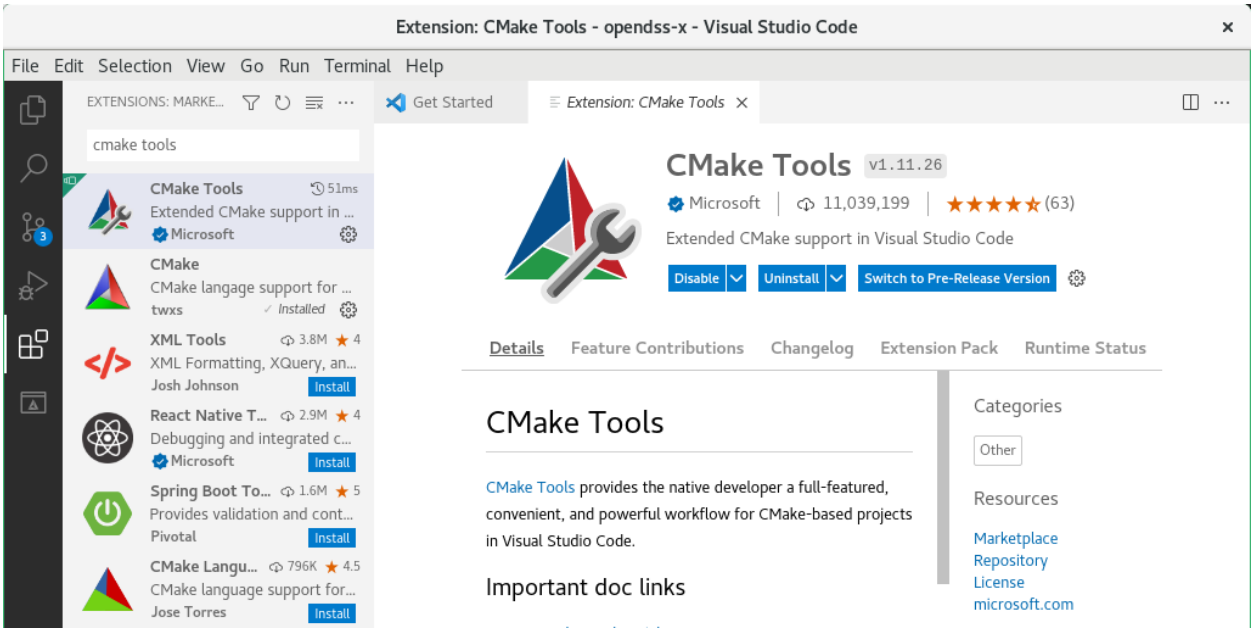
To install these extensions, begin by opening the Extensions view (Ctrl+Shift+X, or View → Extensions). Then do the following steps to install CMake Tools that will also include CMake as a dependency.

1. Enter CMake Tools in the search box and press enter. The extension should appear at the top of the list as shown below.
2. Select the CMake Tools extension to have its details appear in a new tab window *Extension: CMake Tools*.
3. Click the blue Install button to install this extension. 





CMake Tools will install along with CMake.

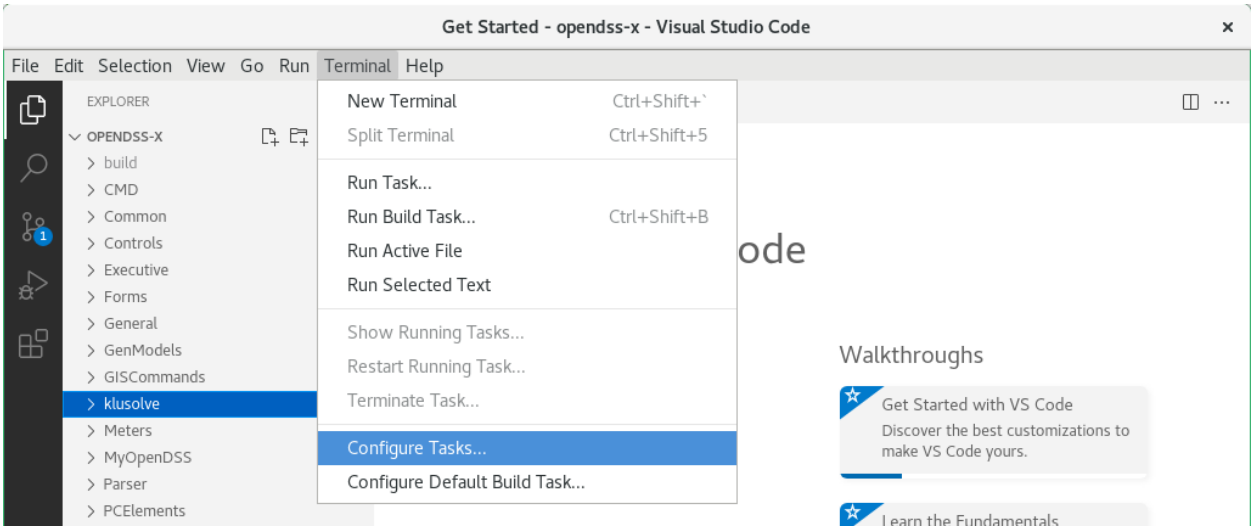


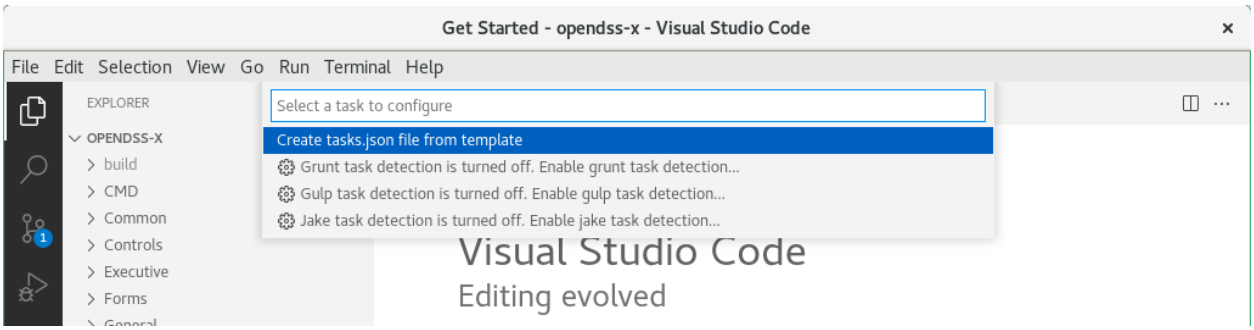
Close the window tab for *Extension CMake Tools* when finished.

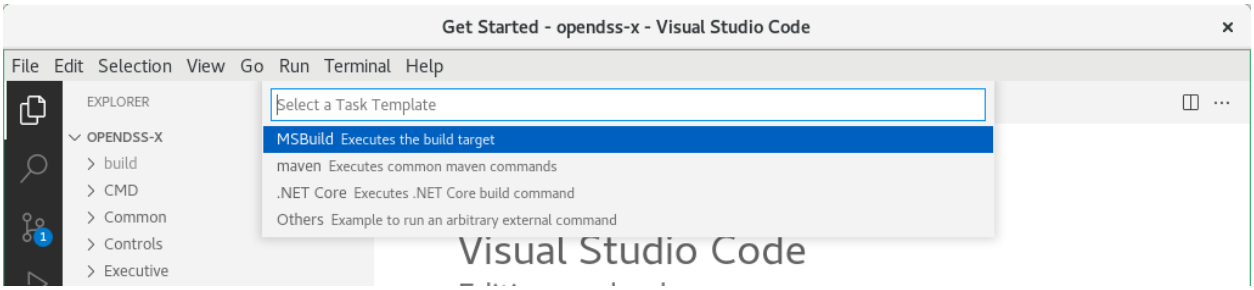
5.1.3 Build Setup

To build the application, the user must configure the tasks.json file as shown below. This is easily be done by following the sequence of steps:

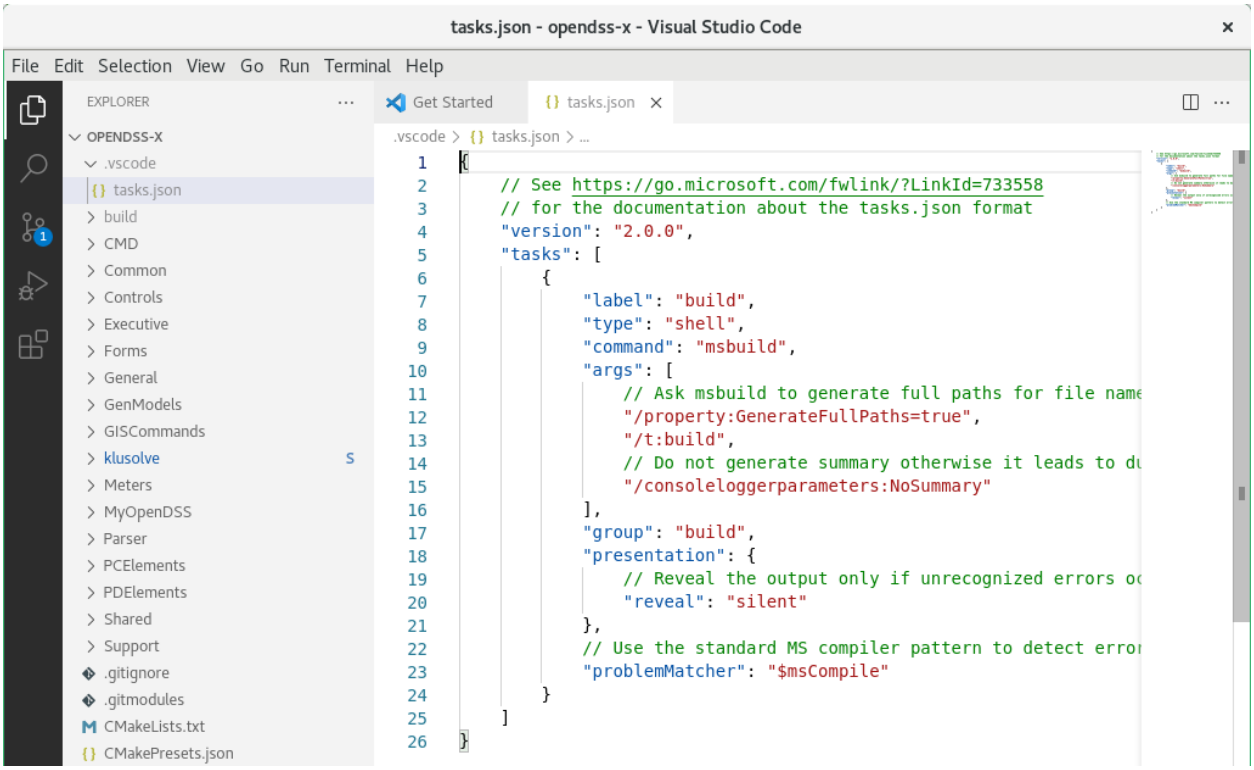
1. Use the menu command Terminal → Configure Tasks….
2. For the “Select a task to configure”, select “Create tasks json.file from template”.
3. For the “Select a Task Template”, select “MSBuild Executes the build target”.







A new editor tab window will appear with a template for the tasks.json file.

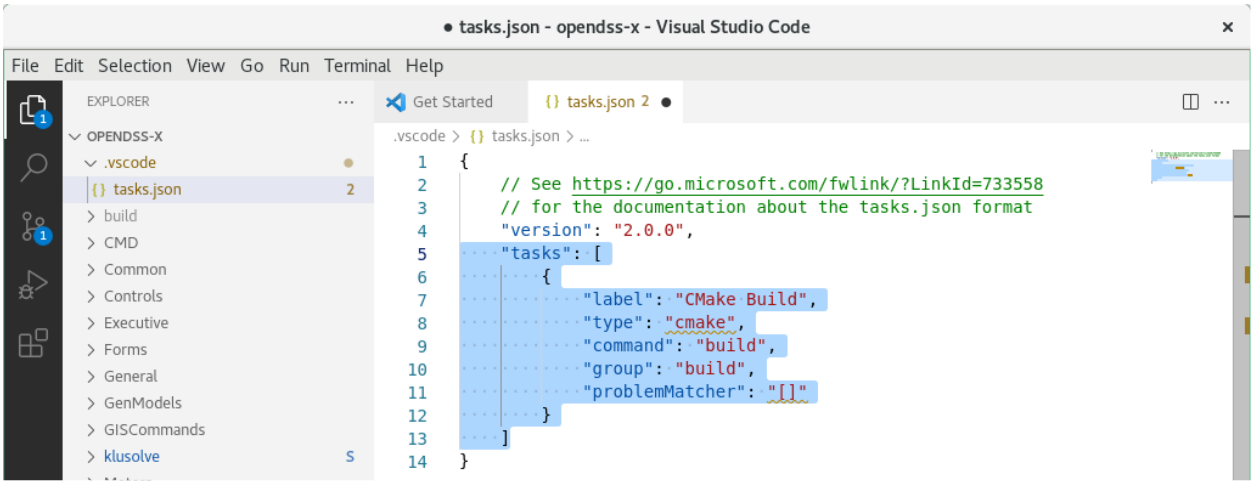


Edit the content of the tasks.json file as given below.

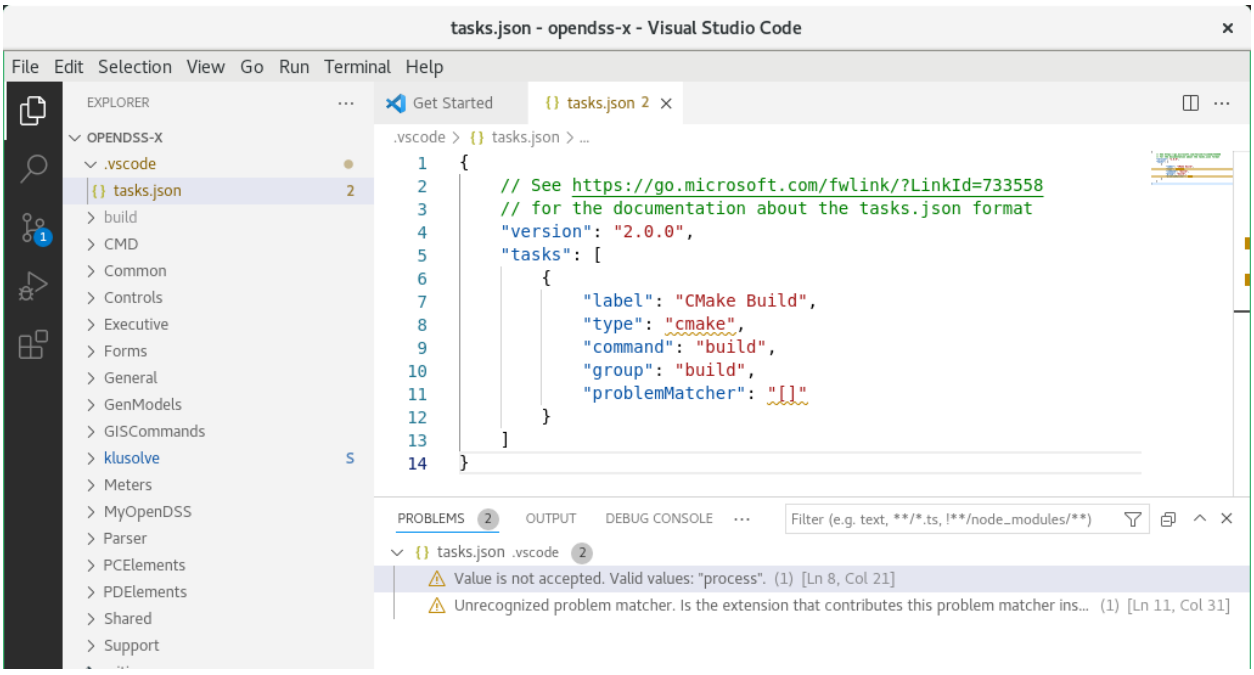
"tasks": [
{

```
        "label": "CMake Build",
        "type": "cmake",
        "command": "build",
        "group": "build",
        "problemMatcher": []
    }
]
```

Save the file by typing Ctrl-S or using the menu command: File → Save.

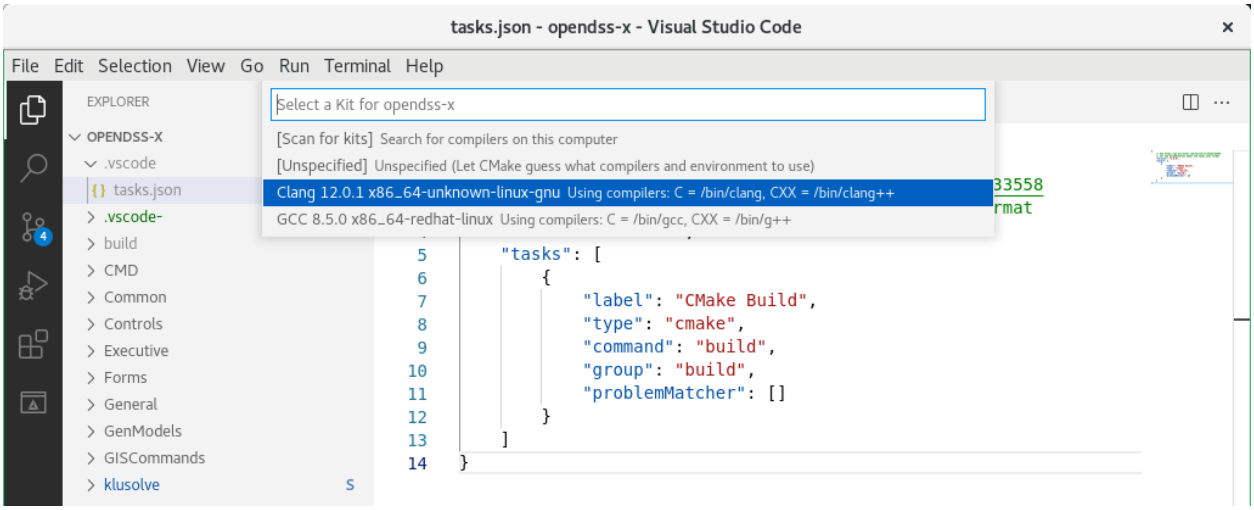


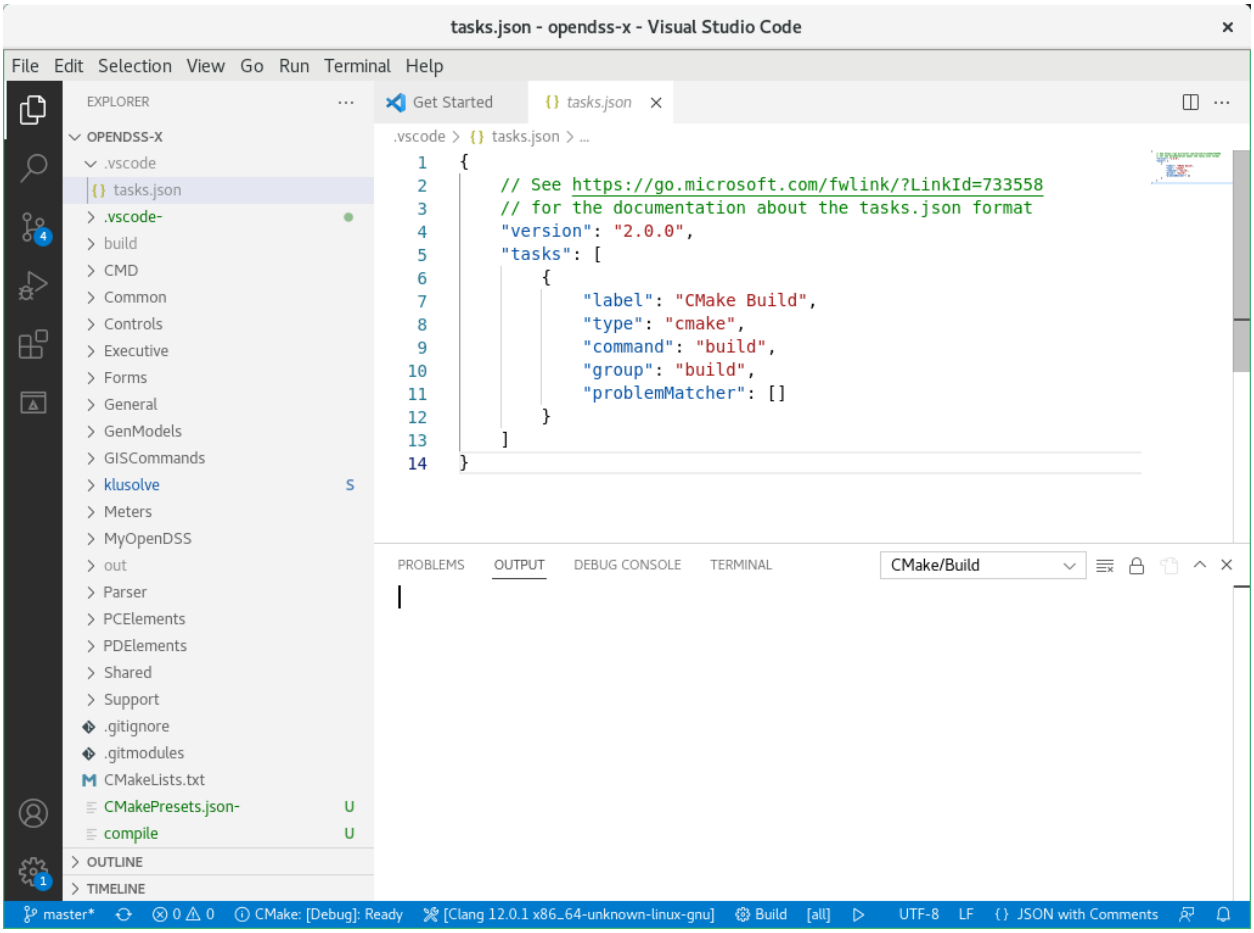
If you get any errors in the PROBLEMS tab at the bottom such as shown below refer to Troubleshooting (Section 5.1.5).



The tasks.json file is called when the user builds the code in VS Code. To configure the Build, one needs to set the build tools kit by clicking on the icon for the build tools in the bottom taskbar (see red box below). If *No Kit Selected* does not appear, then refer to the Trouble Shooting Section 5.1.5.4.

Doing so will result in a prompt for a kit for opendss-x. Select “Clang …” from the list of choices in the drop-down menu. The Clang kit will then appear in the bottom taskbar.





The user can now invoke the build using Clang by clicking the “Build” button in the bottom taskbar (see red box below).

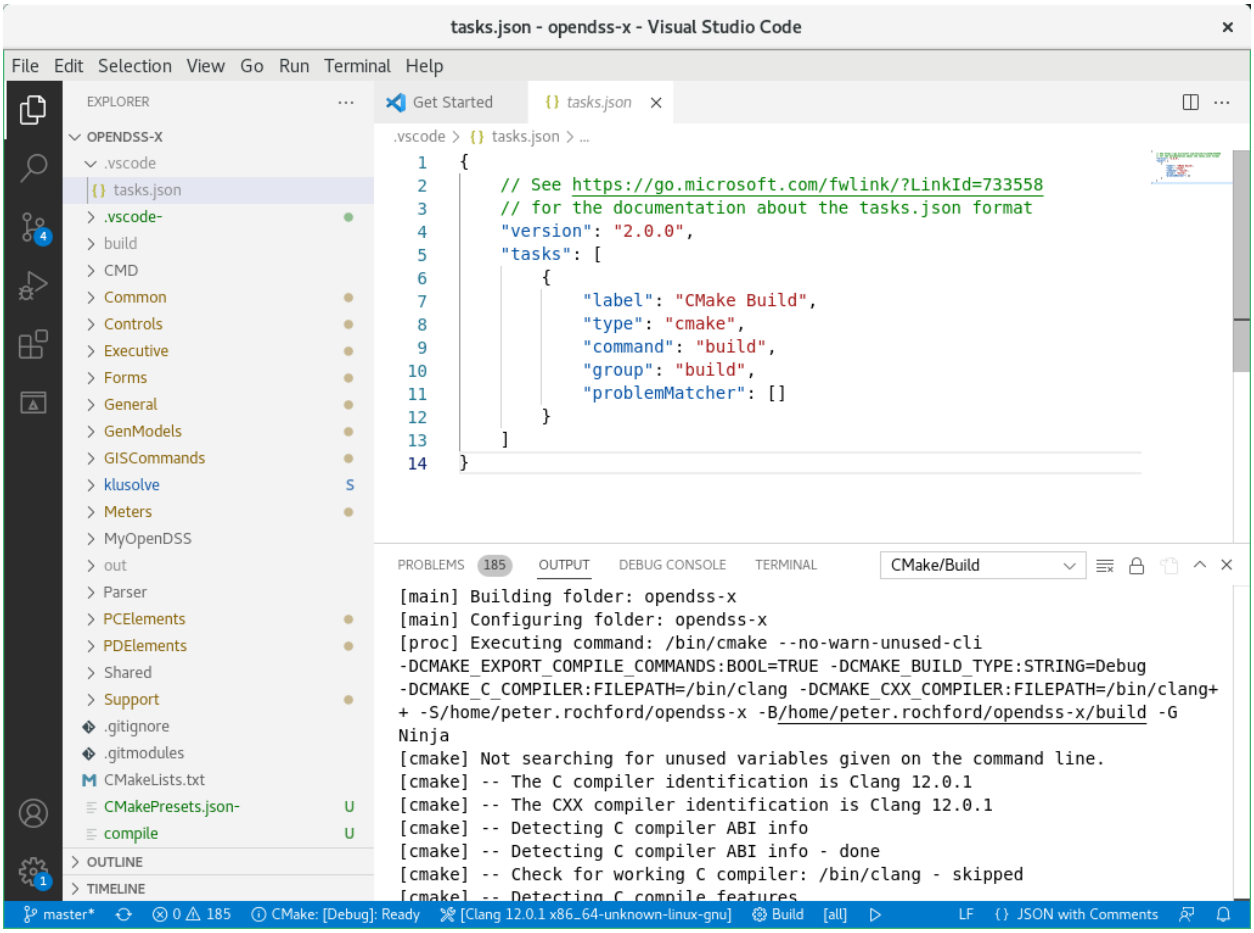


Figure 12. Start of Clang build output

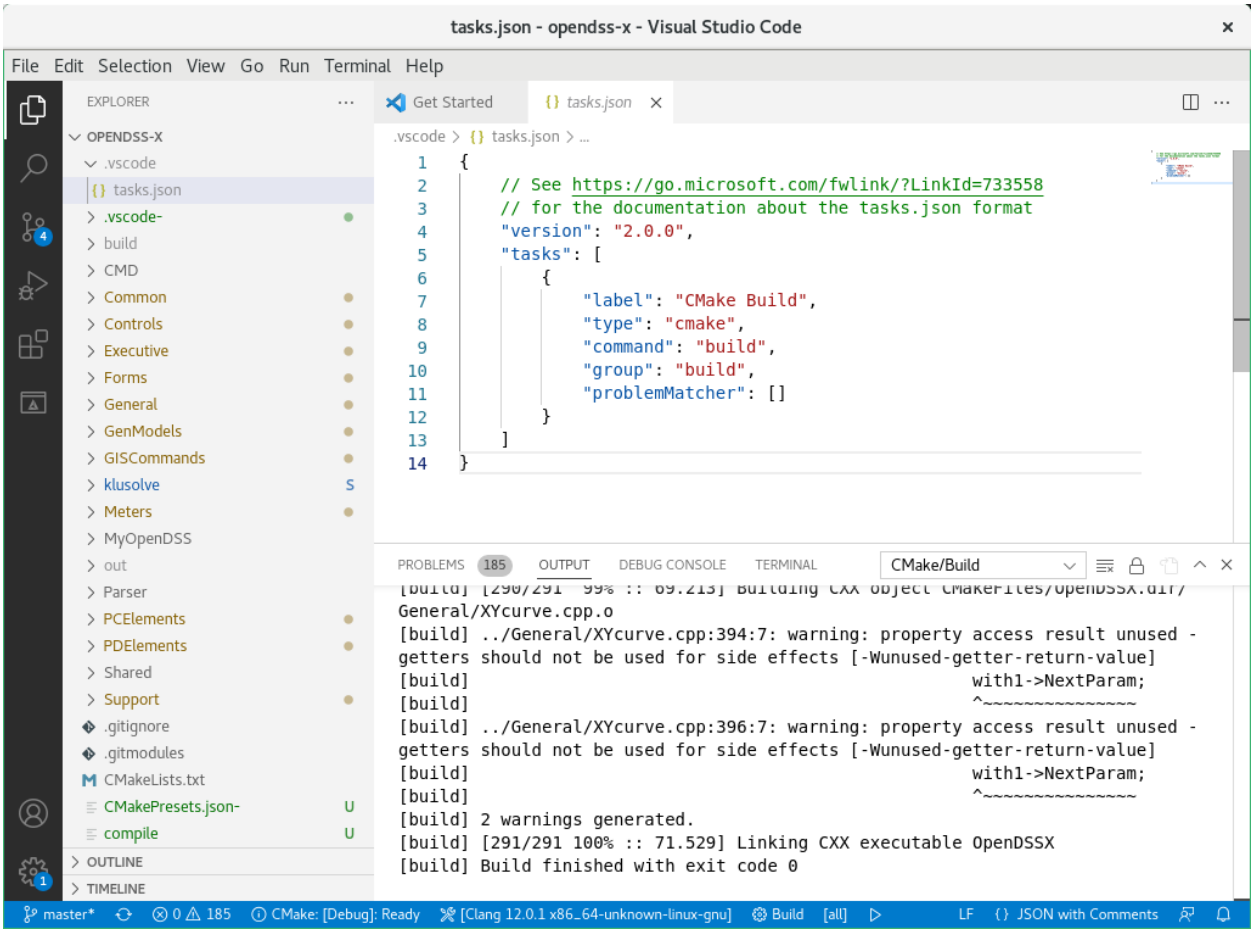
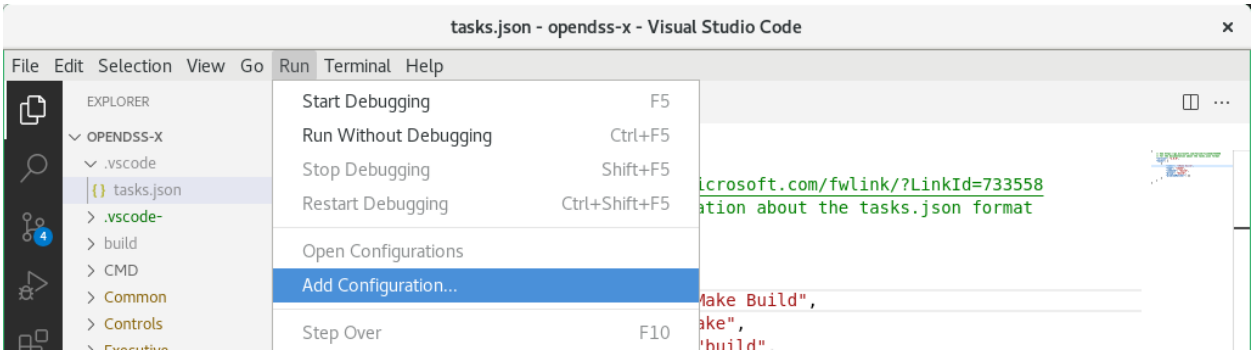
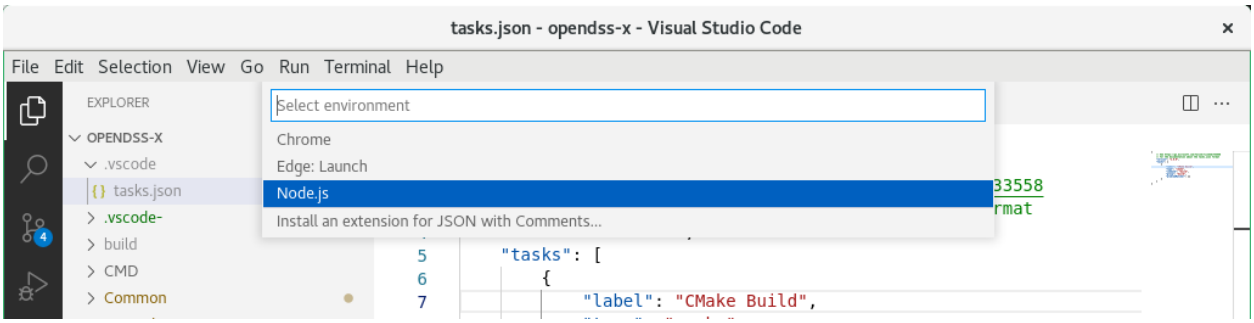


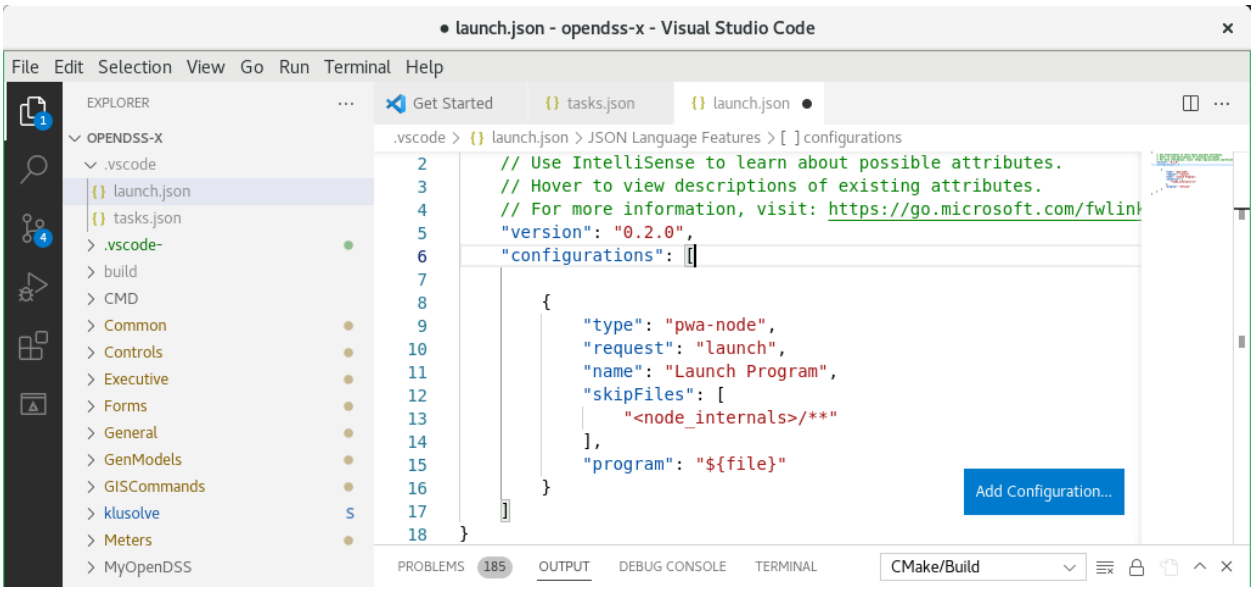
Figure 13. End of successful Clang build output showing no errors

5.1.4 Debugging

The user can setup a debug configuration by creating a launch.json file. To do so, the user must click on *Run → Add Configuration…*, and then choose any option. In this example Node.js is chosen. This will open a new tab window with a default launch.json file.



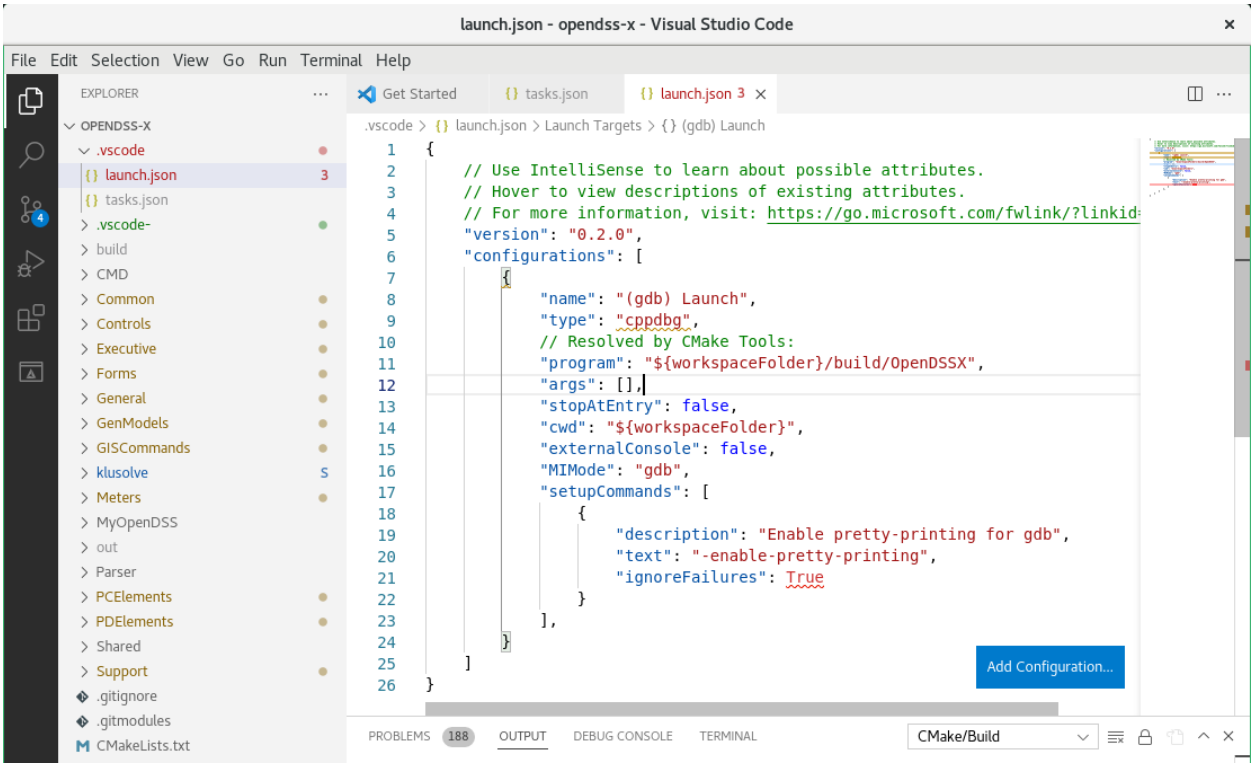




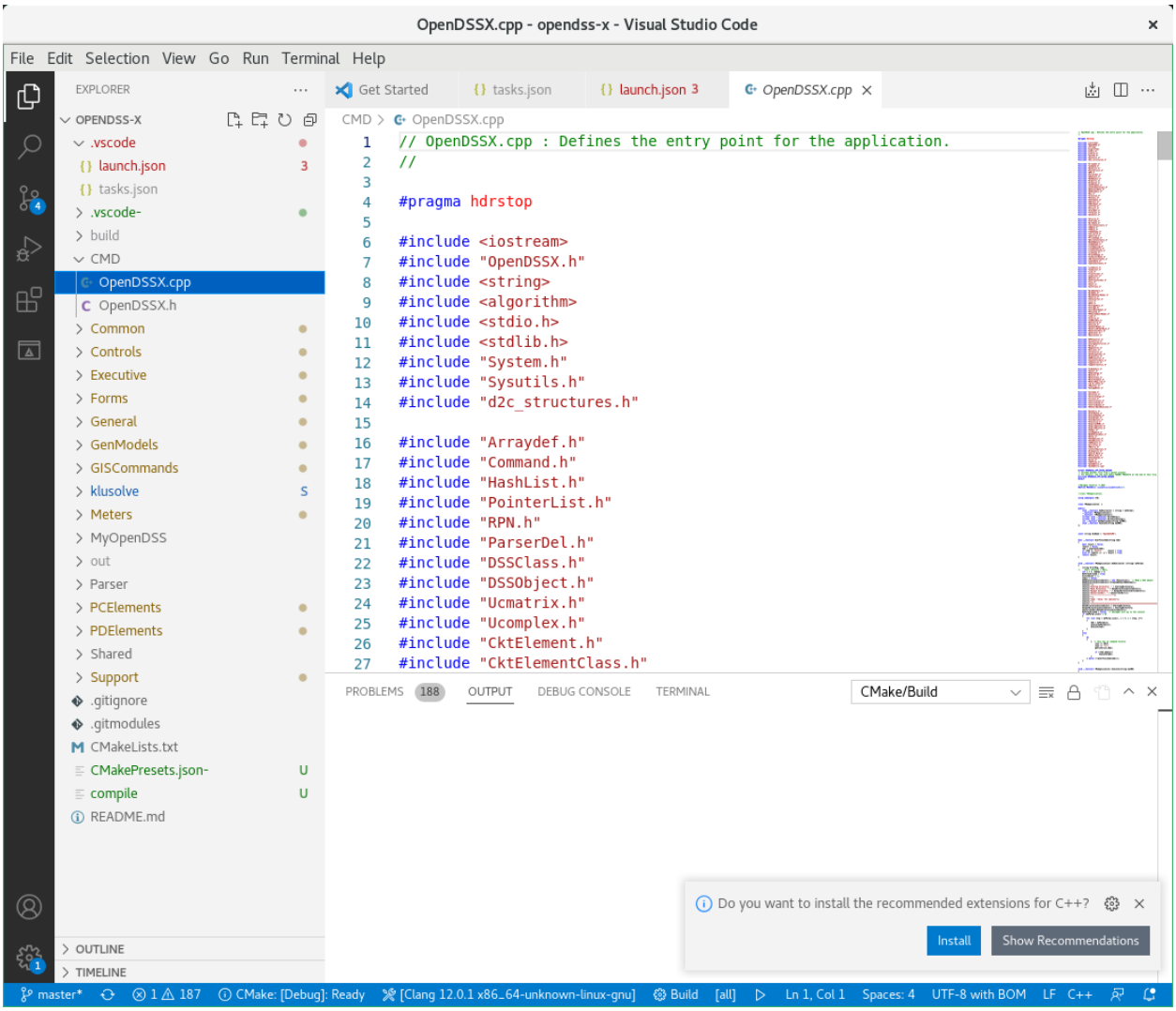
The launch.json can be configured as shown below to allow the debugging through gdb. Save the file when finished copying and pasting the test (Ctrl-S).

```
"configurations": [
  {
    "name": "(gdb) Launch",
    "type": "cppdbg",
    // Resolved by CMake Tools:
    "program": "${workspaceFolder}/build/OpenDSSX",
    "args": [],
    "stopAtEntry": false,
    "cwd": "${workspaceFolder}",
    "externalConsole": false,
    "MIMode": "gdb",
    "setupCommands": [
      {
        "description": "Enable pretty-printing for gdb",
        "text": "-enable-pretty-printing",
        "ignoreFailures": True
```

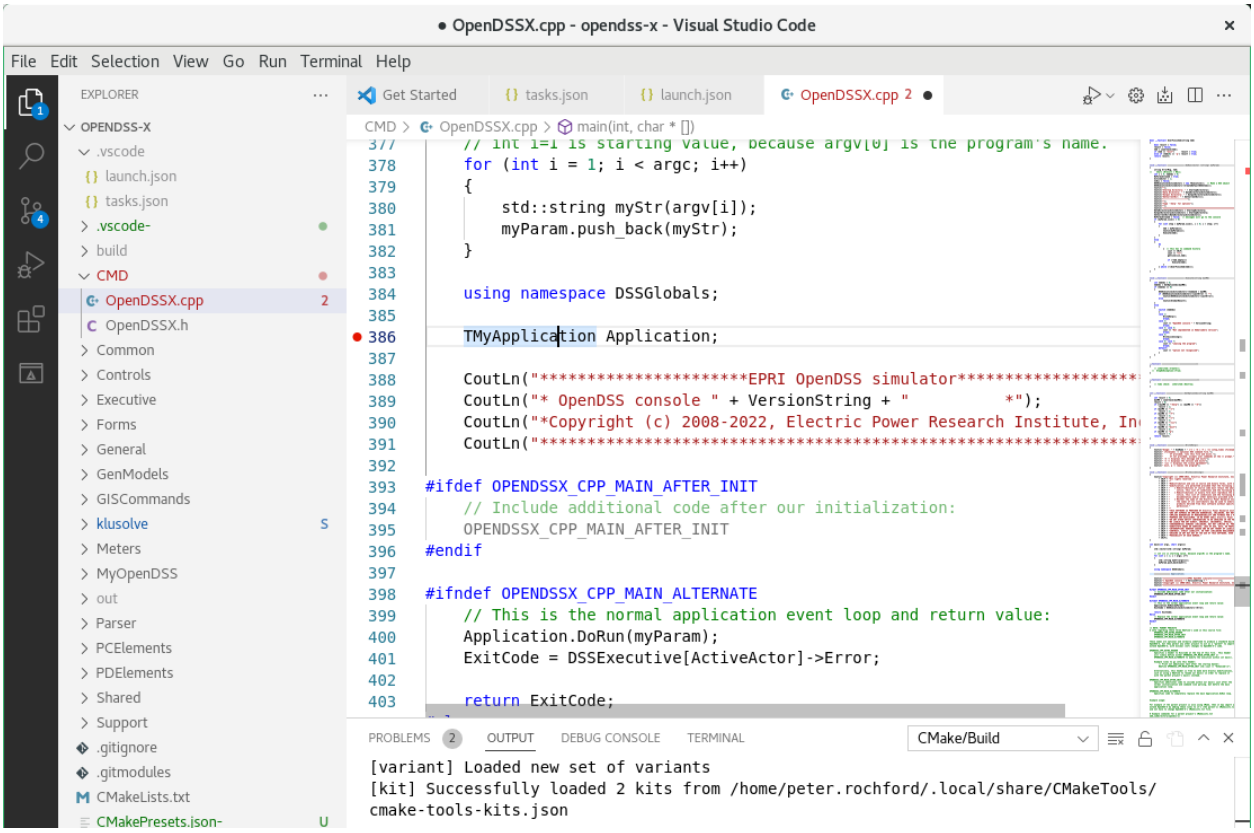
```
        }
      ],
    }
  ]
```



To start debug, the user can set up some breakpoints and hit the F5 key, which should initiate debugging as shown in the figure below. To reproduce the example, open the CMD folder in the explorer window on the left and click on the file OpenDSSX.cpp.



Next scroll down to line 386 and set a breakpoint by clicking in the far-left margin next to a line of code, i.e., in the white space to the left of the line number. A red dot appears in the left margin beside the line number when a breakpoint has been set. You can also select the line and press F9 to set a breakpoint.



If not already installed, you will need to install the C++ Extension Pack to support debugging (cf. Section 5.1.2), which should pop up as a recommendation if necessary.

At this point, VS Code is ready for code editing, running, and debugging so the developer can explore tasks and test the IDE.

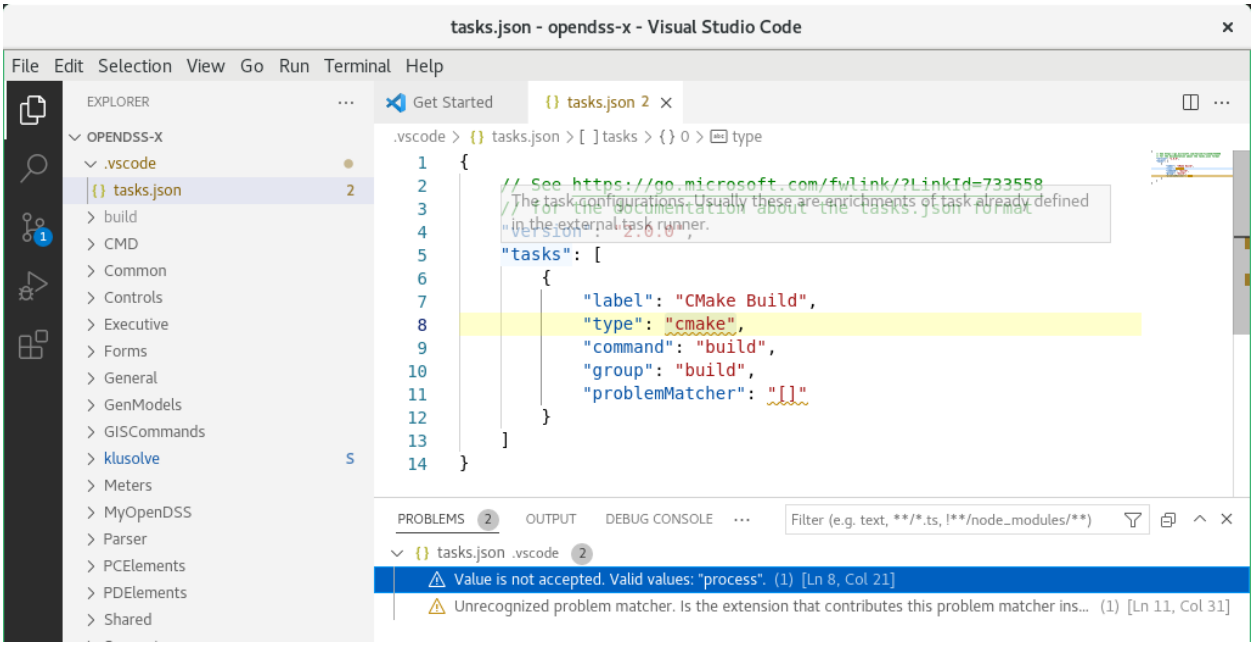
5.1.5 Troubleshooting

5.1.5.1 Cannot Create VS Code Project

If you have opened VS Code before, there may already be a .vscode folder under you home directory, which prevents VS Code from creating one in your project folder. If this is the case, it can be resolved by closing VS Code, deleting the .vscode directory out of the home directory, and typing “code .” from the project folder. If you’re trying to preserve existing settings or other projects, temporarily renaming it instead of deleting might also work.

5.1.5.2 Value is not accepted. Valid values: “process”

This message appears in the PROBLEMS tab at the bottom of the task editor window as shown below. It occurs because VS Code does not recognize cmake.



To resolve the problem, install CMake Tools as described in Section 5.1.2.

5.1.5.3 Unrecognized problem matcher.

This message appears in the PROBLEMS tab at the bottom of the task editor window as shown below. It occurs because the "[]" are enclosed in quotes. Simply remove the quotes.

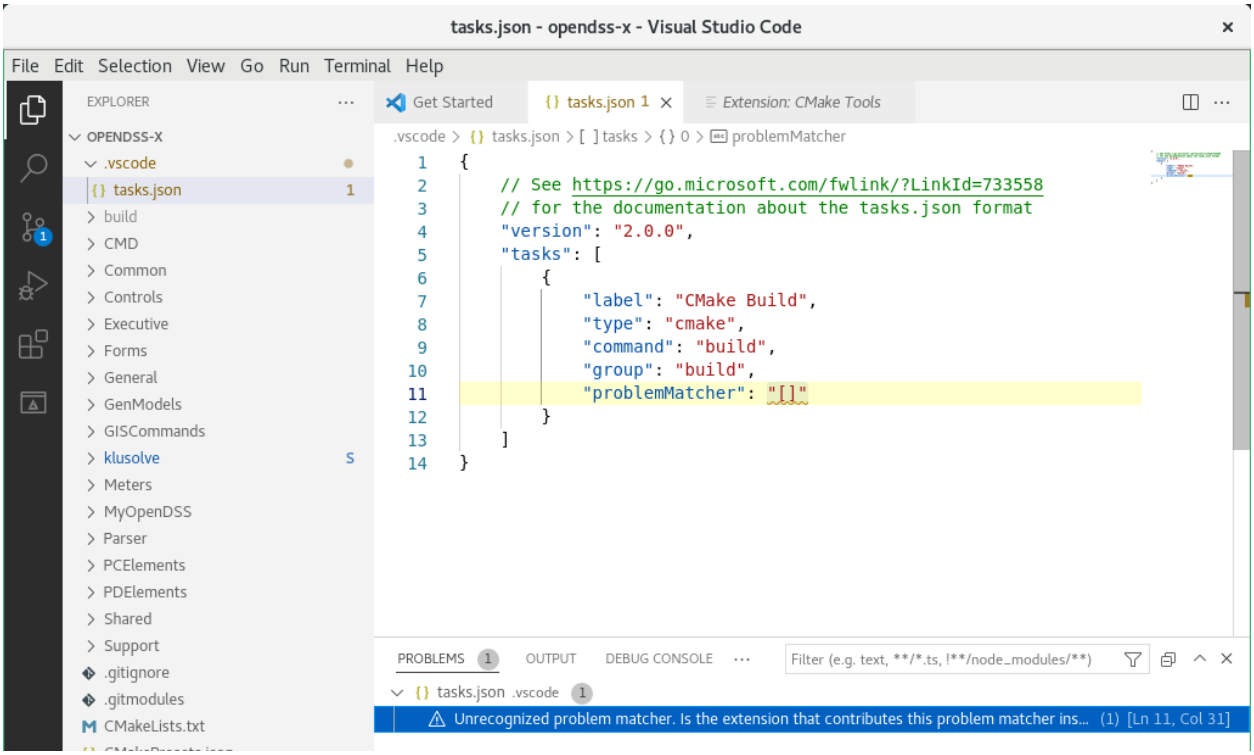


Figure 14. Unrecognized problem matcher because problemMatcher is assigned value using quotes

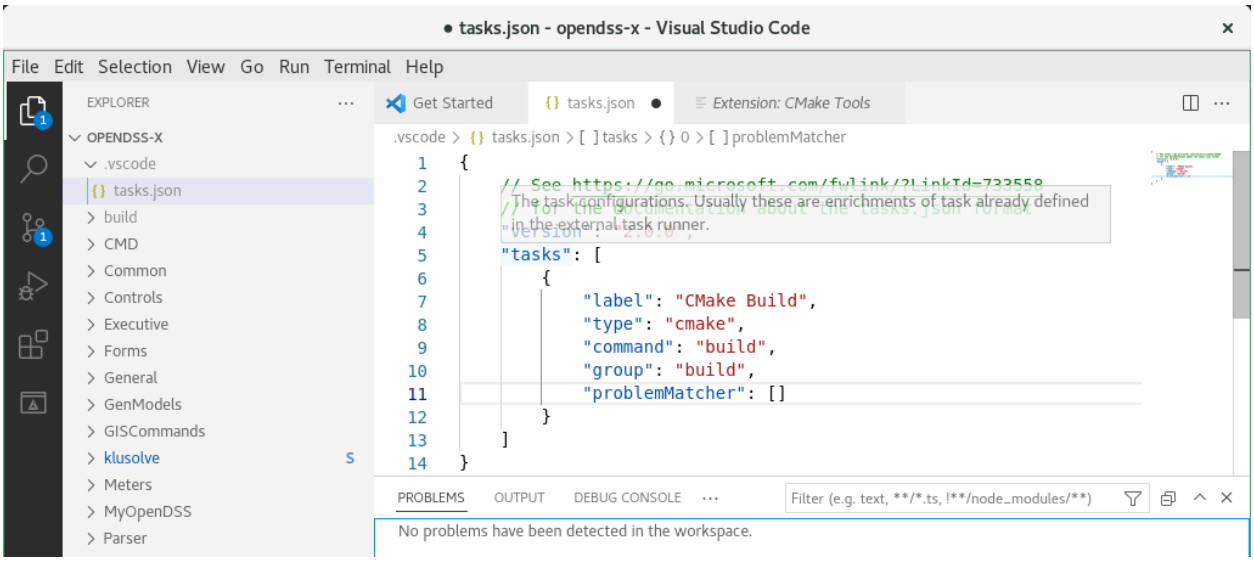
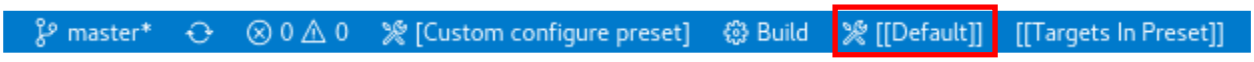


Figure 15. Problem matcher recognizes assignment

5.1.5.4 No Kit Selected Button Missing

If during setup you do not see the No Kit Selected button in the bottom taskbar it is because a CMake presets file already exists in your working code directory. The task bar likely appears as shown below.



To resolve the problem, rename or remove the file CMakePresets.json, e.g.

```
$ mv CMakePresets.json CMakePresets.json-
```

Upon doing so, VS Code should immediately update the taskbar to that shown below.

5.2 Windows

This section describes how to set up VS Code for a Windows. This is the OS in which OpenDSS-X has primarily been done and therefore these instructions are the most complete.

5.2.1 Opening the project

After launching VS Code, select the option “Open a local folder” from the options provided as shown in Figure 16.

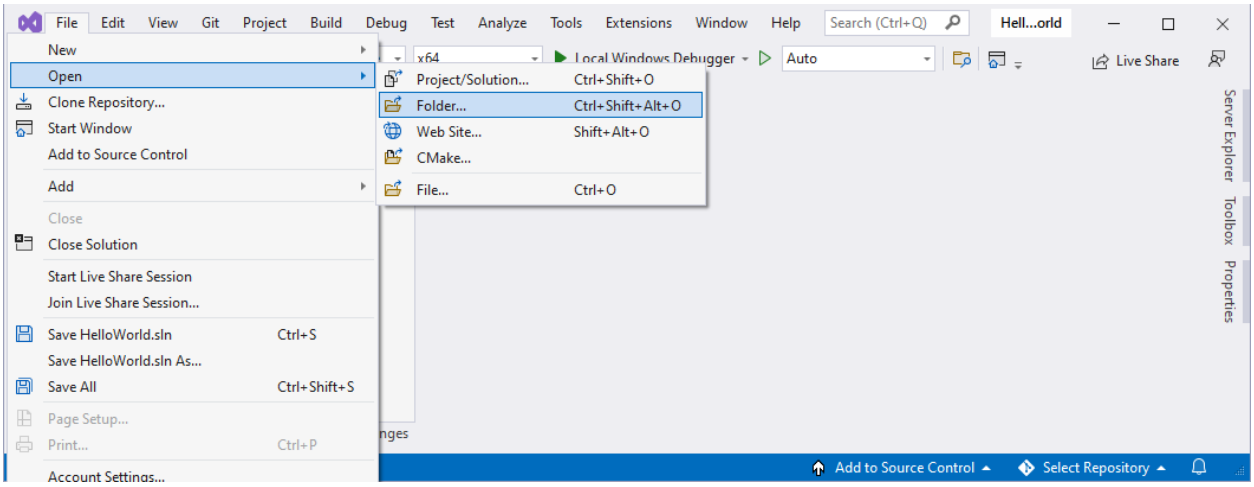


Figure 16. Starting Visual Studio Community

A new browsing window will open for the user to select the project's folder. Please redirect that window to the folder CMD within the OpenDSS-X clone created in Section 3. Once you are within that folder select the option "Select folder" as shown in Figure 17.

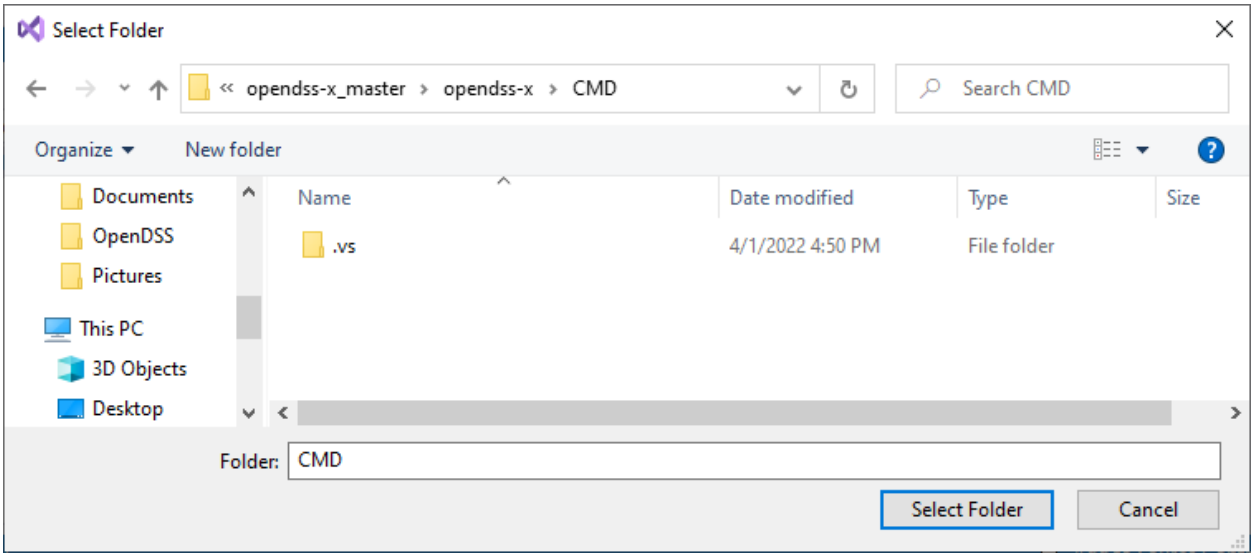


Figure 17. Find the project's source folder

With that, Visual Studio will setup the environment using the CMake files within that folder. The environment should look as shown in Figure 18.

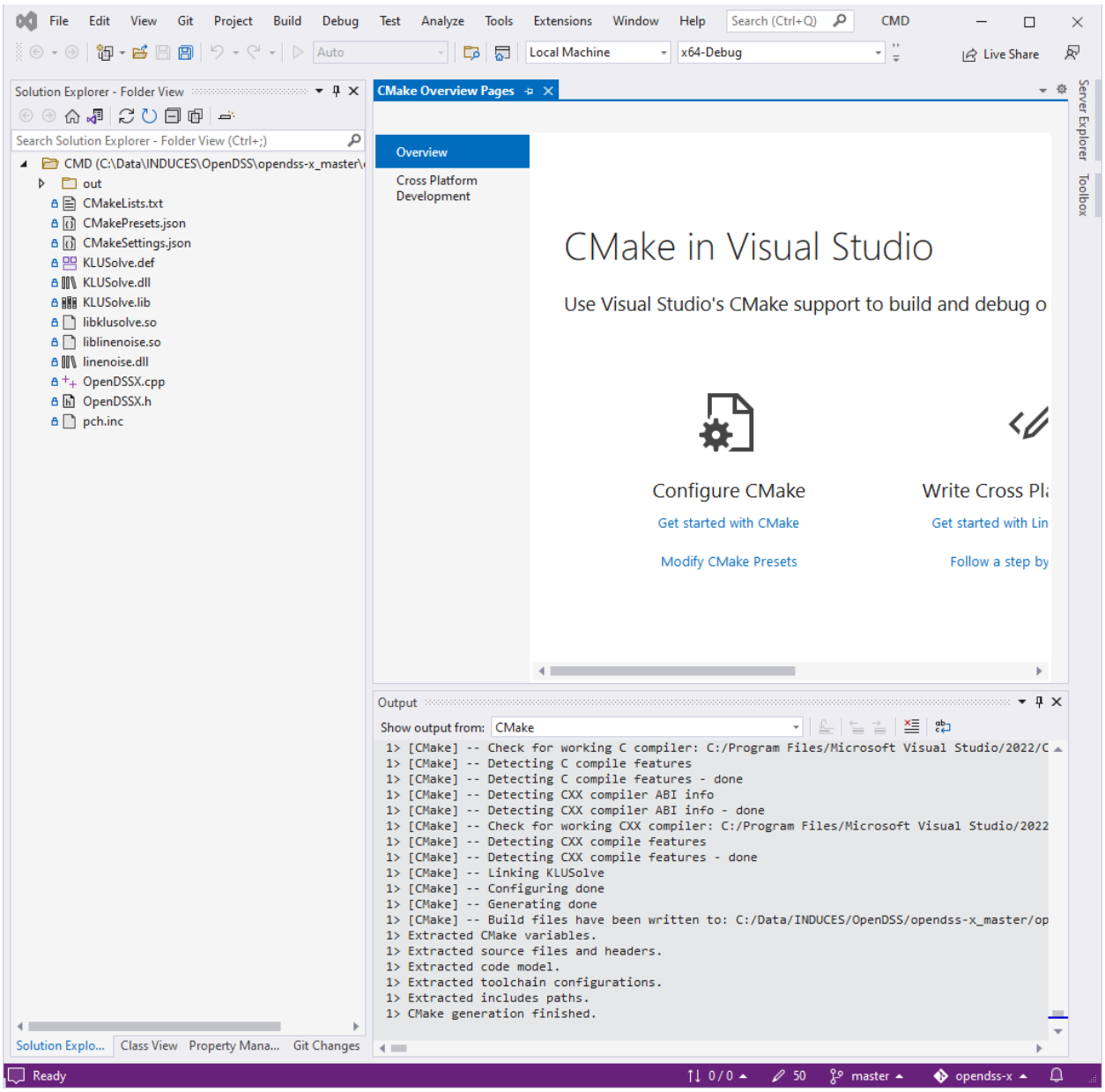


Figure 18. Environment ready

At the left of the screen (normally), the user will find the project's tree, which will display only the files contained within the CMD folder. The first step will be to define the startup item. To do so, at the project's tree right click over the file called OpenDSSX.cpp and select "Set as startup item" from the pop-up menu displayed as shown in Figure 19. That action will tell Visual Studio what's the starting point in the project when compiling using CMake.

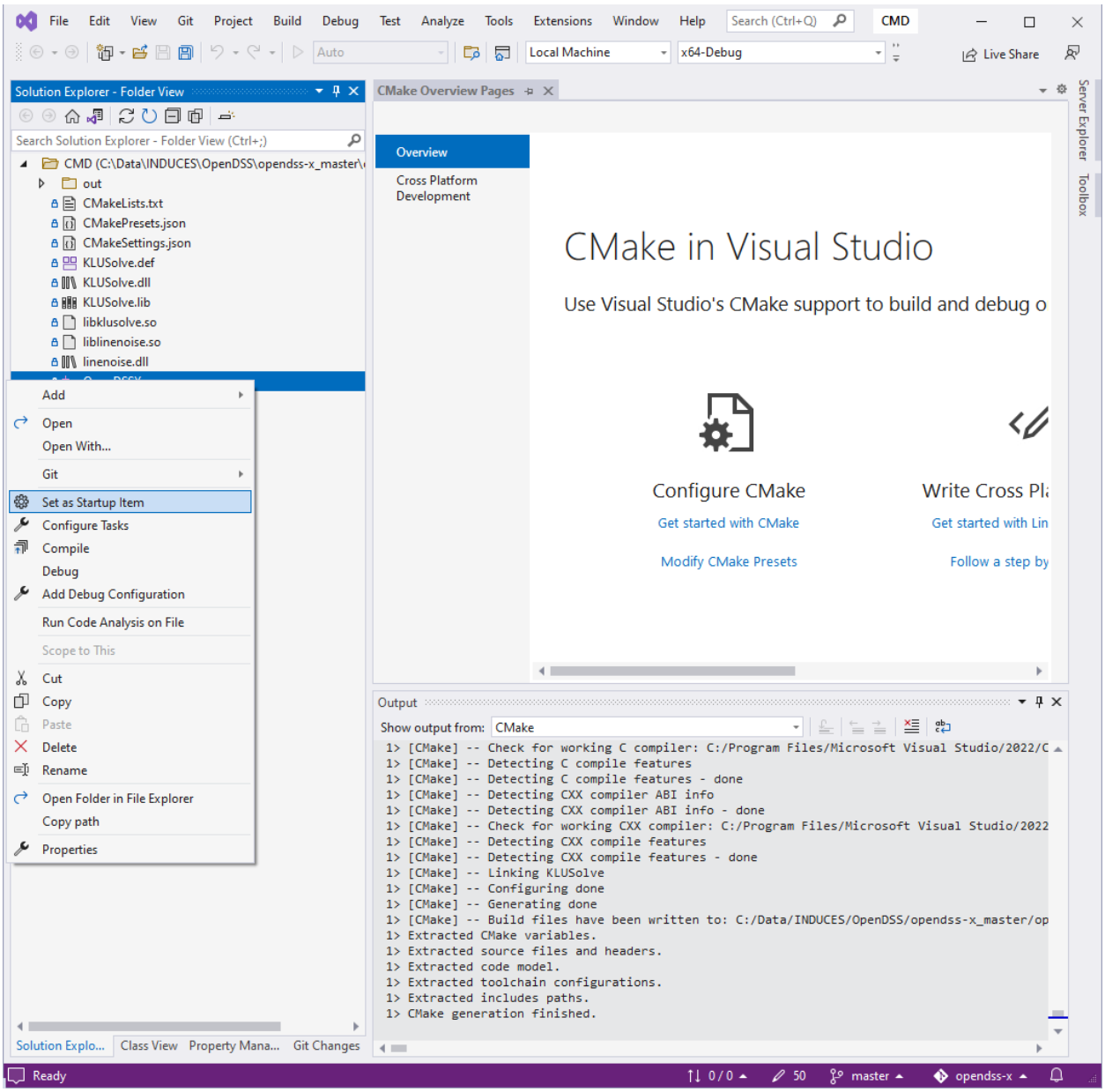


Figure 19. Setting the startup item

Next, compile the project. To do so go to the menu item Build → Build All as shown in Figure 20. That should build the project and throw no errors (Figure 21). If there is any error, it should be notified at the output window at the bottom of the IDE.

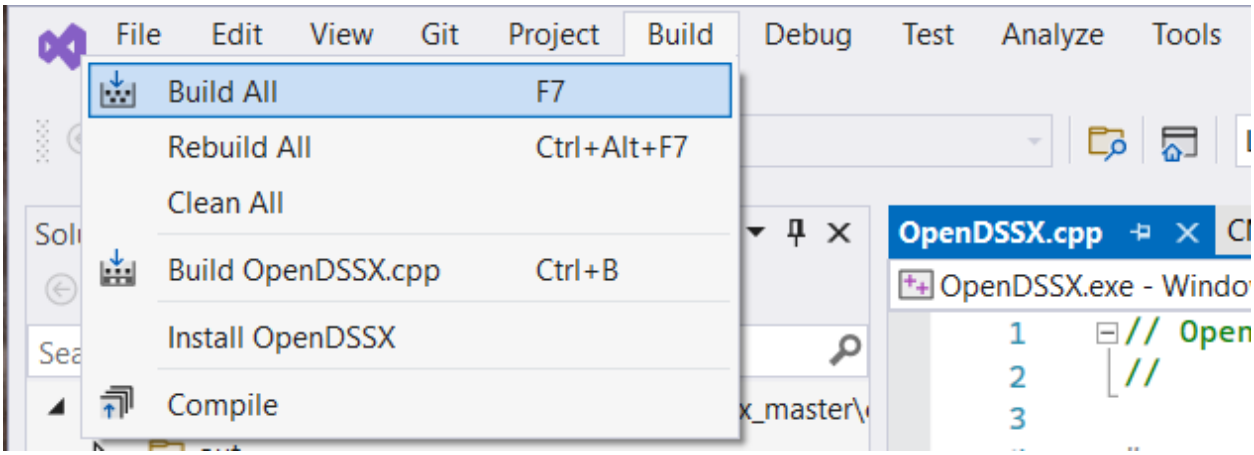


Figure 20. Building the Project using the Visual Studio IDE

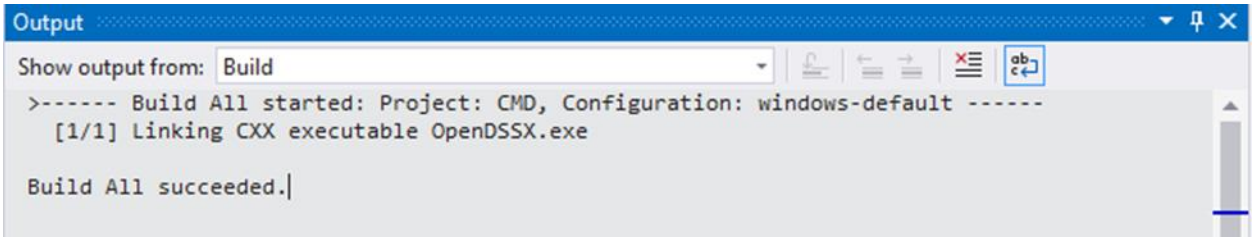


Figure 21. Successful build of OpenDSS-X

5.2.2 Troubleshooting

5.2.2.1 Cannot open file KLUSolve.lib

In the first attempt of running the program you may get an error message telling you that a link file is missing as shown below.

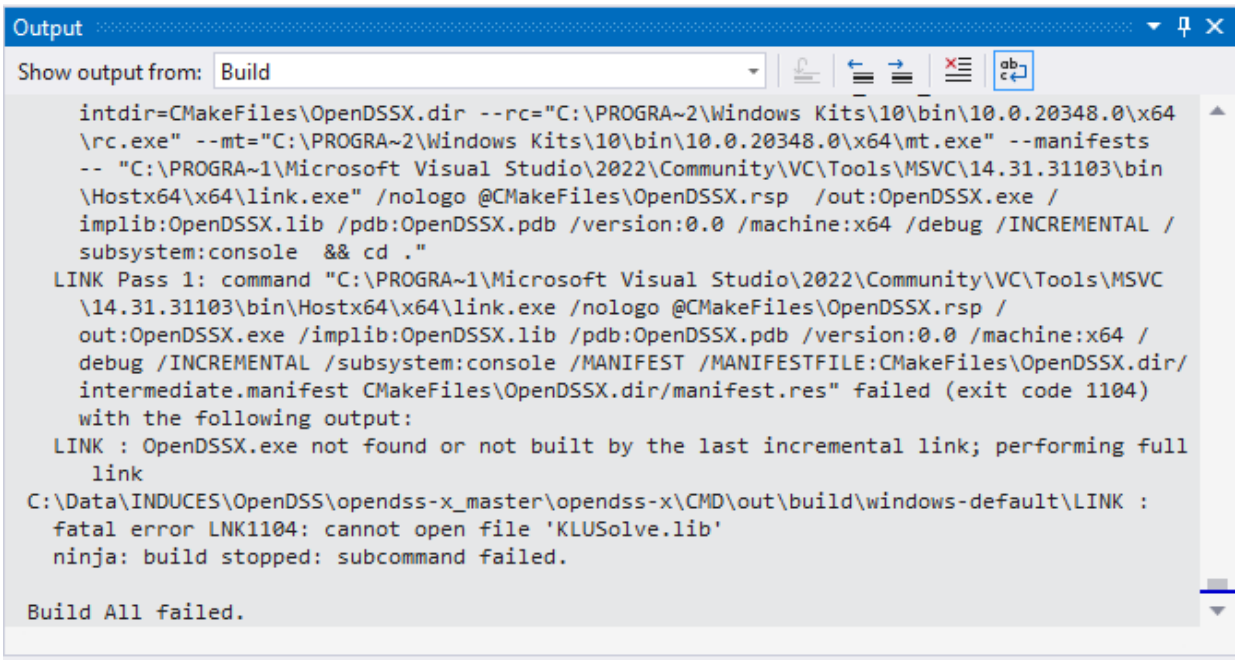


Figure 22. Issues compiling OpenDSS-X for the first time

This is because the output directory where the executable generated by CMake is located cannot find some dependencies. To solve it, go to the folder CMD within the project's directory and copy the files KLUSolve.dll and KLUSolve.lib into the folder …CMD\out\build\windows-default.

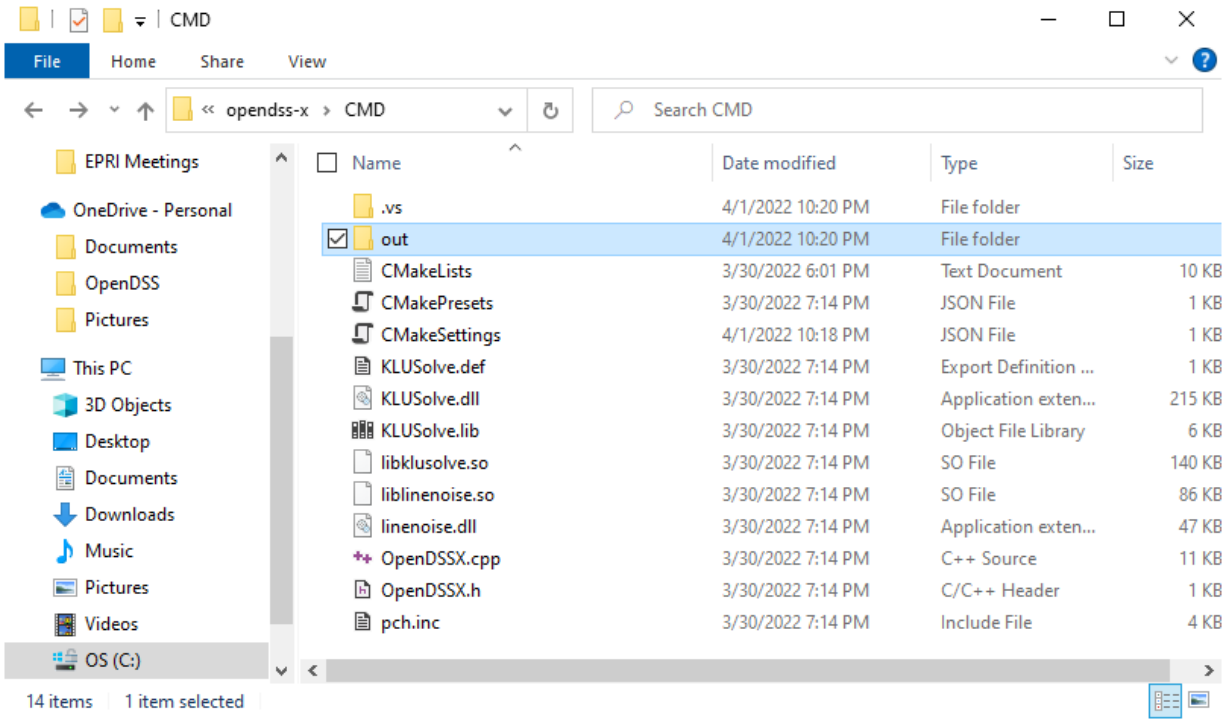


Figure 23. Source for the missing files

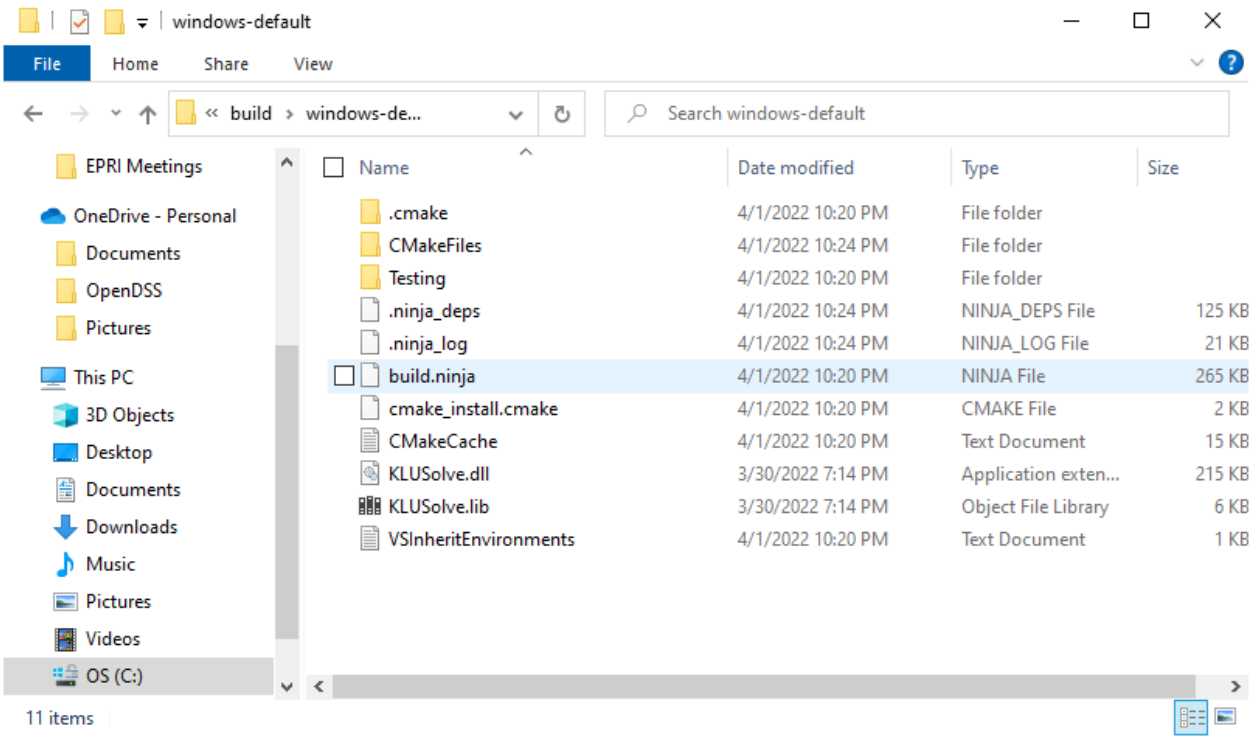


Figure 24. Destination for the missing files

Then, try to run the program again, the error message should disappear, and you should be seeing something like the image displayed below.

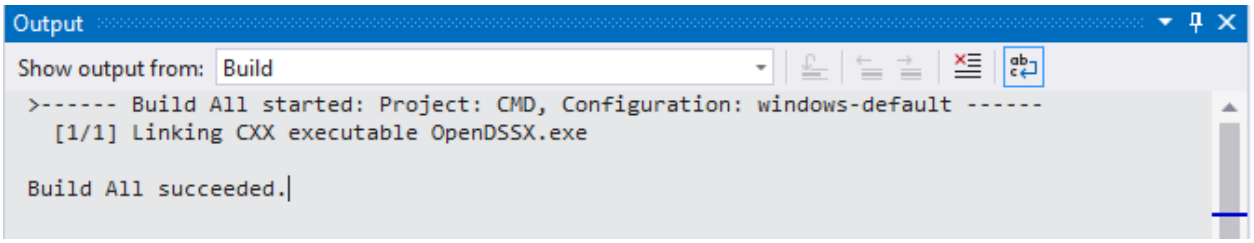


Figure 25. Output when OpenDSS-X was compiled successfully

6 Building Code Using CMake

OpenDSS-X can be built using CMake from a command line within Linux. This entails a 2-step process where CMake is first run to generate the makefiles and make is then run to build the C++ application. Information on CMake can be found on the distribution site for the software application (CMake, 2022).

Obtain a copy of the OpenDSS-X code base as described in Section 3.3. This should provide you a standard cmake project that should enable you to build OpenDSS-X with the Clang compiler front end for the C++/C programming languages within a command line environment.

To build OpenDSS-X using CMake do the following steps.

1. Open a terminal window in the Linux operating system.

2. Move to the top-level directory containing the OpenDSS-X source code.

   $ cd opendss-x

3. If you desire to build the code with debugging capabilities, edit the file CMakePresets.json and change the CMAKE_BUILD_TYPE to:

   “CMAKE_BUILD_TYPE”: “Debug”

   The default of “Release” produces an optimized build.

3. Run the build script.

   $ ./compile

This will produce messages to the terminal window showing progress along with any errors and warnings like that shown below. A log file of the build can be found in build/make.txt, while a list of the errors and warnings can be found in build/make_err.txt.

peter.rochford@ip-10-0-0-228:~/opendss-x

File Edit View Search Terminal Help

```
RHEL$ ll compile
-rwx------. 1 peter.rochford peter.rochford 1214 Aug 11 22:14 compile
RHEL$ ./compile
-- The C compiler identification is Clang 12.0.1
-- The CXX compiler identification is Clang 12.0.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /bin/clang - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /bin/clang++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Failed
-- Looking for pthread_create in pthreads
-- Looking for pthread_create in pthreads - not found
-- Looking for pthread_create in pthread
-- Looking for pthread_create in pthread - found
-- Found Threads: TRUE
Platform assumption: Linux or UNIX
-- Configuring done
-- Generating done
-- Build files have been written to: /home/peter.rochford/opendss-x/build
[  2%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_add.
[  1%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_amd.
[  2%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_chol
[  3%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_comp
[  1%] Building CXX object klusolve/CMakeFiles/klusolve_all.dir/KLUSolve/Source/KLUS
[  3%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_chol
[  0%] Building CXX object klusolve/CMakeFiles/klusolve_all.dir/KLUSolve/Source/KLUS
[  3%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_coun
[  3%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_dfs.
[  3%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_dmpe
[  4%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_cums
[  5%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_drop
[  5%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_dupl
[  5%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_drop
[  6%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_entr
[  6%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_gaxp
[  6%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_etre
[  7%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_happ
[  6%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_erea
[  7%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_fkee
[  8%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_hous
[  8%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_ipve
[  8%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_leaf
[  9%] Building C object klusolve/CMakeFiles/klusolve_all.dir/CSparse/Source/cs_load
```

…

```
[ 97%] Building CXX object CMakeFiles/OpenDSSX.dir/General/Spectrum.cpp.o
[ 97%] Building CXX object CMakeFiles/OpenDSSX.dir/General/PriceShape.cpp.o
[ 99%] Building CXX object CMakeFiles/OpenDSSX.dir/General/WireData.cpp.o
[ 98%] Building CXX object CMakeFiles/OpenDSSX.dir/General/TempShape.cpp.o
[ 98%] Building CXX object CMakeFiles/OpenDSSX.dir/General/TSLineConstants.cpp.o
[ 98%] Building CXX object CMakeFiles/OpenDSSX.dir/General/TSData.cpp.o
[ 99%] Building CXX object CMakeFiles/OpenDSSX.dir/General/XYcurve.cpp.o
[ 99%] Building CXX object CMakeFiles/OpenDSSX.dir/General/XfmrCode.cpp.o
[100%] Building CXX object CMakeFiles/OpenDSSX.dir/GenModels/gencls.cpp.o
[100%] Linking CXX executable OpenDSSX
[100%] Built target OpenDSSX
```

OpenDSS-X should successfully build in less than 1 minute of elapsed time and create the executable: build/OpenDSSX-cui.

To build individual files that have changed rather than all the files do the following steps.

1. Confirm you are in the top level induces directory.

   $ cd ~/opendss-x

2. Run CMake using the standard make command.

   $ CC=clang CXX=clang++ cmake -S . -B build

   $ cmake --build build -j12

```
RHEL$ CC=clang CXX=clang++ cmake -S . -B build
Platform assumption: Linux or UNIX
-- Configuring done
-- Generating done
-- Build files have been written to: /home/peter.rochford/induces/build
RHEL$ cmake --build build -j12
Consolidate compiler generated dependencies of target klusolve_all
[ 51%] Built target klusolve_all
Consolidate compiler generated dependencies of target OpenDSSX
[100%] Built target OpenDSSX
RHEL$
```

7 Test Case

Once the console application has been compiled and is up and running as shown in Figure 26, the console will accept any of the OpenDSS commands described at the OpenDSS Reference Guide (Dugan & Montenegro, 2020). For example, if you try *get NumCPUs*, the console will return the number of CPUs available in the local computer as shown in Figure 27.

For compiling an existing script, use the command *compile* as described in the reference guide. The *Compile* command is followed by the path to the script that will be compiled. With this, OpenDSS-X will execute commands within the script, that may include the circuit model description, solve commands

and/or export/show commands. An example displaying the compilation of the IEEE 8500 node test case is displayed in Figure 28.

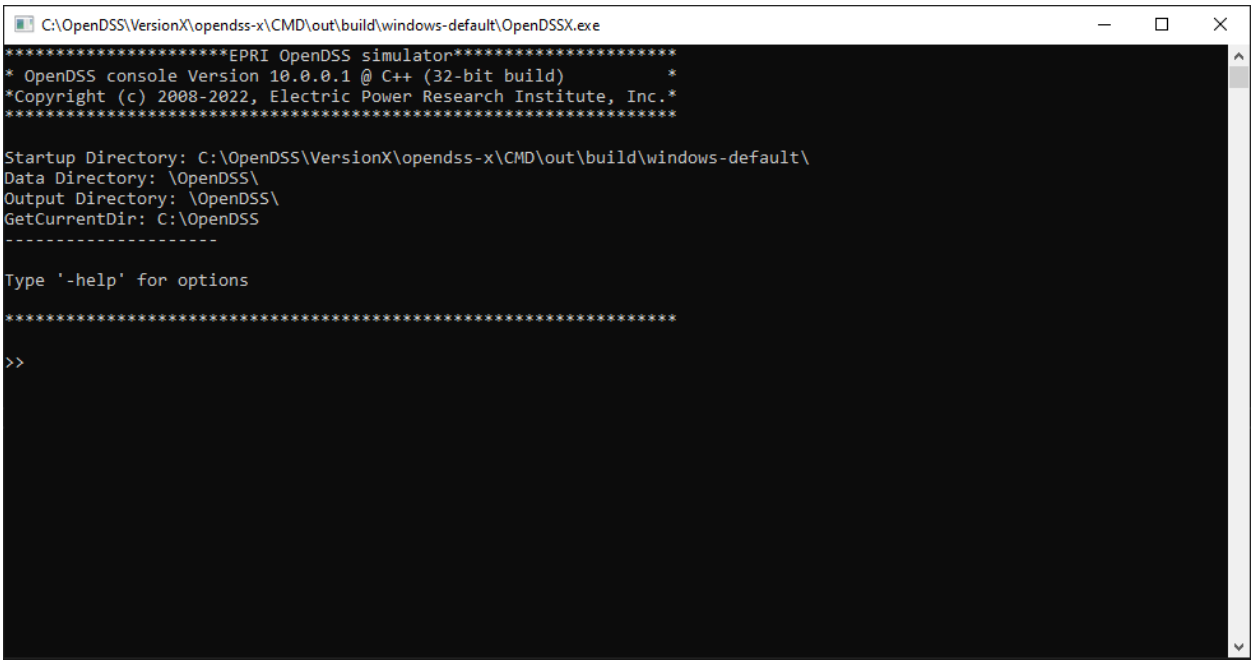


Figure 26. OpenDSS-X console up and running.

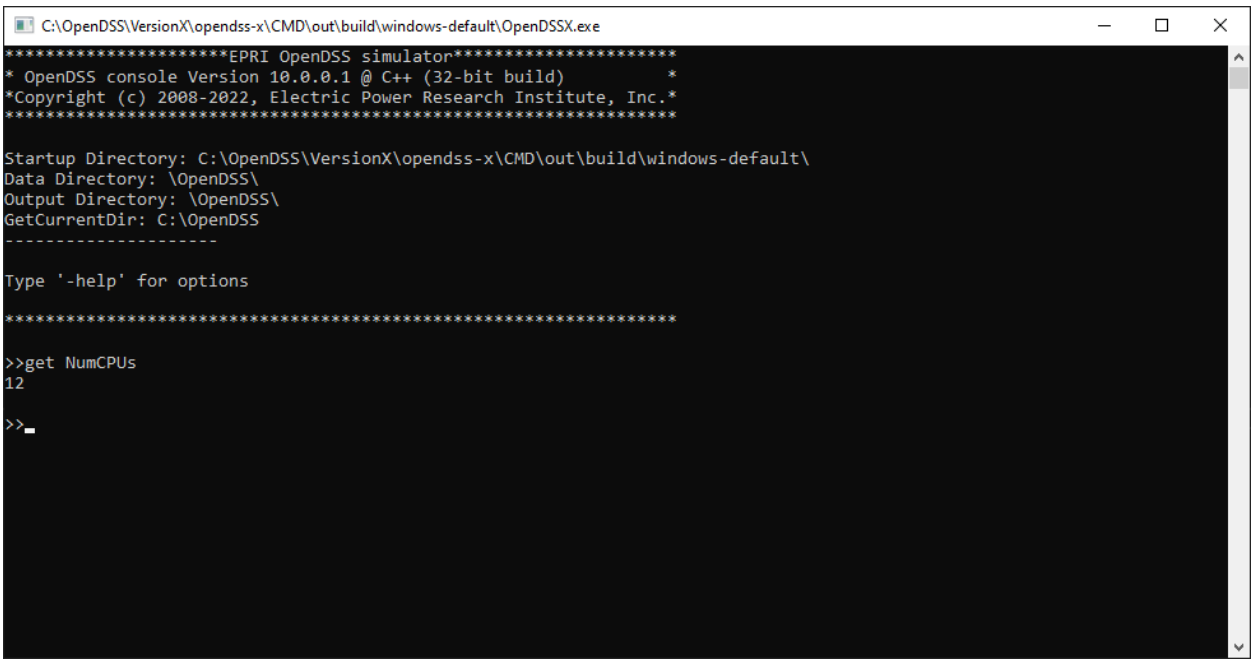


Figure 27. Requesting the number of CPUs in the local computer

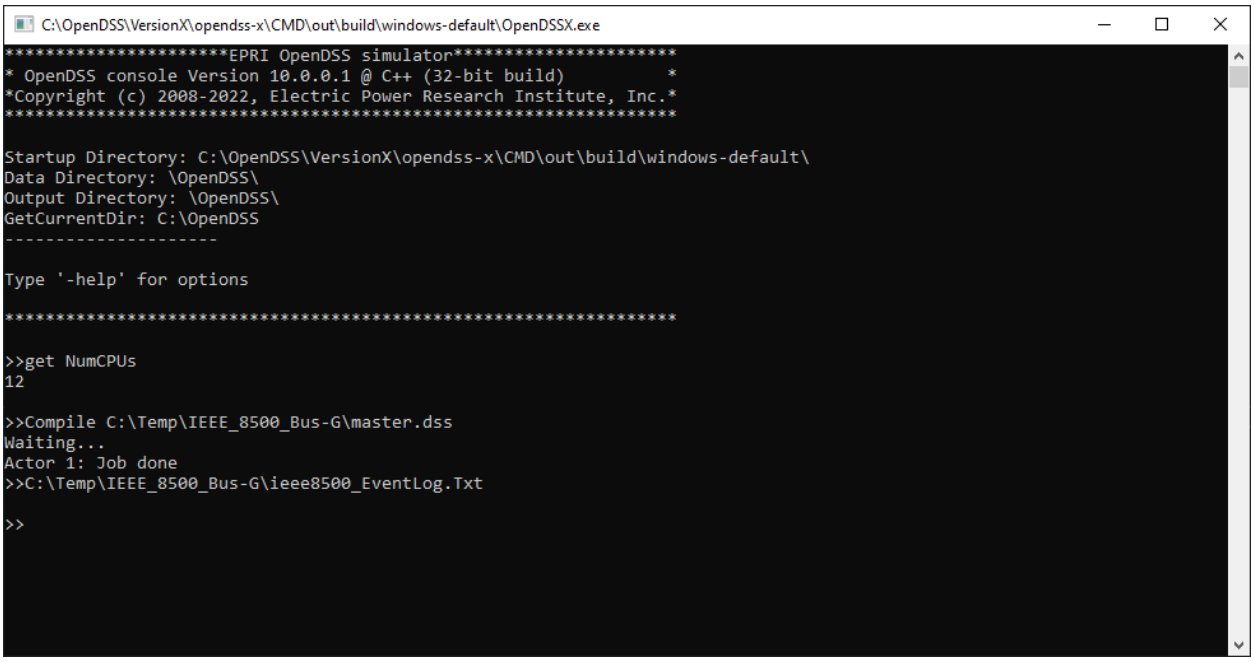


Figure 28. Executing a script solving the IEEE 8500 node test system

As can be seen in Figure 29, OpenDSS-X will report the simulation job status through the console. The result of the programmed job can succeed or fail. In both cases, the user will get feedback just as in the original OpenDSS version. Once the last job is finished, the user can keep typing commands to keep interacting with the active simulation as shown in Figure 24.

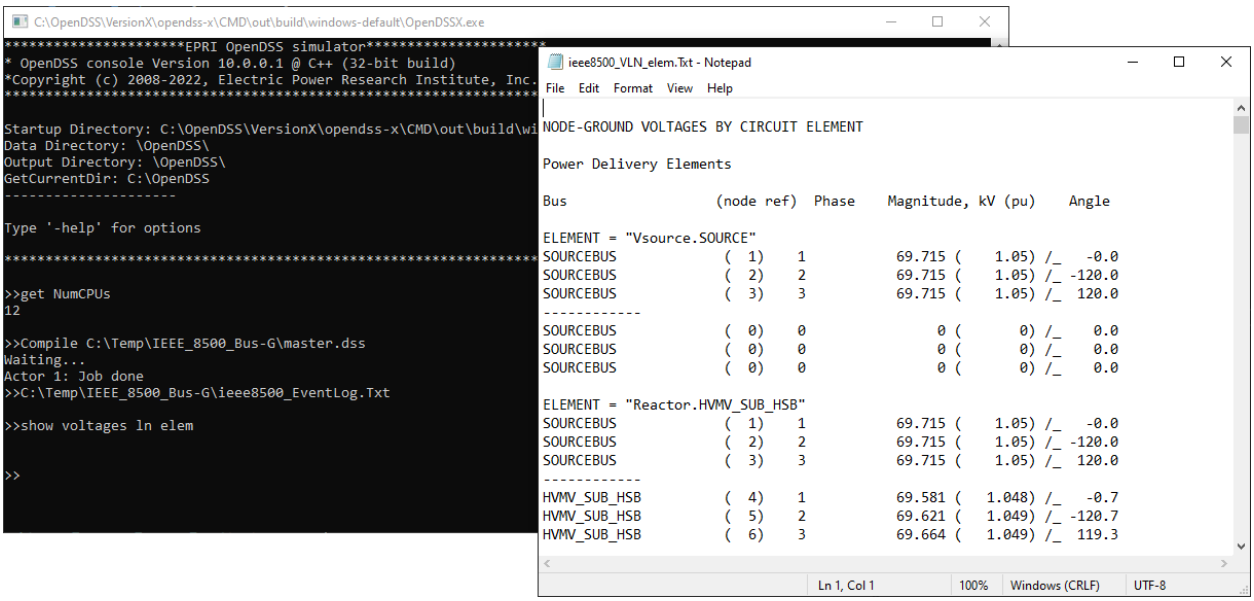


Figure 29. Executing commands after compiling a script

If the intent is to execute a script from outside the console app, this can be done by using command line arguments. The command line arguments need to be enclosed in double quote marks. To compile a

script using command line arguments open a new command prompt window and locate the directory where the OpenDSSX.exe application is located as shown in Figure 30. By default, if OpenDSS-X is built using Visual Studio, the application will reside in a subfolder of CMD\out\build\windows-default.

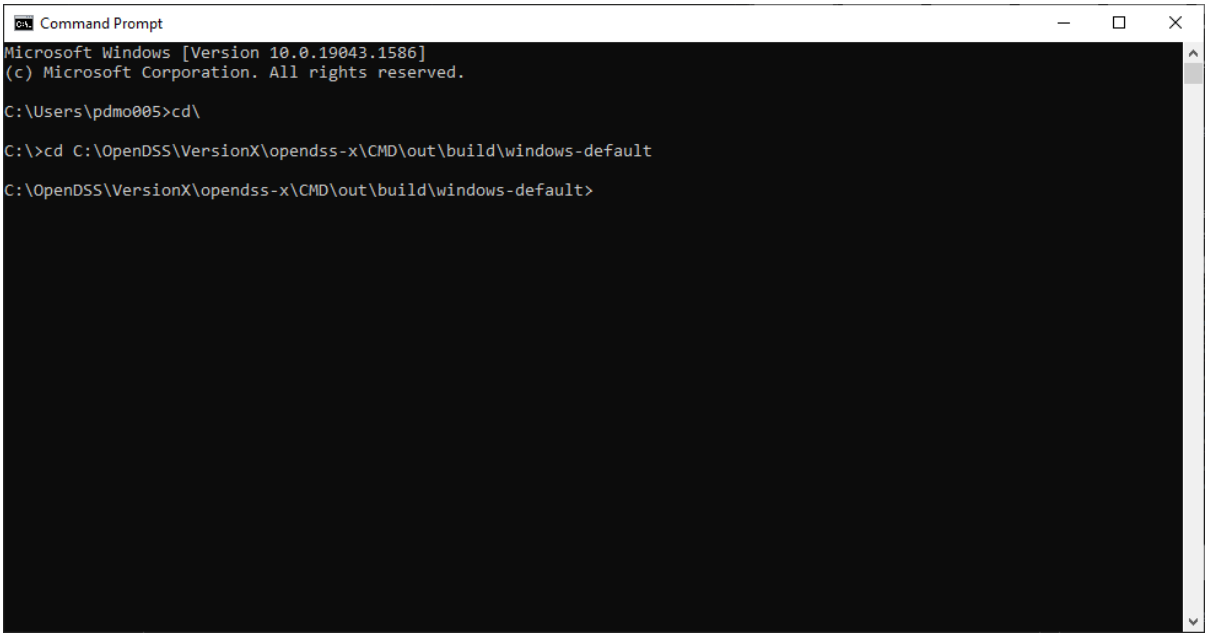


Figure 30. Starting command prompt window (MS Windows OS)

Once located at the application folder, call the application openDSSx.exe followed by the arguments describing the command to execute. For this example, the command will be compiled pointing to the IEEE 8500 node test system used in Figure 23. This action can be seen in Figure 31.

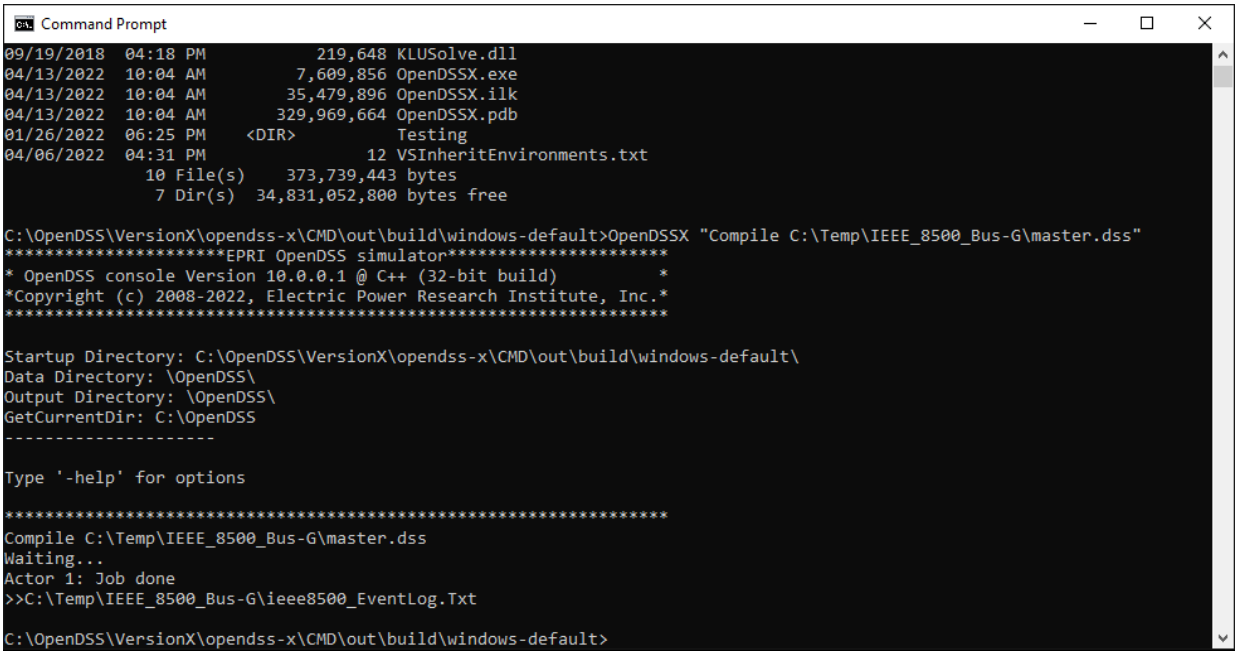


Figure 31. Executing OpenDSS-X commands using command prompt

The user can add as many arguments as needed when calling OpenDSS-X using this procedure, each argument must be separated by spaces and enclosed in double quotes.

8 References

CMake. (2022, April 3). *Running CMake*. Retrieved from CMake: https://cmake.org/runningcmake

Davis, T. (2006). *Direct Methods for Sparse Linear Systems.* Philadelphia, PA: SIAM.

Dugan, R. C., & Montenegro, D. (2020). *The Open Distribution System Simulator (OpenDSS) Reference Guide.* Electric Power Research Institute. Washington, DC: Electric Power Research Institute. Retrieved from https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Distrib/Doc/OpenDSSManual.pdf

Electric Power Research Institute. (2022, April 2). *OpenDSS*. Retrieved from EPRI: https://www.epri.com/pages/sa/opendss

Fu, F. (2019). *OpenDSS Tutorial and Cases.* Iowa State University, Department of Electrical and Computer Engineering. Ames, Iowa: Iowa State University. Retrieved from https://www.coursehero.com/file/87693420/EE653-OpenDSS-Tutorial-and-Casespptx

McDermott, T. (2013). *KLUSolve 1.0*. (EnerNex Corporation) Retrieved from https://github.com/rwl/klusolve

9 Acronyms

DER	Distributed Energy Resource
DSS	Distribution System Simulator
GUI	Graphical User Interface
IDE	Interactive Development Environment
LGPL	Lesser General Public License
OS	Operating System
UI	User Interface

Appendix A IDE Setup

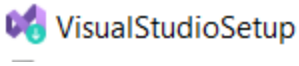
This appendix describes how to install various Interactive Development Environments (IDEs) commonly used for OpenDSS-X C++ software development.

A.1 Visual Studio Community (VS Code)

This is a fully featured, extensible, free IDE for creating modern applications for Android, iOS, Windows, Linux, as well as web applications and cloud services.

A.1.1 Installation

These instructions are for installing VS Code on Windows. The steps should be almost identical for Linux. Download the IDE installer from https://visualstudio.microsoft.com/vs/community. Run the installer by double clicking on the file:



This will cause the Visual Studio Installer start screen to appear (Figure A-1). Click the *Continue* button to get the Visual Studio Installer ready (Figure A-2).

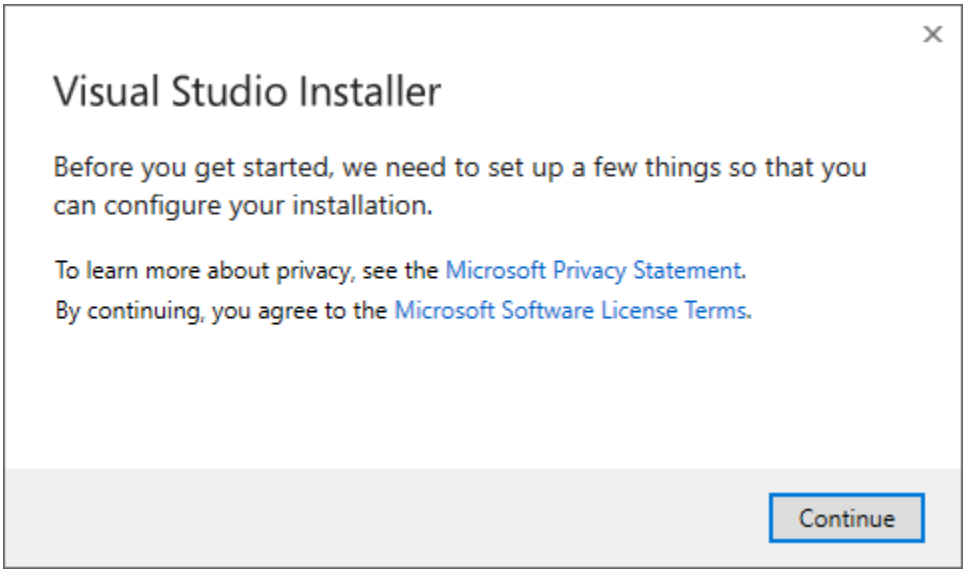


Figure A-1. Visual Studio Installer start screen

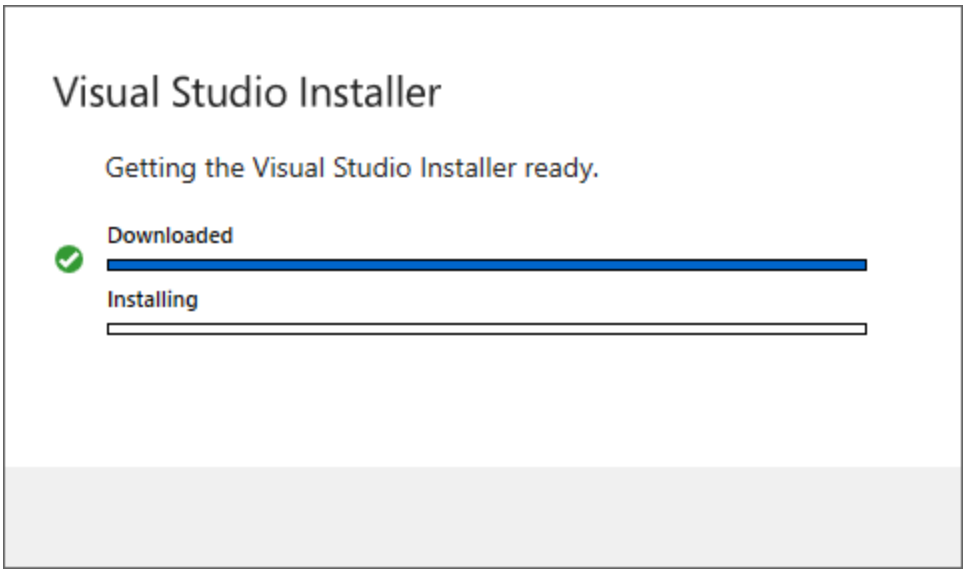


Figure A-2. Visual Studio Installer progress screen

A Visual Studio Installer options screen will appear (Figure A-3). Select *Desktop development with C++*. This will cause the right panel to become updated with the C++ installation details (Figure A-4).

NOTE: The screenshots below are for Visual Studio installer 17.2.5; the installer program is separate from the Visual Studio application and its appearance may vary by version.

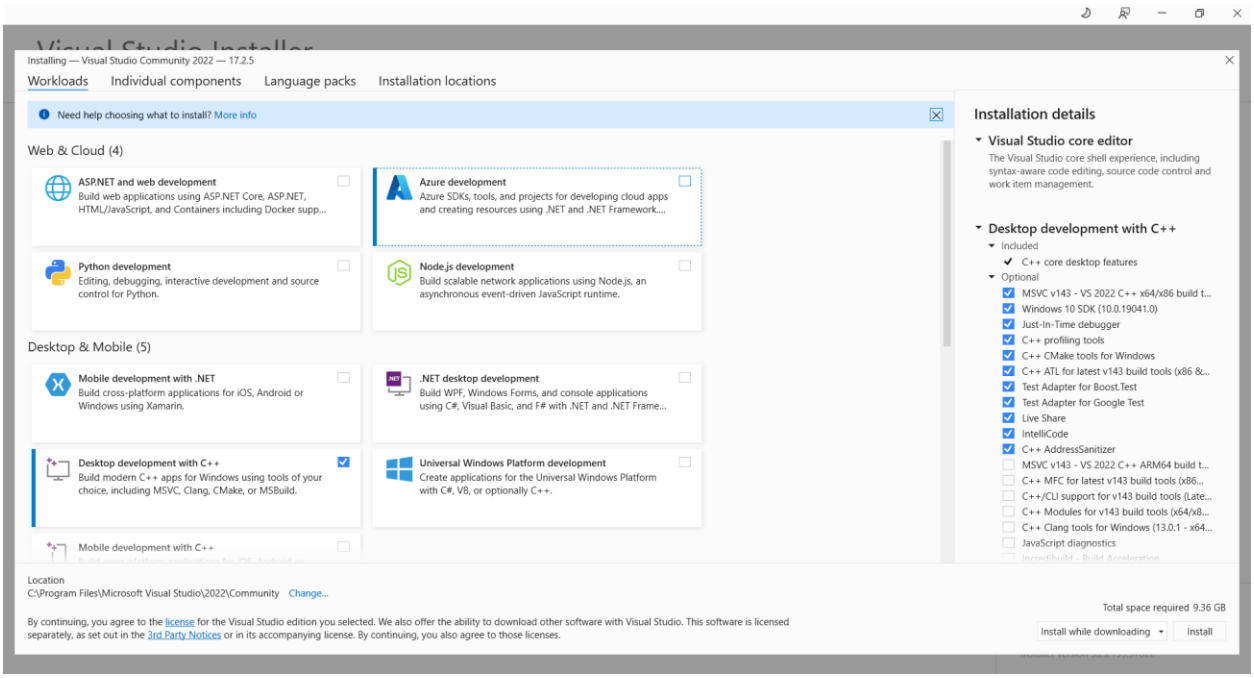


Figure A-3. Visual Studio Installer options screen

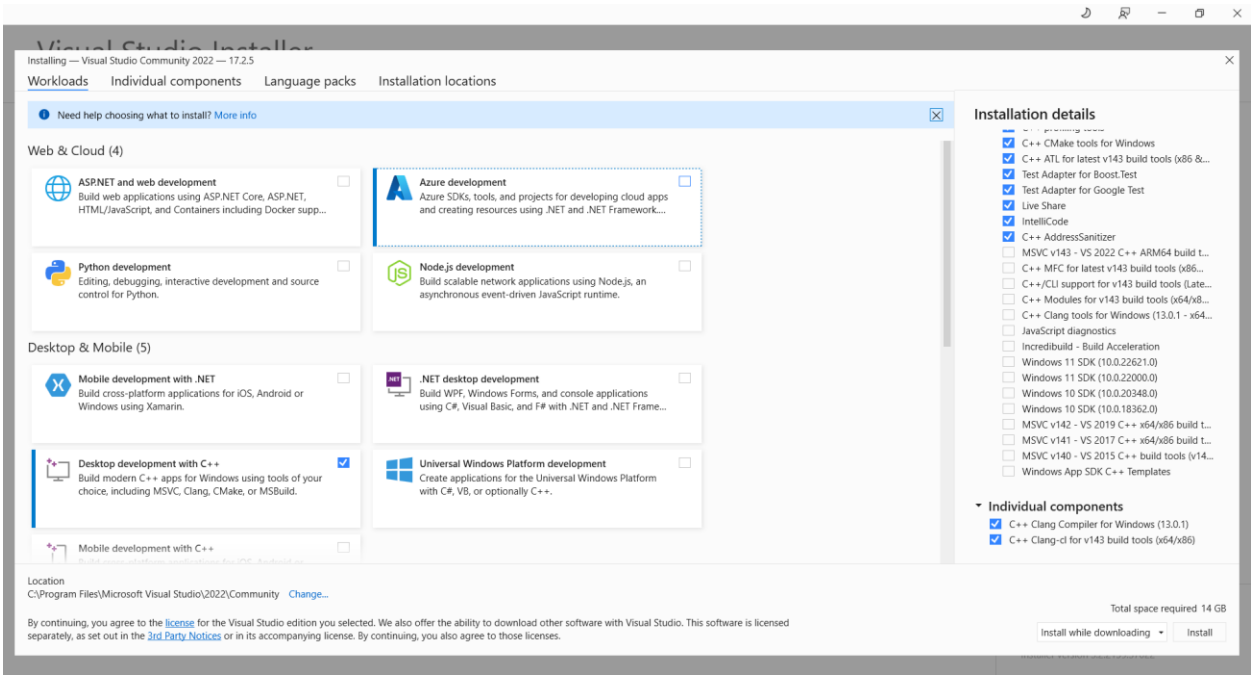


Figure A-4. Visual Studio Installer options screen for C++ installation details

Select the Individual Components tab and scroll down to check the boxes for: C++ Clang Compiler for Windows and Windows 10 SDK (Figure A-5). Then click the *Install* button.

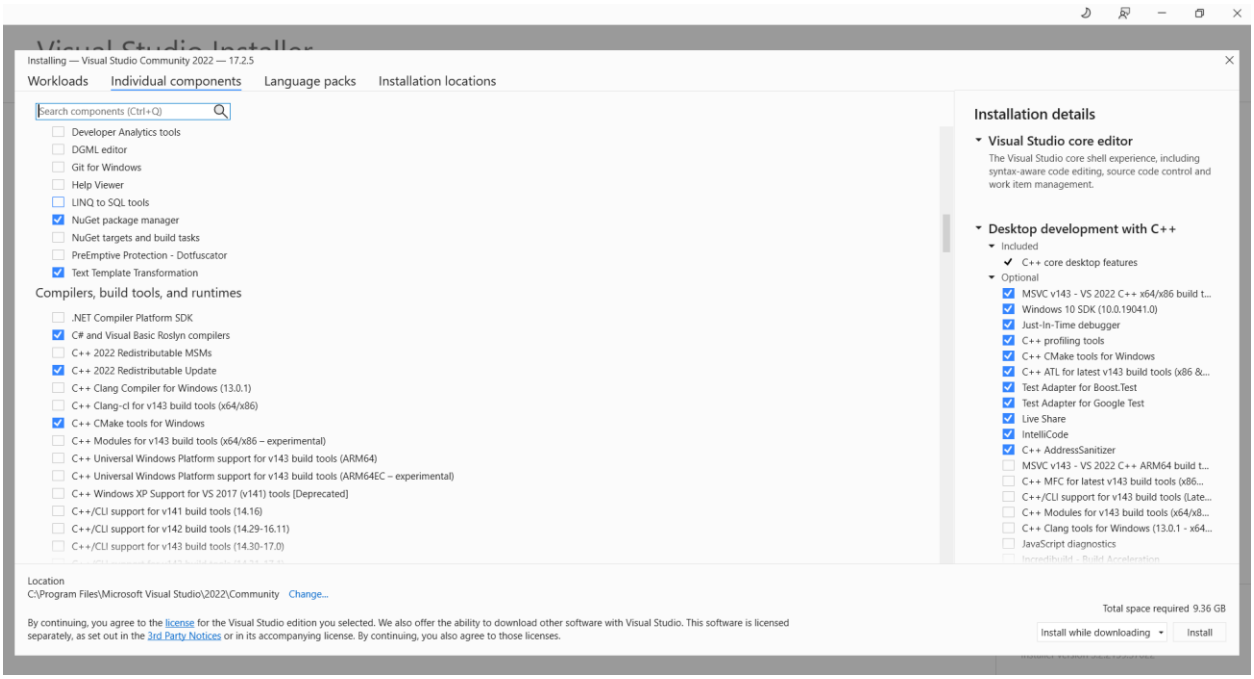


Figure A-5. Visual Studio Installer options screen for Individual Components

An installer progress window will appear showing all installed versions and progress of installing Visual Studio Community (Figure A-10). It takes a few minutes for the IDE to install. An announcement will appear upon completion (Figure A-7). Click the *OK* button to close the window.

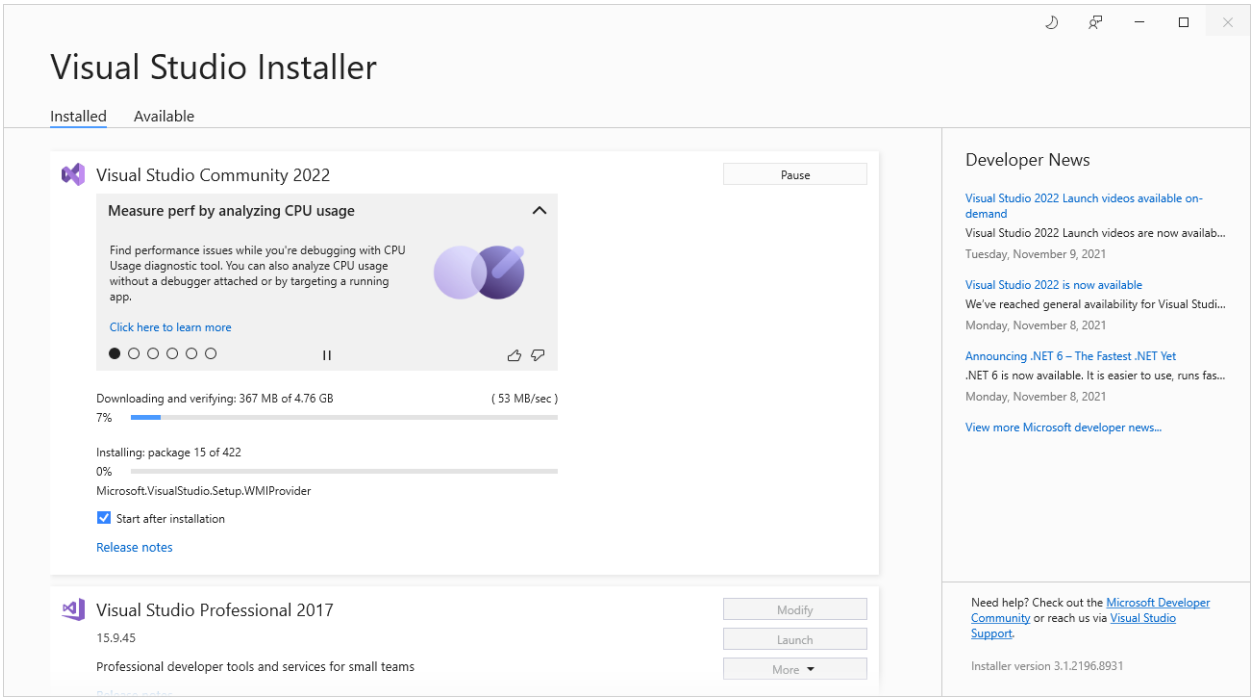


Figure A-6. Visual Studio Installer screen showing installed versions

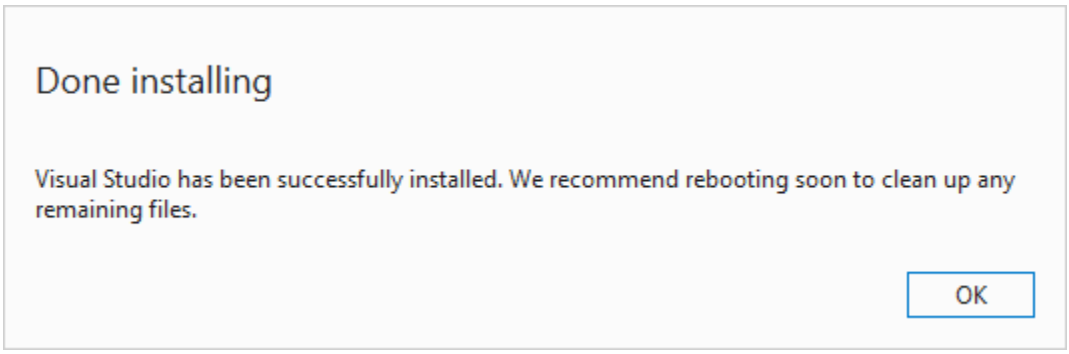


Figure A-7. Visual Studio installation completion screen

In the Visual Studio Community portion of the updated installer window (Figure A-8), click Launch to start the IDE.

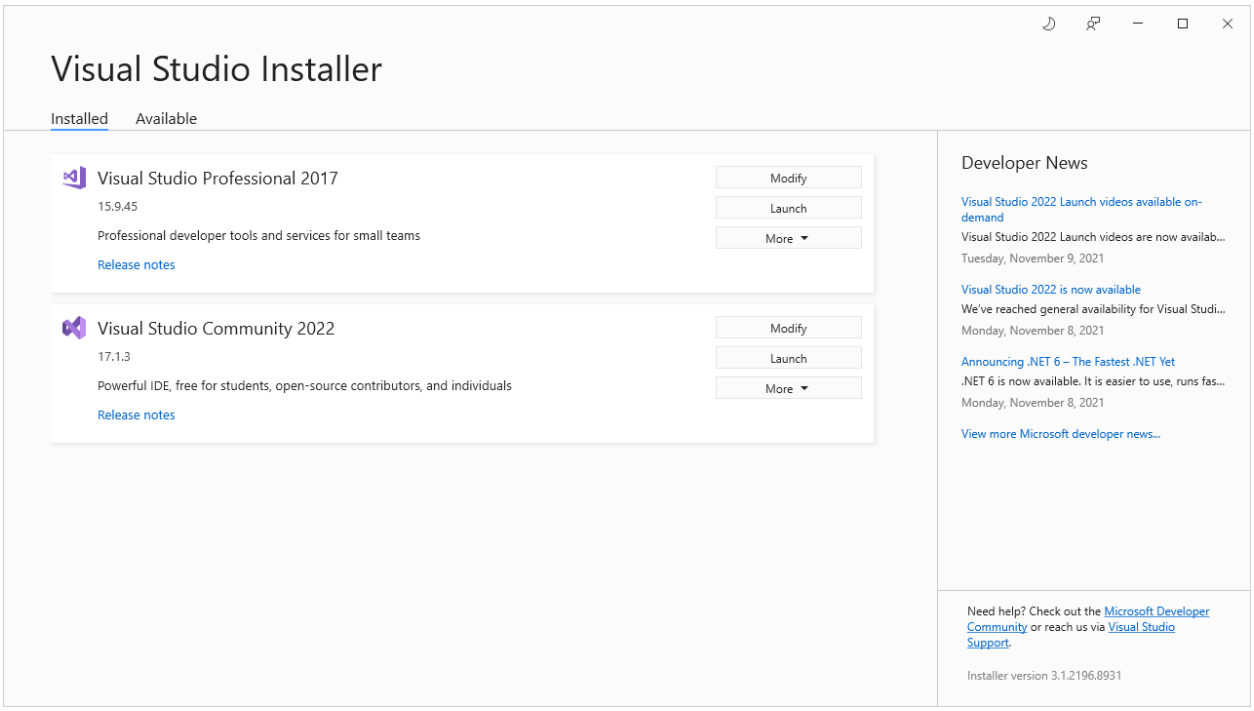


Figure A-8. Visual Studio Installer showing installed versions

Sign into Visual Studio (Figure A-9). This screen will get updated with a message it is setting up the IDE for first use (Figure A-10).

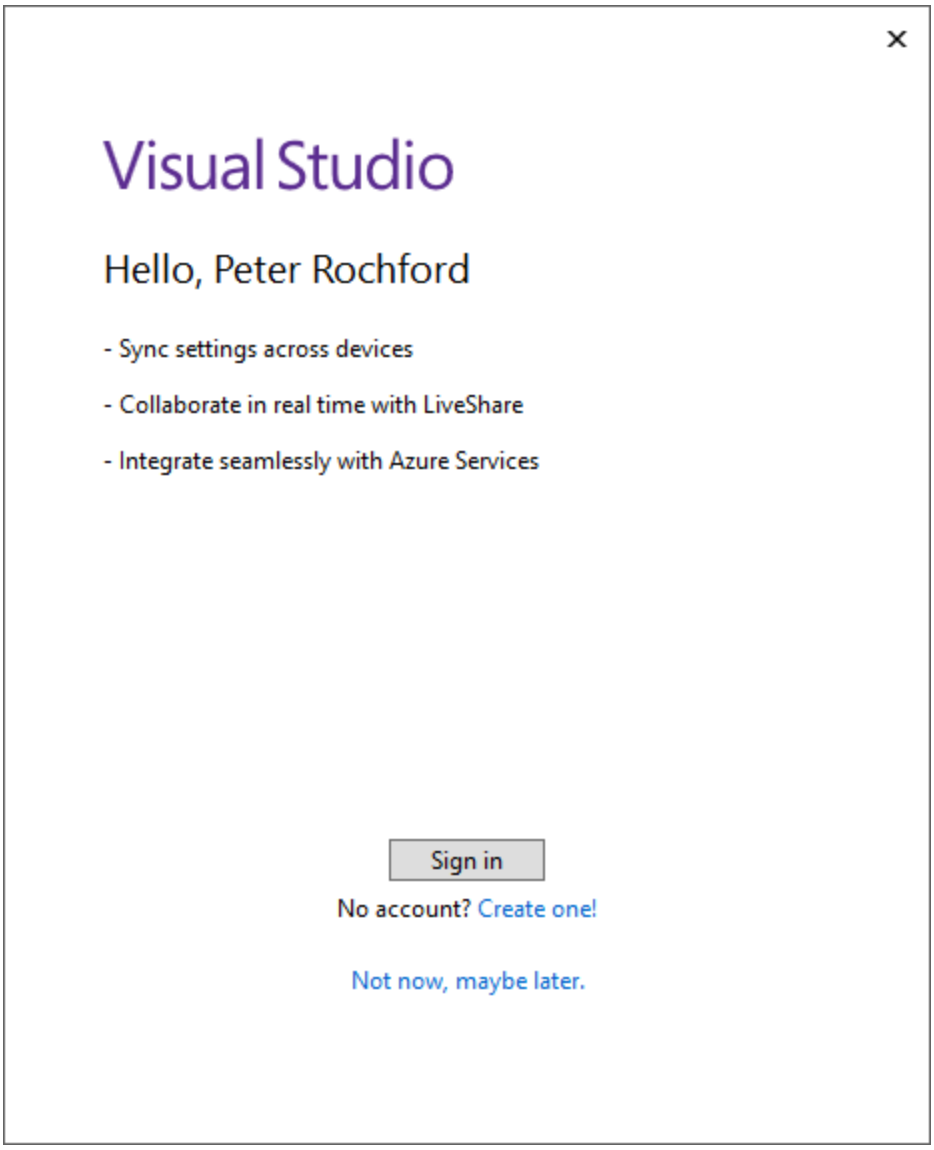


Figure A-9. Visual Studio sign in screen

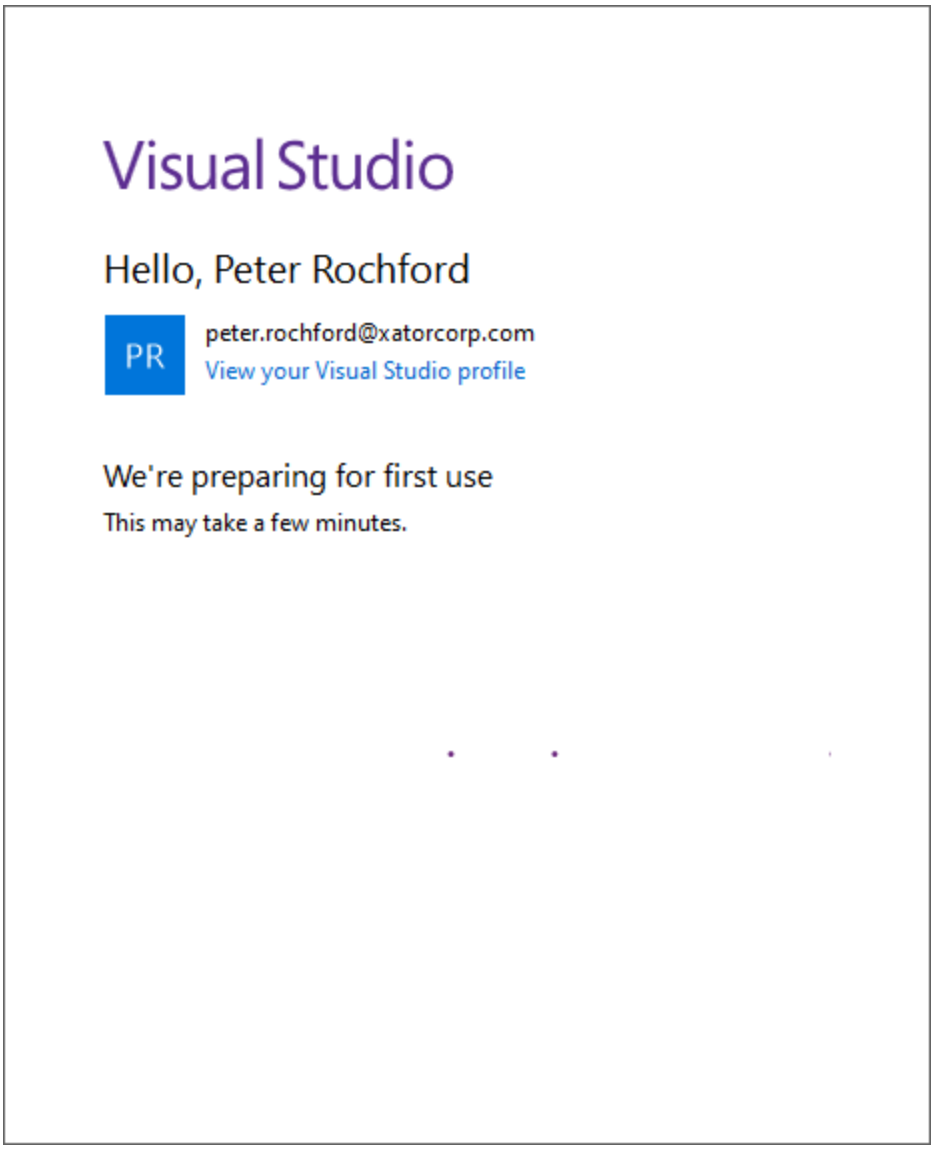


Figure A-10. Visual Studio first use progress screen

A Visual Studio 2022 startup window will appear (Figure A-10). Close the window. Reboot the computer to ensure the installation is updated in the OS.

A.1.2 Test Installation

Test the C++ compiler works with a HelloWorld program by creating a new project. Start the IDE by typing in the Windows start menu “Visual Studio 2022” and then enter. Once the IDE starts up, select *Create a new project* (Figure A-11).

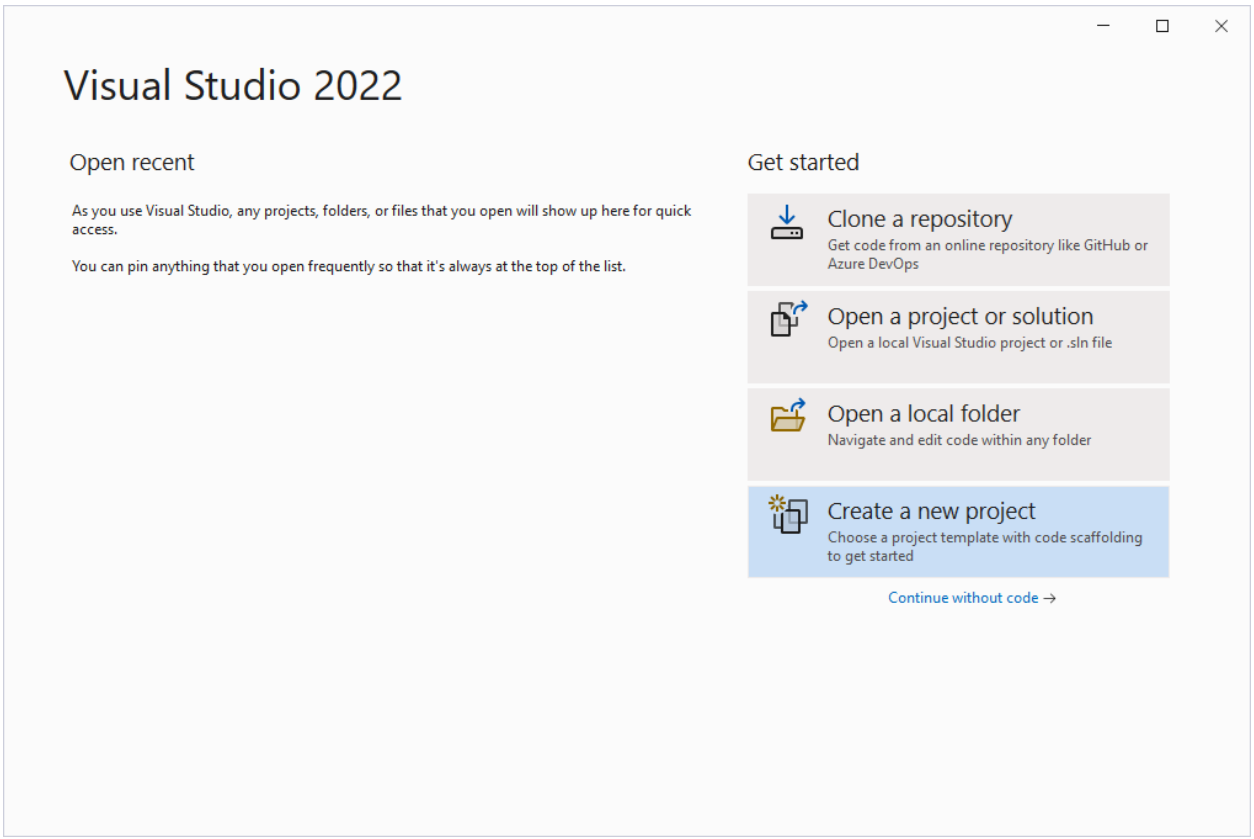


Figure A-11. Visual Studio start up screen

Choose an Empty Project for C++ (Figure A-12). Then click on *Next*.

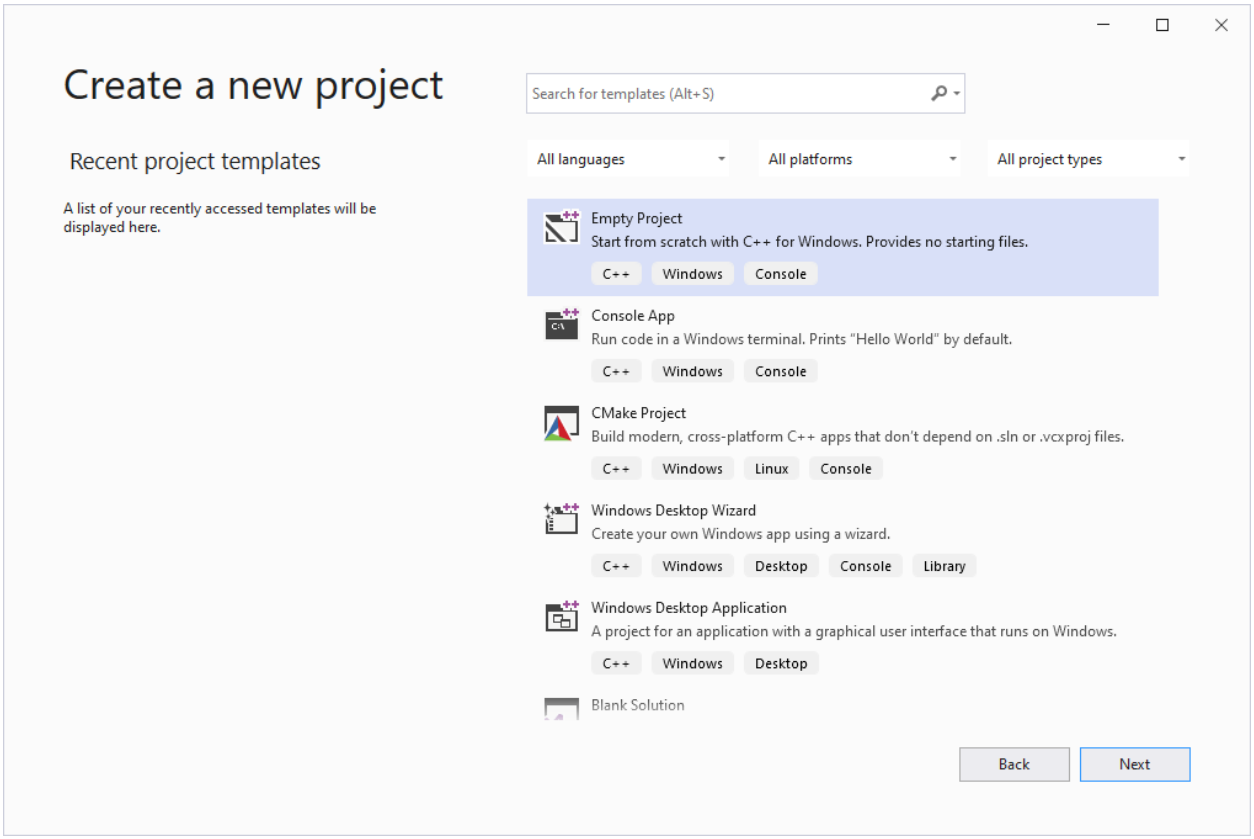


Figure A-12. Visual Studio create a new project screen

Enter HelloWorld for the project and solution name (Figure A-13). Accept the default location. Check the box for the solution and project to be in the same directory. Then click the *Create* button.

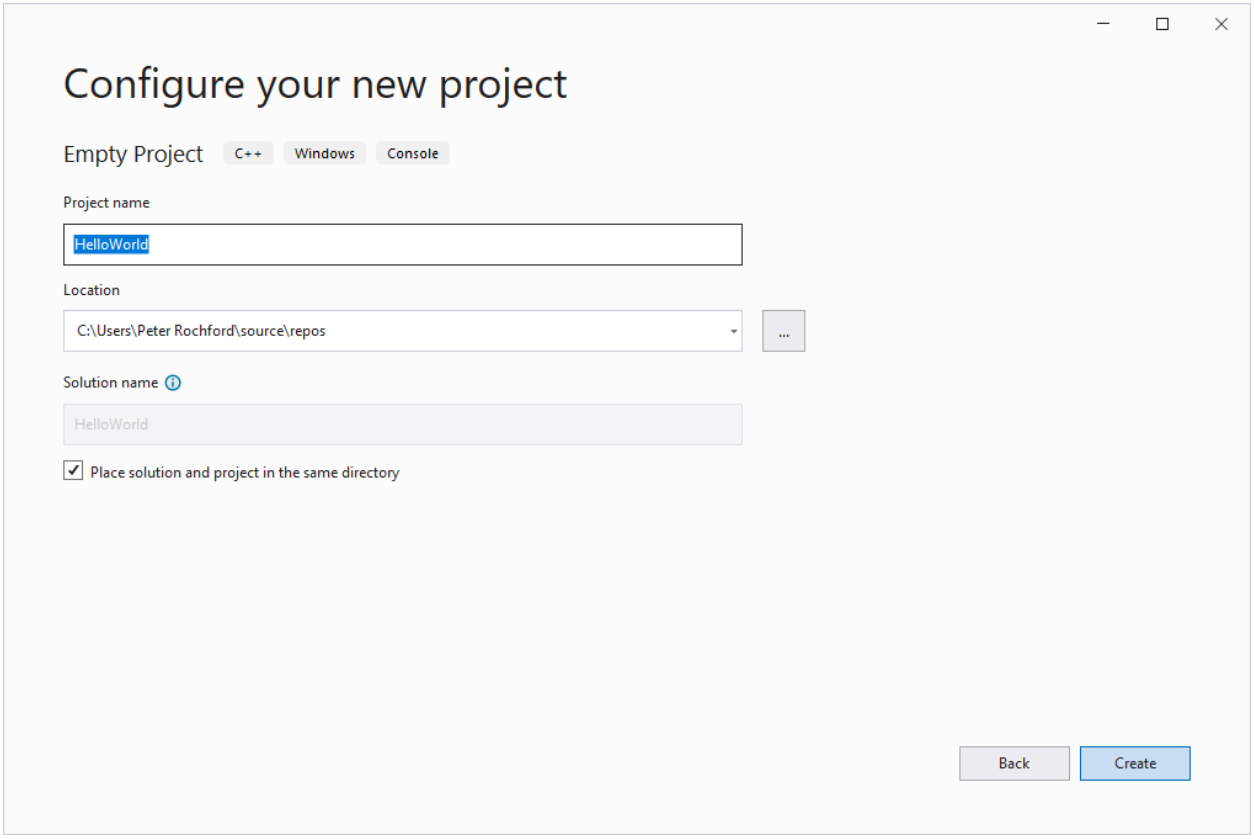


Figure A-13. HelloWorld project set up

The IDE will appear with the Solution Explorer window on the left by default (Figure A-14).

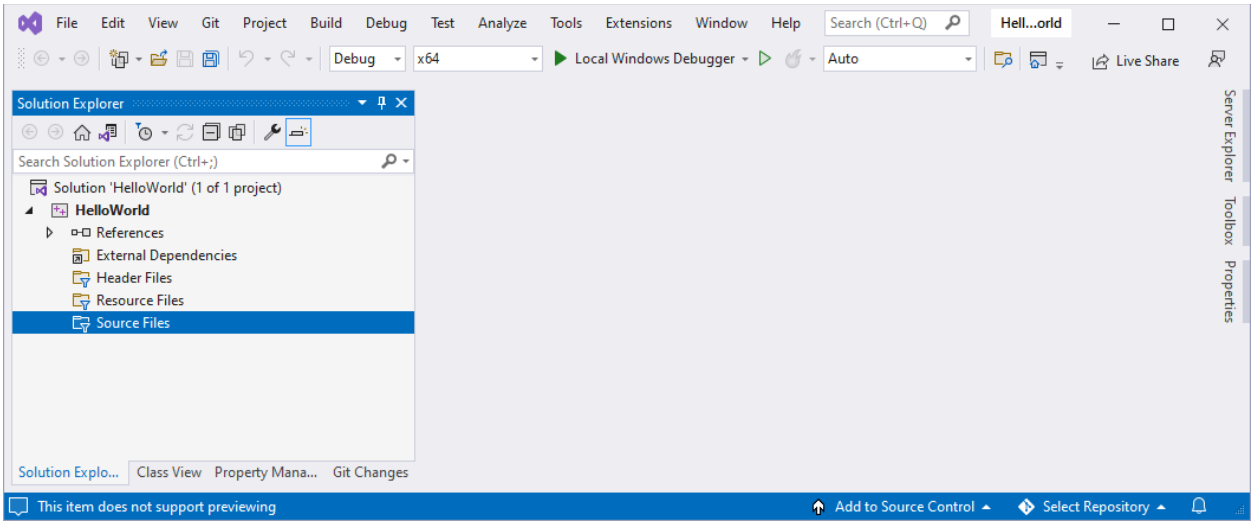


Figure A-14. Visual Studio showing HelloWorld project

Right click on *Source Files* and select *Add → New Item …* (Figure A-15).

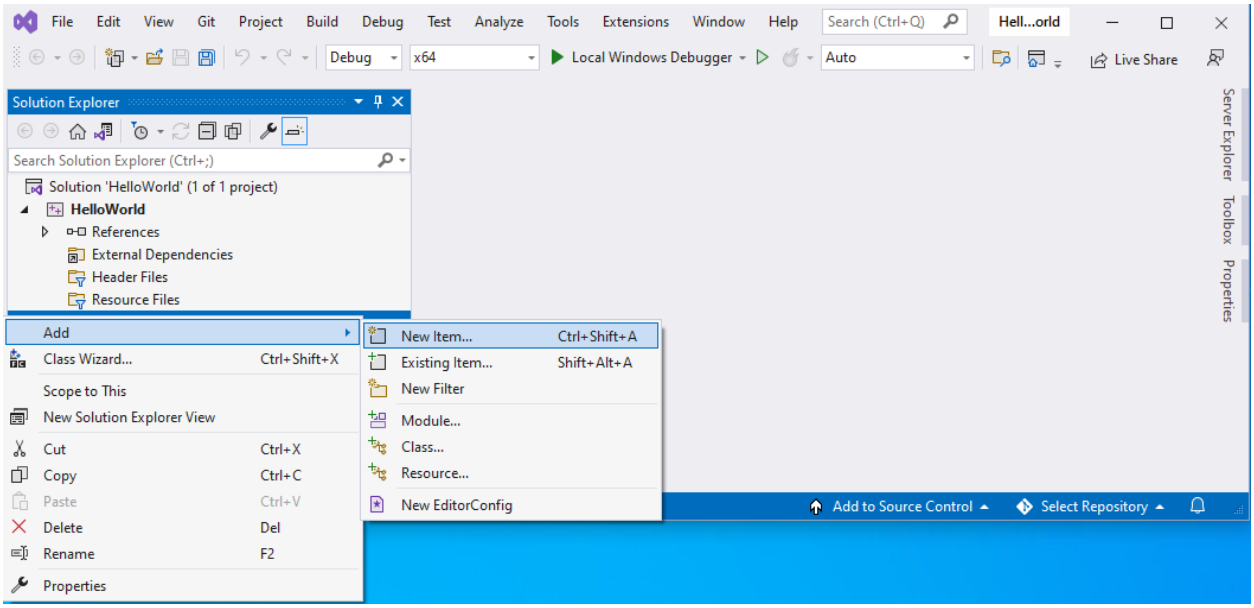


Figure A-15. Add new item to HelloWorld project

Select *C++ File (.cpp)* and then click on the *Add* button (Figure A-16).

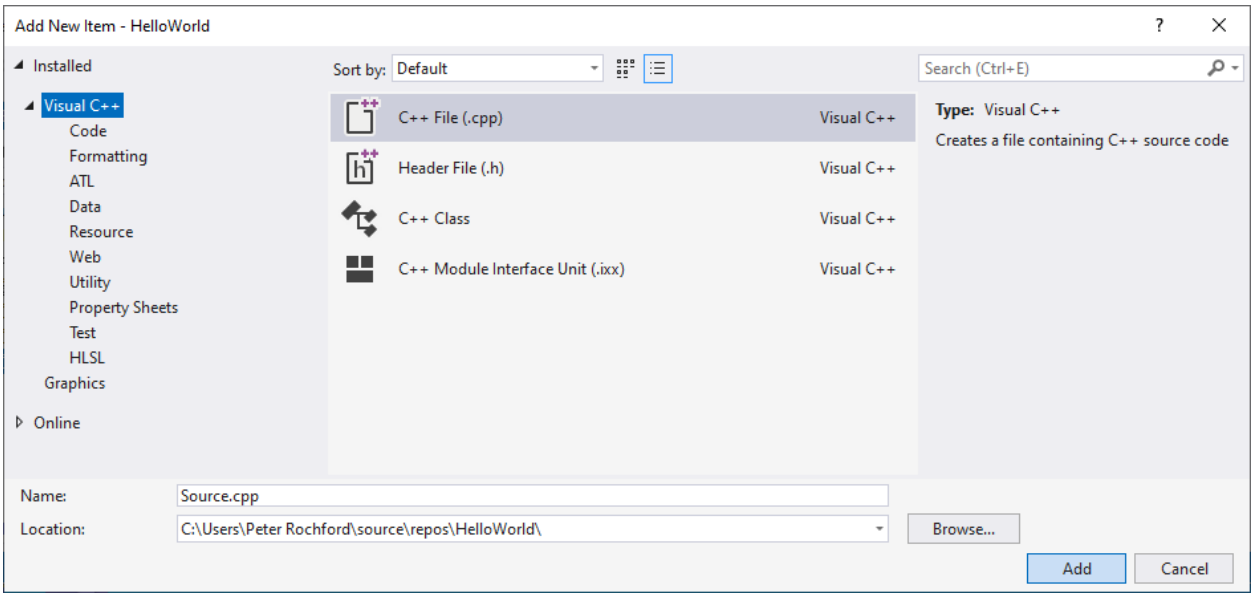


Figure A-16. Add C++ file to project

Right click on the Source.cpp file and rename to HelloWorld.cpp (Figure A-17).

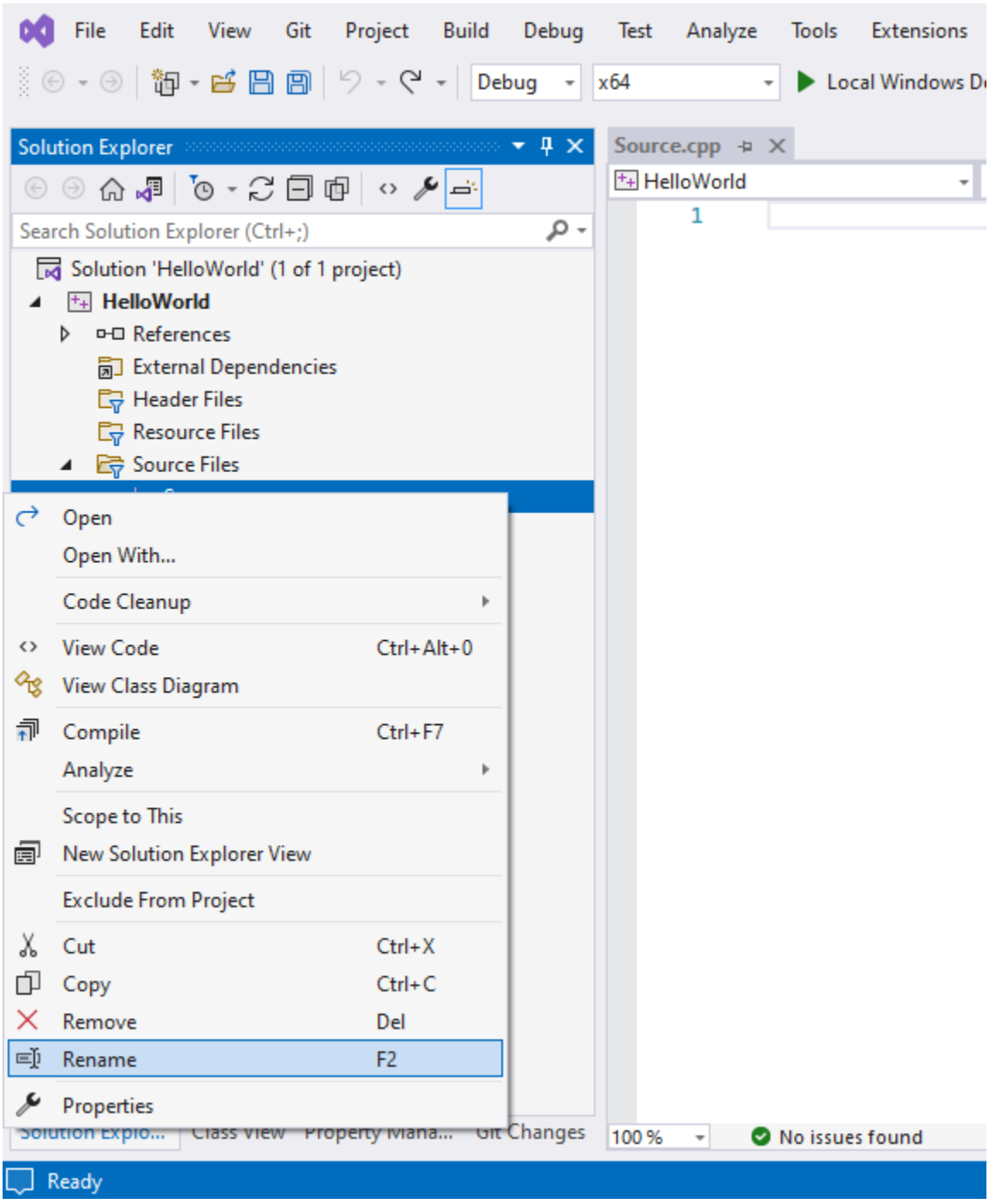


Figure A-17. Rename C++ source file

Type in a simple Hello World program (Figure A-18). Sample code is below.

```
//================================================================
========
```

```
// // Description : Hello World in C++, Ansi-style
//============================================================================
========
#include <iostream>
using namespace std;
int main() {
	cout << "Hello World!" << endl; // prints Hello World!
	return 0;
}
```

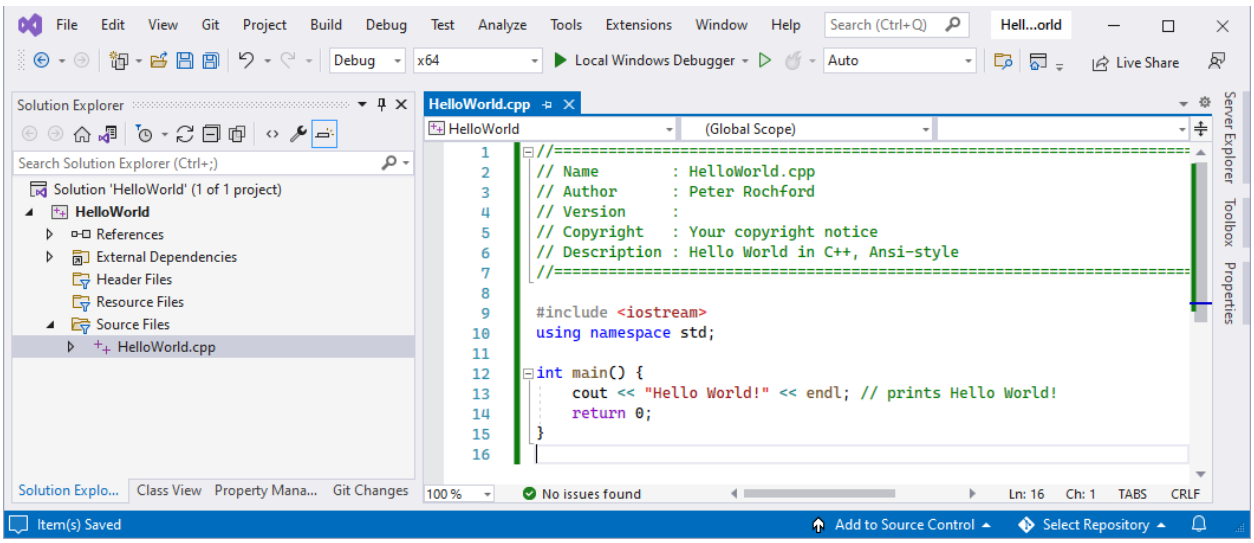


Figure A-18. HelloWorld program in Visual Studio

Save the file (Ctrl+s).

Build the solution (F7).

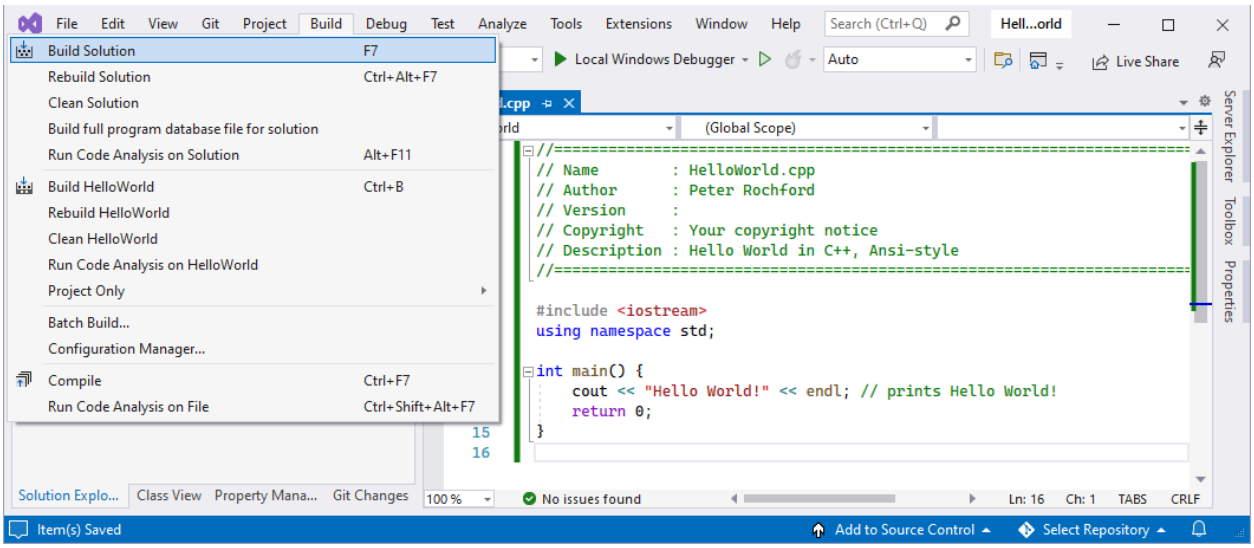


Figure A-19. Visual Studio Installer progress screen

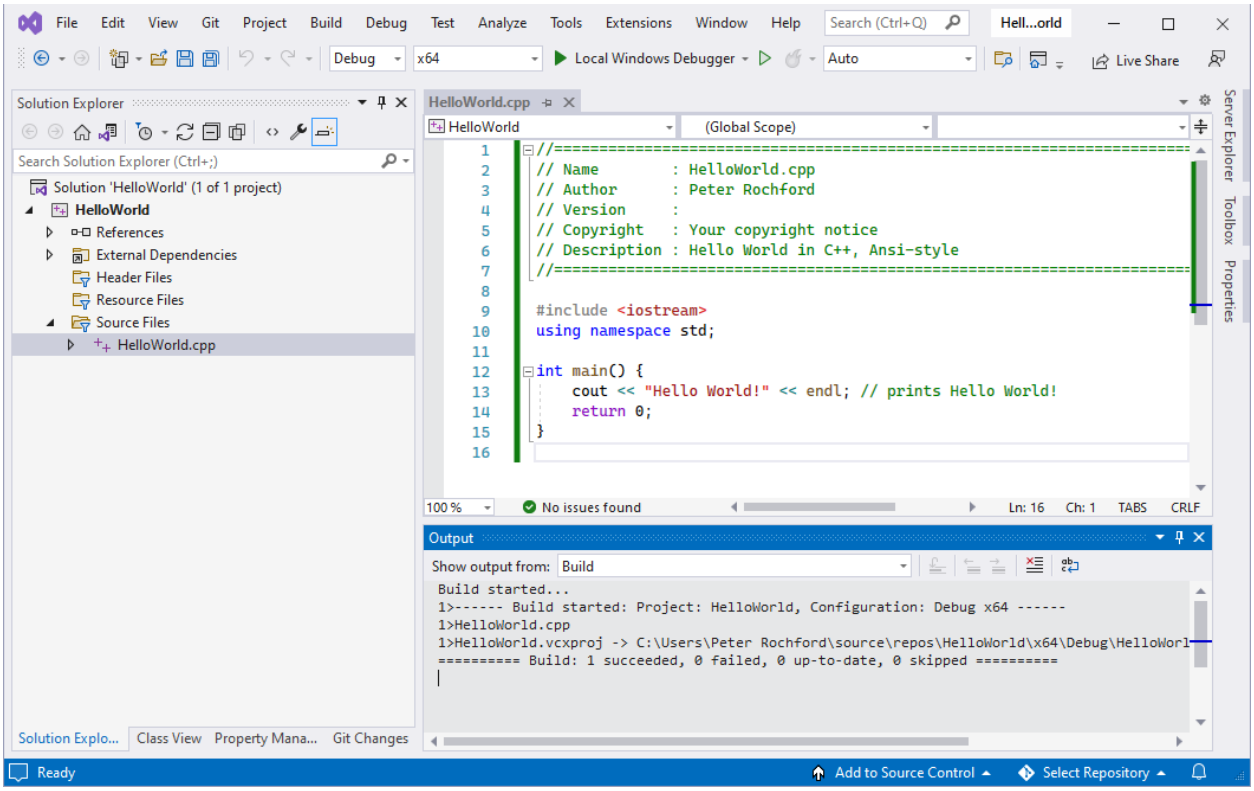


Figure A-20. Visual Studio Installer progress screen

Run the program by pressing Ctrl+F5 or clicking the green arrow in the menu bar (Figure A-211).

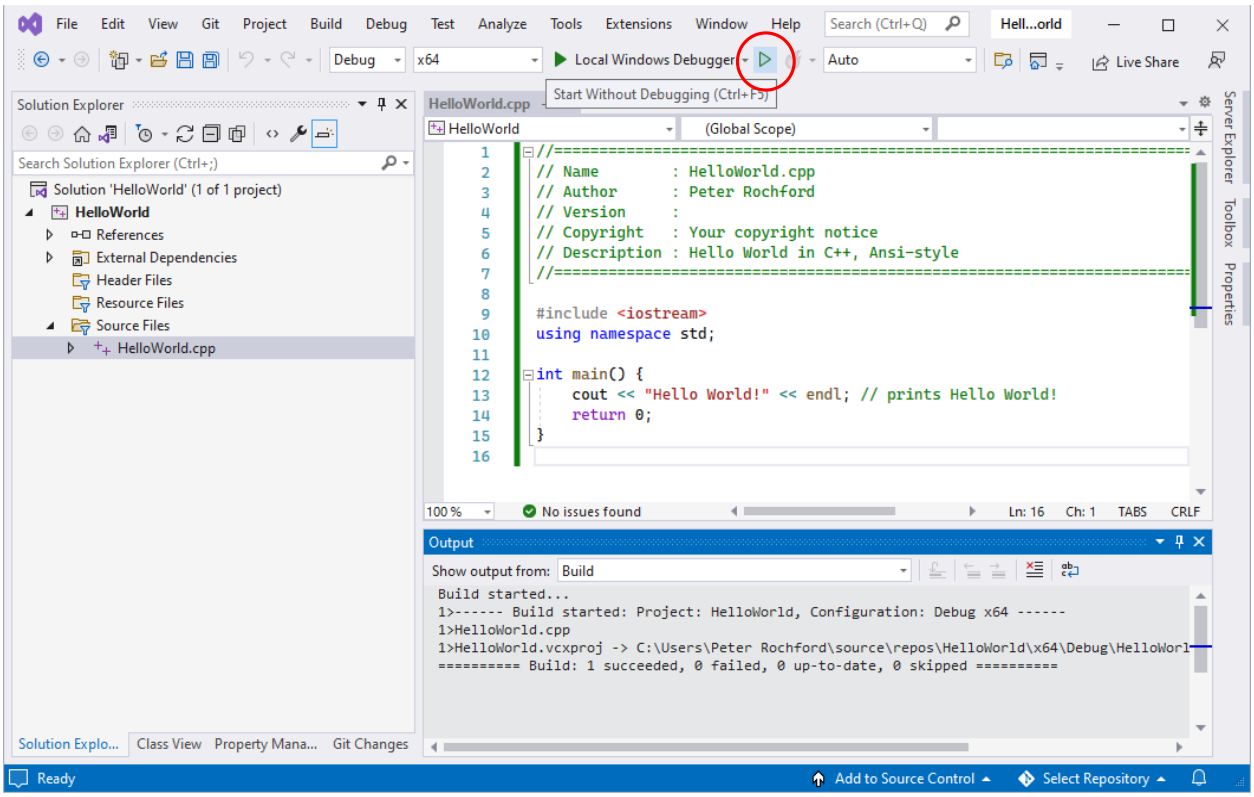


Figure A-21. Run button in Visual Studio

Confirm “Hello World!” message appears in a command window (Figure A-22).

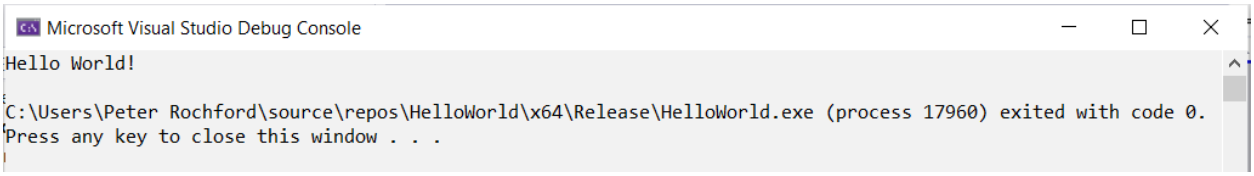


Figure A-22. Terminal window with Hello World! message