A BEGINNER'S GUIDE TO DEVELOPMENT IN OpenSees

Authors:
PRAVEEN KAMATH
LIMING JIANG

Project Supervisor:
ASIF USMANI

Contents

Disclaimer:

The document is written to guide the absolute beginners working on development of OpenSees. It includes download, install, run and debug OpenSees source code. It does not cover the working instructions for end users of the framework. It may / may not work on your installed copy of Microsoft Visual Studio for a number of reasons. The solutions mentioned in this document have been successfully tried and tested on ***Microsoft Visual Studio Professional 2010****. Email the authors at* ***praveen.kamath@hotmail.com*** *or* ***liming.jiang@ed.ac.uk*** *if you have any questions or comments or concerns. For further information on OpenSees Developers Group, log on to our* ***wiki page****:* `https://www.wiki.ed.ac.uk/display/opensees/UoE+OpenSees`.

Section 1

BRIEF OVERVIEW

THE ACRONYM OpenSEES - Open System for **E**arthquake **E**ngineering **S**imulation.

▶ OpenSees is a software framework for building finite element applications in structural and geotechnical systems.

▶ OpenSees developers group at the University of Edinburgh is working on Thermo-Mech. version of OpenSees to facilitate the analysis of Structures in Fire.

▶ The thermo-mechanical analysis codes are written using the Object Oriented Language, C++, along with additional commands for the scripting language Tcl/Tk.

▶ SIF Builder stands for Structures in Fire Builder, which is a project being developed by the research group to analyse the structures under real fire scenarios.

▶ OpenSees enables a user to run both traditional and parallel finite element applications for simulating the response of structural and geotechincal systems subjected to earthquakes and other extreme loading conditions.

Section 2

OPENSEES GOALS, FEATURES and MOTIVES

2.1 Goals:

- ▶ Open source finite element code development
- ▶ Education
- ▶ Interactive web-based communication of users and developers
- ▶ Provide supercomputing platform for users and developers through NEESHub's cloud computing resources

2.2 Features:

OpenSees source codes are available for free for further development. The interpreters provide flexibility to:

- ▶ Create pre and post processors for OpenSees Interpreters
- ▶ Create new modules like elements, materials, solvers, integrators for OpenSees interpreters
- ▶ Program and validate new modules with user specified benchmarks
- ▶ Enable a wider capability to existing framework to account for pragmatic structural engineering problems

2.3 Motive behind open source software development:

- ▶ Linus's Law: "given enough eyeballs all bugs are shallow"
- ▶ Free software attracts users
- ▶ New ideas can be explored and built upon if they are available to be scrutinized

- Many software developed in civil engineering in research institutions are lost when the graduate students leave
- Software developed in civil engineering research requires greater testing and flexibility

Section 3

GENESIS OF OPENSEES FRAMEWORK

OpenSees emerged out of ‘Object oriented finite element programming: frameworks for analysis, algorithm and parallel computing” (1997), a Ph.D thesis authored by Frank McKenna at the University of California at Berkeley. Ever since its inception, several modifications have been made to the framework, as a sign of incessant development though the original design has remained the same.

Section 4

PREREQUISITES FOR ASPIRING OPENSEES DEVELOPERS

- ▶ Working knowledge of C++: a statically typed, free-form, multi-paradignm, compiled, general-purpose programming language. It is regarded as a "middle-level" language, as it comprises a combination of both high-level and low-level language features
- ▶ A fairly well understanding of the Object-Oriented programming concepts
- ▶ Most commonly the structure and usage of classes, constructors, destructors, overloading functions and classes, class pointers, friendship and inheritance, polymorphism, virtual members, virtual classes, virtual functions, namespaces, memory management and other widely used terms in object oriented programming.

Section 5

ONLINE RESOURCES ON OPENSEES

OpenSees development has taken a new shape in the history of free civil engineering software with its powerful documentation and user-friendly instructive resources from **National Earthquake Engineering Centre (NEES)**. It enables users and developers from across the globe to witness the powerful software framework and utilize in their own way either using it to empower their own computational efforts or contribute to the evolving next generation framework in Earthquake Engineering.

5.1 Official Webpage

OpenSees offers extensive online resources for both users and developers. Following links summarises different features available on OpenSees web portals of both Berkeley and Edinburgh.

Official Berkeley OpenSees Homepage: **http://opensees.berkeley.edu/**
OpenSees homepage is typically designed with a very simple and user-friendly interface with several options to guide a user based on their area of interest. Three main tabs which spur interest in any aspiring OpenSees scholar are **User**, **Developer** and **Parallel**.
The vertical panel on the left of the webpage consists of the following.
OpenSees Wiki: Wikipedia supported free resource for opensees user and developer.
Message Board: An interactive Q&A online forum which powers numerous users and developers at various research and academic institutes around the world to post queries, which is the best alternative to learn and exchange views on OpenSees development.
User Doc: Provides access to most popular and advanced user manuals online.
Download: A protected link for registered users to download latest version of OpenSees: executable binary and Activestate Tcl/Tk.
Source Code: A user friendly, interactive version of source code browser powered by Web Subversion Repositories which enables user to browse a complete structured source codes in the OpenSees project.

Section 6

STRUCTURE OF OPENSEES

6.1 General

Before getting started with OpenSees development, it is imperative to have minimal knowledge of the project architecture. The program is structured by writing several hundreds of individual classes and their sub-classes. OpenSees application program interface (**API**) provides a vital reference to understanding the overall structure of the framework. OpenSees API consists of classes from the OpenSees framework, arranged in alphabetical order. Click on each class to open a **doxygen** generated document which has the following features.

- Pre-processor directives, header file (*.h) and source files (*.cpp) for the class
- Inheritance diagram for the class
- List of all the members in the class
- List of public member functions
- Constructor and destructor documentation
- Member function documentation (Implemented in - - - Referenced by)

6.2 Additional tabs in the API

Namespaces - Commonly used namespaces.
Files - List of all the files in the OpenSees project and their path.
Directories - Directory hierarchy.
Class List - List of all the classes arranged automatically.
Class Hierarchy - Sorted inheritance list of classes.
Class Members - List of all class members with links of their classes.

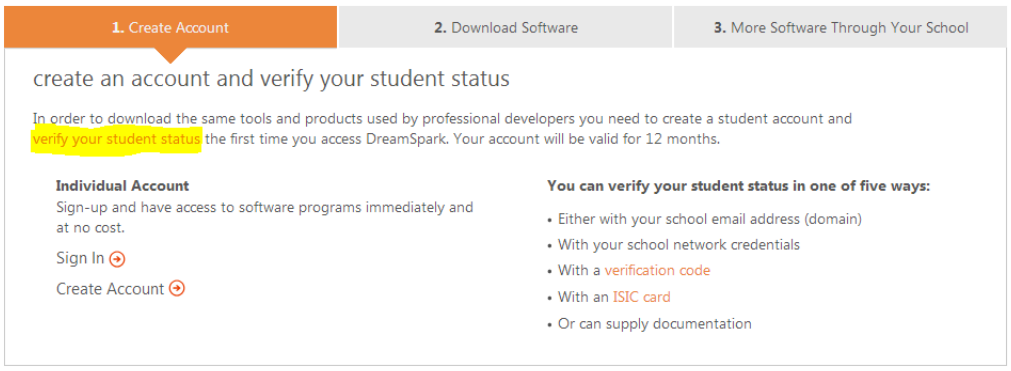
Section 7

DOWNLOAD AND INSTALL MICROSOFT VISUAL STUDIO AND TCL/TK

The source codes are written in the Object oriented language, C++. The source codes for OpenSees were developed using Microsoft Visual Studio 2005. The source codes can also be compiled using higher versions Microsoft Visual Studio 2008 2010, 2012 and 2013. It is advisable to stick to version **2008 / 2010** to avoid compatibility issues during the conversion of the solution (*.sln) files. The following steps show the instructions to download and install the 'absolute essentials' for running OpenSees on a Windows Platform, Microsoft Visual Studio and Tcl/Tk.
If you are a student / researcher / faculty of a University, all software from Mucrosoft may be downloaded from **Microsoft Dreamspark**.

7.1 Microsoft Visual Studio

Step 1. Before signing up for an individual account, verify your student status using your school email Address.
Note: It is encouraged to merge the dreamspark account with Microsoft Outlook account (if you already have one)



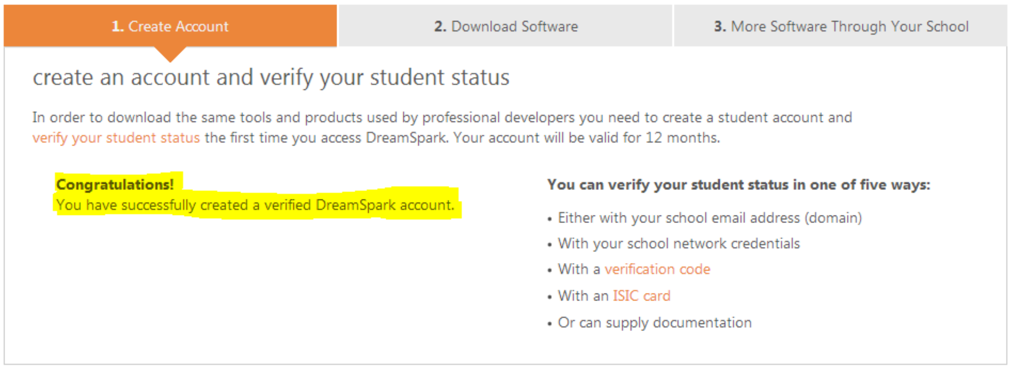
create an account and verify your student status

In order to download the same tools and products used by professional developers you need to create a student account and verify your student status the first time you access DreamSpark. Your account will be valid for 12 months.

Individual Account
Sign-up and have access to software programs immediately and at no cost.
Sign In
Create Account

You can verify your student status in one of five ways:
- Either with your school email address (domain)
- With your school network credentials
- With a verification code
- With an ISIC card
- Or can supply documentation

Step 2. Upon completing the verification, you should receive the following greeting.



1. Create Account | **2.** Download Software | **3.** More Software Through Your School

create an account and verify your student status

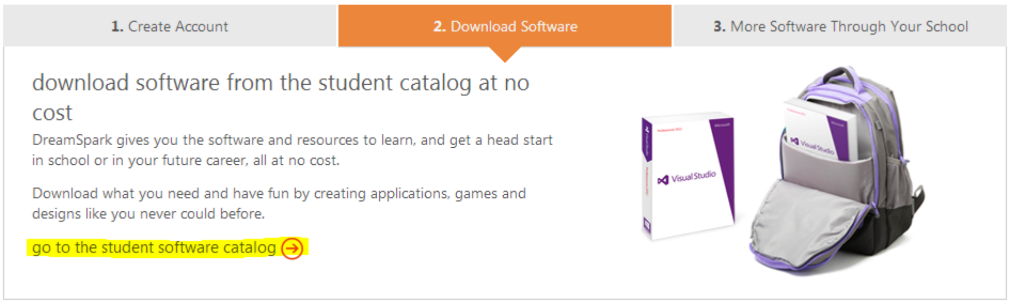
In order to download the same tools and products used by professional developers you need to create a student account and verify your student status the first time you access DreamSpark. Your account will be valid for 12 months.

Congratulations!
You have successfully created a verified DreamSpark account.

You can verify your student status in one of five ways:

- Either with your school email address (domain)
- With your school network credentials
- With a verification code
- With an ISIC card
- Or can supply documentation

Step 3. Click on **Download** Software tab and then click on go to the **student software catalog**



1. Create Account | **2.** Download Software | **3.** More Software Through Your School

download software from the student catalog at no cost

DreamSpark gives you the software and resources to learn, and get a head start in school or in your future career, all at no cost.

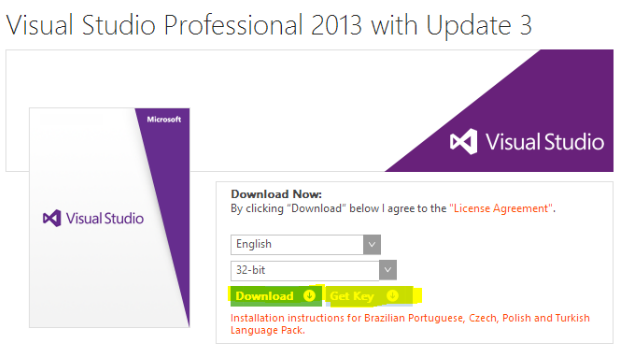
Download what you need and have fun by creating applications, games and designs like you never could before.

go to the student software catalog

Step 4. Browse and select for the desired version of **Microsoft Visual Studio Professional** under **Developer & Designer Tools**.

Note: Choose ver 2008 / ver 2010.

Step 5. Click on Get Key button and note down the 16 digit product key before the download



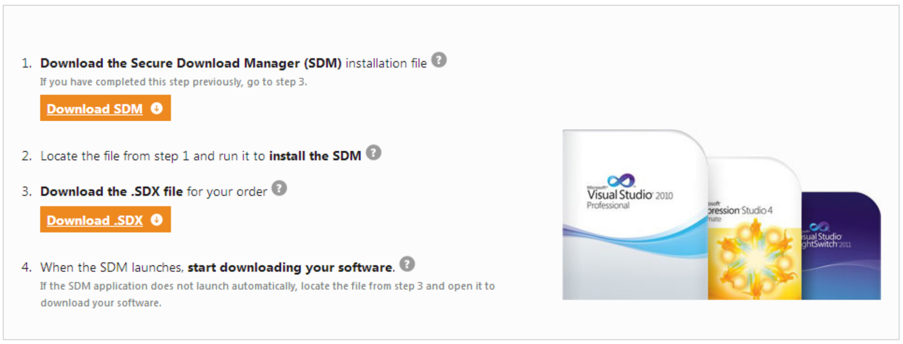
Visual Studio Professional 2013 with Update 3

Download Now:
By clicking "Download" below I agree to the "License Agreement".

English
32-bit
Download | Get Key

Installation instructions for Brazilian Portuguese, Czech, Polish and Turkish Language Pack.

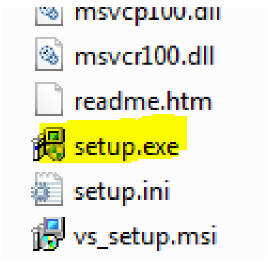
Step 6. Click on **Download** and follow the onscreen instructions to download the software.


1. **Download the Secure Download Manager (SDM)** installation file
   If you have completed this step previously, go to step 3.
   Download SDM
2. Locate the file from step 1 and run it to **install the SDM**
3. **Download the .SDX file** for your order
   Download .SDX
4. When the SDM launches, **start downloading your software.**
   If the SDM application does not launch automatically, locate the file from step 3 and open it to download your software.

An iso image (*.iso) of Microsoft Visual Studio will be downloaded to your computer

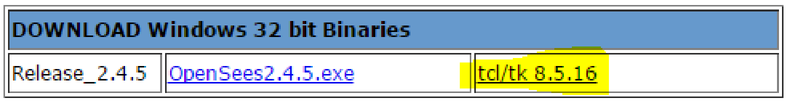
Step 7. The file can be opened using a virtual disk application such as **WinCDEmu** (***Download WinCDEmu here***)

Step 8. Open the WinCDEmu mounted virtual drive **(V:)** from My Computer and run **setup.exe** to Install Microsoft Visual Studio Professional.
Note: Keep complete install as the default option.


msvcp100.dll
msvcr100.dll
readme.htm
setup.exe
setup.ini
vs_setup.msi

7.2 Tickle ToolKit (Tcl/Tk)

Step 1. Download tcl/tk X.X.XX from the **OpenSees Berkeley** portal



DOWNLOAD Windows 32 bit Binaries		
Release_2.4.5	OpenSees2.4.5.exe	tcl/tk 8.5.16

Step 2. Double click on the downloaded version of tcl/tk
Note: In most of the windows 7 PCs, right click on the file and click Run as Administrator

Step 3. Select **Next** (at bottom right) $\longrightarrow$ I accept the license agreement and select the Tcl folder for installation.
Note: Make sure you Install Tcl in `C:\ProgramFiles\Tcl` *Folder. This folder does not exist by default and hence be created. Also, by default Tcl tends to install in* `C:\Tcl`*. This path needs to be changed to* `C:\ProgramFiles\Tcl`*.*

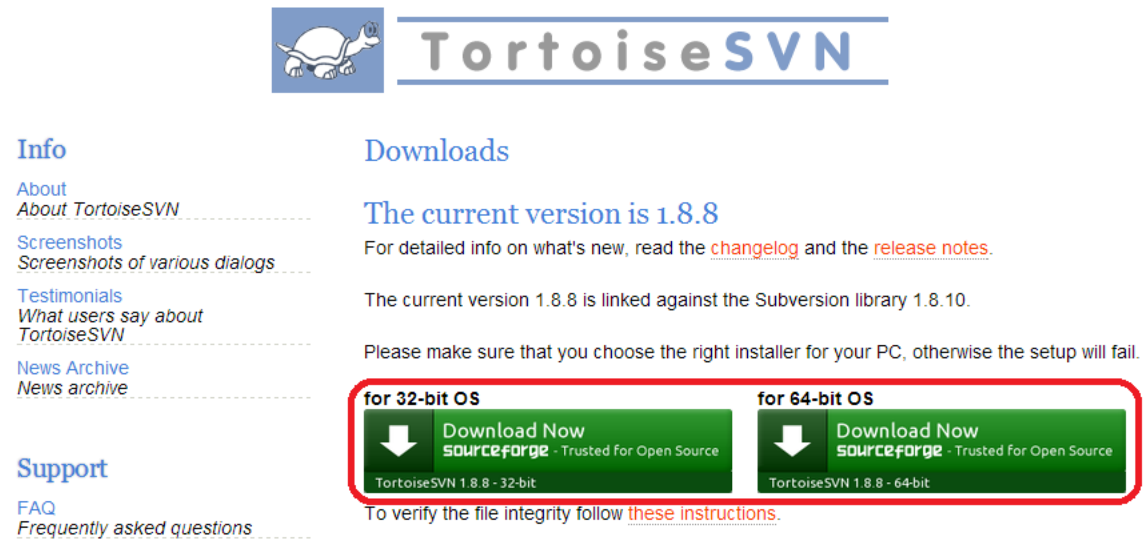
Section 8

DOWNLOADING OPENSEES SOURCECODE USING TortoiseSVN

OpenSees source codes are available for download from a revision and version control system developed by **Apache Subversion (SVN)**. To download any package from a version control system, a subversion client is needed. Following steps will guide you to download the OpenSees source codes using TortoiseSVN client.

8.1 Downloading TortoiseSVN

Step 1. Download the version control / source control software TortoiseSVN, using the link given below *(based on Apache^{TM} Subversion (SVN)®)*, depending on whether your version of windows is 32 bit / 64 bit.
http://tortoisesvn.net/downloads.html



Info

About
About TortoiseSVN

Screenshots
Screenshots of various dialogs

Testimonials
What users say about TortoiseSVN

News Archive
News archive

Support

FAQ
Frequently asked questions

Downloads

The current version is 1.8.8

For detailed info on what's new, read the changelog and the release notes.

The current version 1.8.8 is linked against the Subversion library 1.8.10.

Please make sure that you choose the right installer for your PC, otherwise the setup will fail.

TortoiseSVN
for 32-bit OS
Download Now
sourceforge - Trusted for Open Source
TortoiseSVN 1.8.8 - 32-bit
for 64-bit OS
Download Now
sourceforge - Trusted for Open Source
TortoiseSVN 1.8.8 - 64-bit

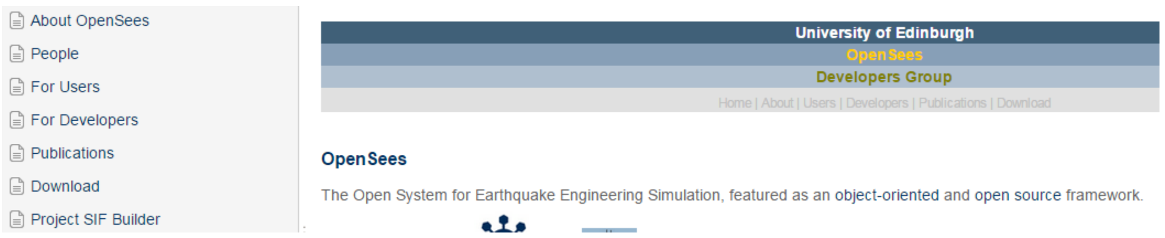
To verify the file integrity follow these instructions.

Step 2. Run the downloaded file and install the subversion package by following onscreen instructions.**TortoiseSVN**^{TM}

8.2 Downloading the OpenSees Source Code

Step 1. Click the following link or Copy and Paste it on your browser
https://www.wiki.ed.ac.uk/display/opensees/UoE+OpenSees

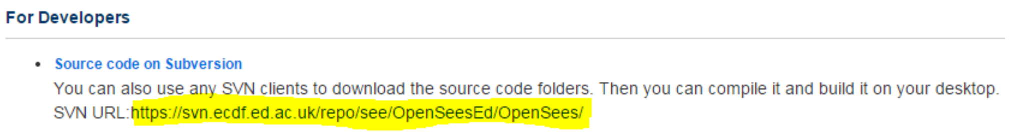
Step 2. Click on **Download** under SPACE SHORTCUTS located on the left hand panel as shown or on the bottom of the page.



OpenSees

The Open System for Earthquake Engineering Simulation, featured as an object-oriented and open source framework.

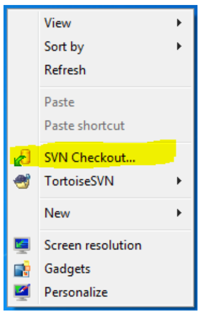
Step 3. Copy highlighted part of SVN link found at the bottom of the page under the section FOR DEVELOPERS



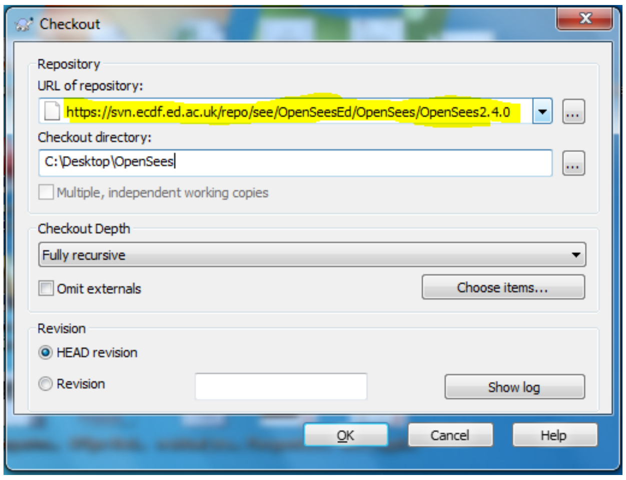
For Developers

- Source code on Subversion
  You can also use any SVN clients to download the source code folders. Then you can compile it and build it on your desktop.
  SVN URL:https://svn.ecdf.ed.ac.uk/repo/see/OpenSeesEd/OpenSees/

Step 4. Go to the folder where you want the source codes to be downloaded and **Right Click anywhere on the screen and select SVN Checkout**



Step 5. In the checkout window, enter the following URL under the **URL of Repository** field. Choose the checkout depth Fully Recursive.
https://svn.ecdf.ed.ac.uk/repo/see/OpenSeesEd/OpenSees/



URL of repository: https://svn.ecdf.ed.ac.uk/repo/see/OpenSeesEd/OpenSees/OpenSees2.4.0

Checkout directory: C:\Desktop\OpenSees

Multiple, independent working copies

Fully recursive

Omit externals

HEAD revision

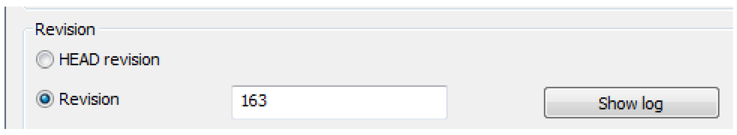
Revision

Step 6. Wait until the source code downloads and the checkout action shows **Completed!**

Note:

**Choose your own checkout directory*

You may checkout a particular revision by entering its number under* ***Revision*

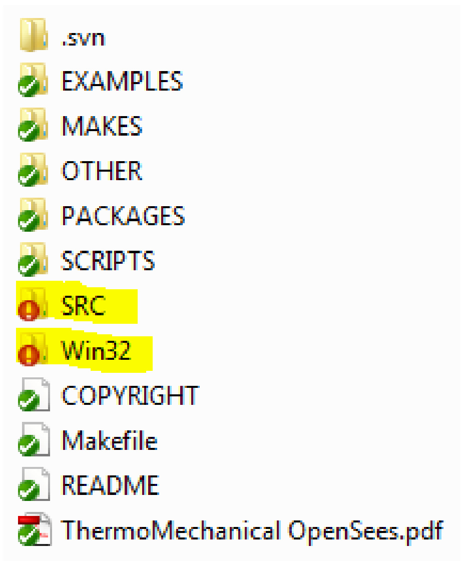


Revision
HEAD revision
[✓ Revision] 163

Section 9

WORKING WITH OPENSEES SOURCE CODES

After downloading the source codes using the subversion as explained in section 7.2, open the folder **OpenSees2.4.0**. It contains the following folders.



.svn
EXAMPLES
MAKES
OTHER
PACKAGES
SCRIPTS
SRC
Win32
COPYRIGHT
Makefile
README
ThermoMechanical OpenSees.pdf

The two main folders of interest for OpenSees development are **SRC** and **Win32**. The SRC folder contains the source scripts and the files in win32 control the source codes in SRC. The source files (*.cpp) and the header files (*.h) for individual projects can be found in the sub folders of SRC.

Before Building the Solution, DO THESE

- Check the **Configuration Manager**
  Click on ***Build*** in the menu bar and select the option 'Configuration Manager' to open it in a separate window.
  Ensure that the projects '***OpenSeesTk***', '***HTMain***' and '***quickmain***' are unchecked. If one of these is checked, you may encounter errors.

- Set Enable Incremental Linking to '**NO**'
  Right click on the project (startup project: eg., OpenSees) $\longrightarrow$ *Configuration properties* $\longrightarrow$

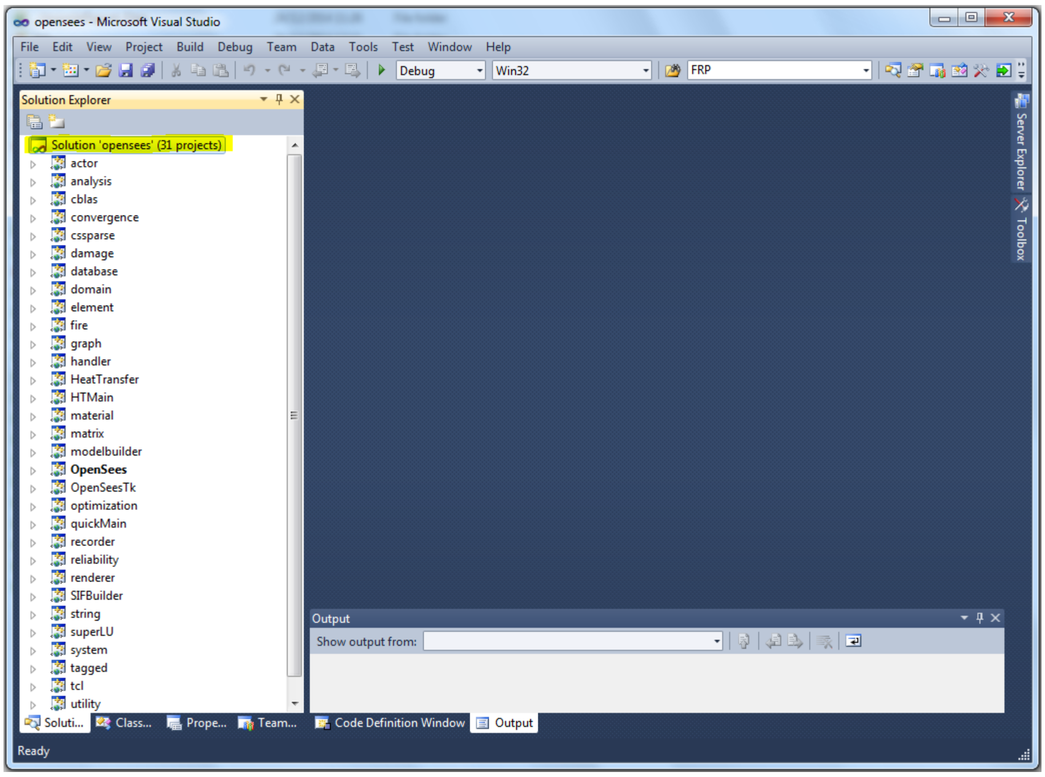
Linker ⟶ *General* ⟶ *Enable Incremental Linking* ⟶ *"NO (/INCREMENTAL:NO)"*

- Set **OpenSees** as startup project *(if not already set)*
  Right click on the project ***OpenSees*** in the solution explorer and select the option '*Set as StartUp Project*'. OpenSees now turns bold, which indicates that it has been set as a startup project.

Following steps show the instruction for compiling OpenSees.

Step 1. Open the folder containing the OpenSees project solution file
`OpenSees2.4.0/Win32/openSees.sln`

Step 2. Double click / run the file opensees.sln to open the project.
Note: For VS versions higher than 2008, the project files needs to be converted into new format. Follow the onscreen instructions and wait until the project files load and appear in the solutions explorer. This may take a few minutes. Also, choose to create a backup of the old solution when prompted. Also, if you see a class view instead of solution explorer, check FAQ

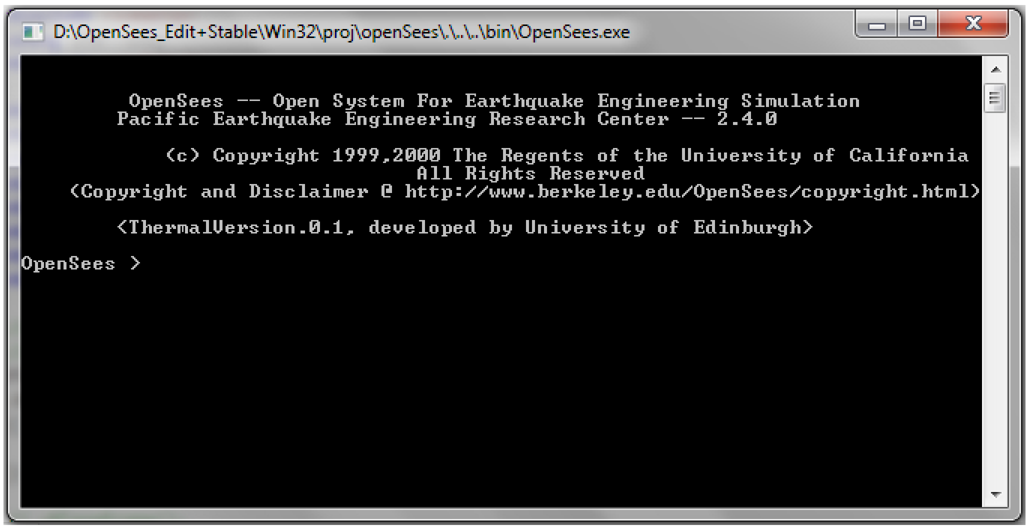


Step 3. Wait until the project loads and Ready appears on the bottom left of the screen. All the projects appear in the solution explorer to the left of the VS screen.
Note: Check if the main project OpenSees is boldfaced. If not, right click on the project and select the option Set as StartUp Project.
Also, check if the list contains the project SIFBuilder.

Step 4. To run the solution or to compile, click on the menu bar,
Build ⟶ Build Solution, or simply press F7

Wait until the solution is built. This process may take several minutes. A successful build will output the following.

Step 5. To Debug, click on the menu bar,
Debug $\longrightarrow$ Start Debugging, or simply press F5
Wait until OpenSees command window opens.


D:\OpenSees_Edit+Stable\Win32\proj\openSees\.\..\..\bin\OpenSees.exe

```
         OpenSees -- Open System For Earthquake Engineering Simulation
        Pacific Earthquake Engineering Research Center -- 2.4.0

            (c) Copyright 1999,2000 The Regents of the University of California
                                 All Rights Reserved
    (Copyright and Disclaimer @ http://www.berkeley.edu/OpenSees/copyright.html)

        (ThermalVersion.0.1, developed by University of Edinburgh)

OpenSees >
```

Section 10

COMMON ERRORS IN MS VISUAL STUDIO AND THEIR SOLUTION

This document illustrates working instructions for debugging the errors encountered in compiling the open-source framework, Open System for Earthquake Engineering Simulations, **OpenSees** using Microsoft Visual Studio IDE. The document is written to guide the beginners working on development of OpenSees.

COMMON ERRORS AND THEIR SOLUTIONS

Error # C1083
Cannot find twoNodeLink.h

Reason:
Some times the library linkages will be missed out when Visual Studio loads a project. The link needs to be re-established to successfully run the solution

Solution:
Right click on the class "actor" $\longrightarrow$ Properties $\longrightarrow$ C/C++ $\longrightarrow$ Additional Include Directories $\longrightarrow$ $<$ *Edit* $>$ $\longrightarrow$ add the path of the folder containing the missing file `..\..\..\src\element\twoNodeLink`
Trick: Search for the missing file in the directory to find its path/location

Error # LNK2019
unresolved external symbol

Actual error line:
material.lib(TclModelBuilderUniaxialMaterialCommand.obj) : error LNK2019: unresolved external symbol "void * __cdecl OPS_DoddRestr(void)"(?OPS_DoddRestr@@YAPAXXZ) referenced in function "int __cdecl TclModelBuilderUniaxialMaterialCommand(void *,struct Tcl_Interp *,int,char const * *,class Domain *)"
(?TclModelBuilderUniaxialMaterialCommand@@YAHPAXPAUTcl_Interp@@HPAPBDPAV

Domain@@@Z)

Reason:

OpenSees is an open source framework where developers from all over contribute. Sometimes, the program they have written, although it works, may encounter issues when run on platforms on which they were built. For example, if the programs were built on unix platform, they might have linker issues when run using Microsoft Visual Studio. The easiest solution would be to comment out those lines of codes and make rest of the codes run properly. This is the best solution ONLY when the part of the codes causing error are UNIMPORTANT for your applications.

Solution:

Search for the .cpp file corresponding to the object file which is referenced in the error message. For example, TclModelBuilderUniaxialMaterialCommand.cpp and comment out the following lines pertaining to the error:

```
//extern void *OPS_Dodd_Restrepo(void);
/* } else if ((strcmp(argv[1],"Dodd_Restrepo") == 0) ||
    (strcmp(argv[1],"DoddRestrepo") == 0) ||
    (strcmp(argv[1],"Restrepo") == 0)) {
    void *theMat = OPS_Dodd_Restrepo();
    if (theMat != 0)
    theMaterial = (UniaxialMaterial *)theMat;
    else
return TCL_ERROR;*/
```

Error # C1083 (*with c1xx*)

Actual error line:

c1xx : fatal error C1083: Cannot open source file: `'..\..\..\SRC\element\bearing\ElastomericX.cpp'` : No such file or directory

c1xx : fatal error C1083: Cannot open source file: `'..\..\..\SRC\element\bearing\HDR.cpp'` : No such file or directory

c1xx : fatal error C1083: Cannot open source file: `'..\..\..\SRC\element\bearing\LeadRubberX.cpp'` : No such file or directory

Reason:

Some times the library linkages will be missed out when Visual Studio loads a project. The link needs to be re-established to successfully run the solution.

Solution:

Step 1: *(if it doesn't already exist)*

Right click on the class "element" ⟶ Properties ⟶ Additional Include Directories ⟶ < *Edit* > ⟶ add the path of the folder containing the missing file `..\..\..\src\element\elastomericBearing`

Step 2: *(if it exists)*

Right click on the class "element" ⟶ Properties ⟶ Additional Include Directories ⟶ < *Edit* > ⟶ delete the path of the folder containing the missing file `..\..\..\src\element\`

`bearing` if already done, skip steps 1 & 2

Step 3:

Manually delete these lines in the file **element.vcxproj** located in `OpenSees/Win32/proj/element`

```
  <ClInclude Include="..\..\..\SRC\element\bearing\ElastomericX.h" />
  <ClInclude Include="..\..\..\SRC\element\bearing\HDR.h" />
  <ClInclude Include="..\..\..\SRC\element\bearing\LeadRubberX.h" />
```

Manually delete these lines in the file **element.vcxproj.filters** located in `OpenSees/Win32/proj/element`

```
 <ClInclude Include="..\..\..\SRC\element\bearing\ElastomericX.h">
  <Filter>bearing</Filter>
 </ClInclude>
 <ClInclude Include="..\..\..\SRC\element\bearing\HDR.h">
  <Filter>bearing</Filter>
 </ClInclude>
 <ClInclude Include="..\..\..\SRC\element\bearing\LeadRubberX.h">
  <Filter>bearing</Filter>
 </ClInclude>
```

Note: *Save and close the solution file (*.sln) before modifying the *.vcxproj and *.vcxproj.filters*

Error # LNK1181

LINK : fatal error LNK1181: cannot open input file 'cssparse.lib' **Reason:**

Missing library file

Solution:

Rebuild the project:

Right click over the class (Example: cssparse) in the solution explorer ⟶ Project Only ⟶ Rebuild only cssparse.

Note: *This may trigger same error from other projects. Keep rebuilding the projects individually until the error vanishes.*

Error # LNK1123

Actual error line:

LINK : fatal error LNK 1123: failure during conversion to COFF: file invalid or corrupt

Solution:

Right click on the project (startup project: eg., OpenSees) ⟶ Configuration properties ⟶ Linker ⟶ General ⟶ Enable Incremental Linking ⟶ "NO (/INCREMENTAL:NO)"

Error # C2065

error C2065: 'MAT_TAG_ElasticmtaerialNewThermal' : undeclared identifier **Reason:**

No identifier or Tag defined. Whenever a new material is created, it should be assigned a unique identifier or tag. If not defined, or if wrongly defined, it gives an error.
Solution:

Add a material tag in the source file *classTags.cpp* by adding the following line.
#define MAT_TAG_ElasticMaterialNewThermal 111
Here, *111* is the developer defined material tag. This number should be unique and different from any other material tags to avoid duplication.

Error # LNK2019 ***followed by*** **LNK1120**
unresolved external symbol

Actual error line:
error LNK2019: unresolved external symbol "public: __thiscall ElasticBeam2dBuck::ElasticBeam 2dBuck(int,double,double,double,int,int,class CrdTransf &,double,double,double)" (??0ElasticBeam 2dBuck@@QAE@HNNNHHAAVCrdTransf@@NNN@Z) referenced in function "int __cdecl Tcl ModelBuilder_addElasticBeam(void *,struct Tcl_Interp *,int,char const * *,class Domain *,class TclModelBuilder *,int)" (?TclModelBuilder_addElasticBeam @@YAHPAXPAUTcl_Interp@@HPAPBDPAVDomain@@PAVTclModelBuilder@@H@Z) xx> `.\..\..\bin/openSees.exe` : fatal error LNK1120: 1 unresolved externals
Reason:
This error occurs when the program is unable to find the source and header files. Some times the version of OpenSees solution checked out from SVN does not automatically include the source and header files which were newly added to the solution. These files have to be checked manually and added to the appropriate project in the solution explorer.
Solution:

The error message always shows the class in which the error has occurred. In this example, it is **ElasticBeam2dBuck**. Search for the missing header and source files. *ElasticBeam2dBuck.cpp* and *ElasticBeam2dBuck.h* files are found in `OpenSees\SRC\element\elasticBeamColumn\`. Find the filter *elasticBeamColumn* under element. Right click on the filter ⟶ Add ⟶ Existing Item ⟶ select the source and header file ⟶ click Add.
Rebuild the project and debug, if successful.

POST - DEBUGGING

If the build is successful, check if the project SIFBuilder is recognized by the framework. Follow the steps given below.
Step 1: Click on the menu bar,
Debug ⟶ Start Debugging, or simply press F5.
Alternately, you may also press the debig button the debug button *(solid green arrowhead pointing right)* on the on the toolbar. **Step 2:** On successful completion, an OpenSees Command Window

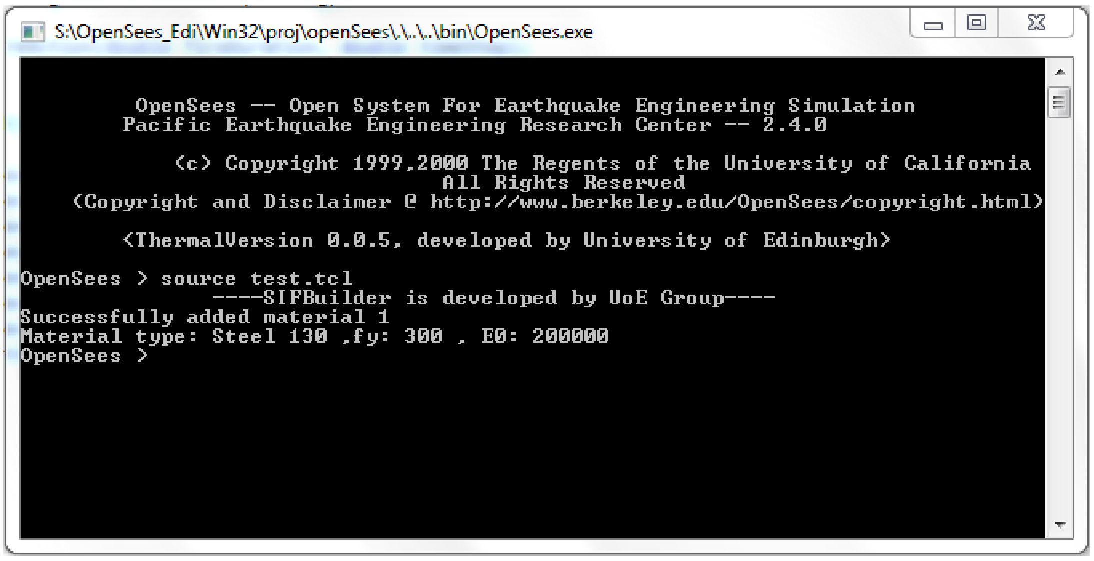
should pop out.

Step 3: Type the command *source test.tcl* at the command prompt in the opensees command window.

OpenSees > source test.tcl

If a project is recognized successfully,

you will see this message.



S:\OpenSees_Edi\Win32\proj\openSees\.\..\..\bin\OpenSees.exe

```
         OpenSees -- Open System For Earthquake Engineering Simulation
        Pacific Earthquake Engineering Research Center -- 2.4.0

            (c) Copyright 1999,2000 The Regents of the University of California
                                 All Rights Reserved
    (Copyright and Disclaimer @ http://www.berkeley.edu/OpenSees/copyright.html)

        <ThermalVersion 0.0.5, developed by University of Edinburgh>

OpenSees > source test.tcl
               ----SIFBuilder is developed by UoE Group----
Successfully added material 1
Material type: Steel 130 ,fy: 300 , E0: 200000
OpenSees >
```

General Instructions

When running the solutions, even after applying solutions to an error, if the solution fails, try these combinations:

Attempt 1: Build $\longrightarrow$ Solution

if this fails,

Attempt 2: Build $\longrightarrow$ Rebuild Solution

if this too fails,

Attempt 3: Build $\longrightarrow$ Clean Solution; Build $\longrightarrow$ Build Solution

Section 11

ADDING YOUR OWN CODE TO OPENSEES

Externally written codes (C / C++ / Fortran) can be added to OpenSees interpreters using dynamic link libraries (*.dll) on windows machine. Most commonly added components are:
New Material
New Element
New Solver
New Integrator
New Solution Algorithm
New Recorder

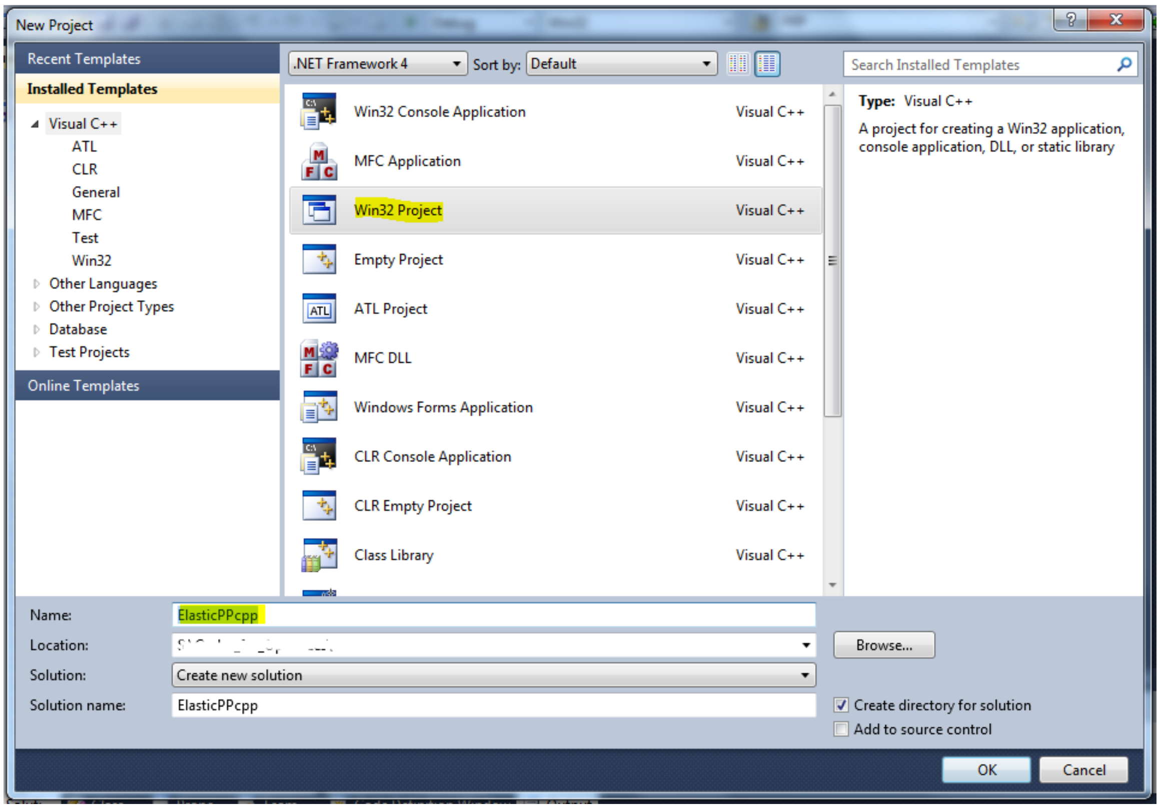
11.1 Generating a dynamic link library (*.dll) using Microsoft Visual Studio

In this section, we will go through the creation of a *dll* file for a new material (*ElasticPPcpp*) written using C++.
The source *(ElasticPPcpp.cpp)* and header *(ElasticPPcpp.h)* files for the new element considered in this example are found in:
`OpenSees\DEVELOPER\material\cpp`
Note: This section does not require to have the OpenSees project open. DLL files can be generated separately and then imported into the project.

Step 1: Open **Microsoft Visual Studio**

Step 2: **File $\longrightarrow$ New $\longrightarrow$ Project**
Choose **win32 project Visual C++** and Type a name which is the same as the name of the class / source / header file. eg., ElasticPPcpp, choose a known location by clicking **Browse...** and click **Ok** to open Win32 Application Wizard.
Click on Application Settings and check the following options and click **Finish**:
Application type: **DLL**
Additional options: **Empty project**

New Project Window


New Project

Recent Templates
Installed Templates
- Visual C++
  - ATL
  - CLR
  - General
  - MFC
  - Test
  - Win32
- Other Languages
- Other Project Types
- Database
- Test Projects

Online Templates

.NET Framework 4 Sort by: Default

Search Installed Templates

Win32 Console Application	Visual C++
MFC Application	Visual C++
Win32 Project	Visual C++
Empty Project	Visual C++
ATL Project	Visual C++
MFC DLL	Visual C++
Windows Forms Application	Visual C++
CLR Console Application	Visual C++
CLR Empty Project	Visual C++
Class Library	Visual C++

Type: Visual C++

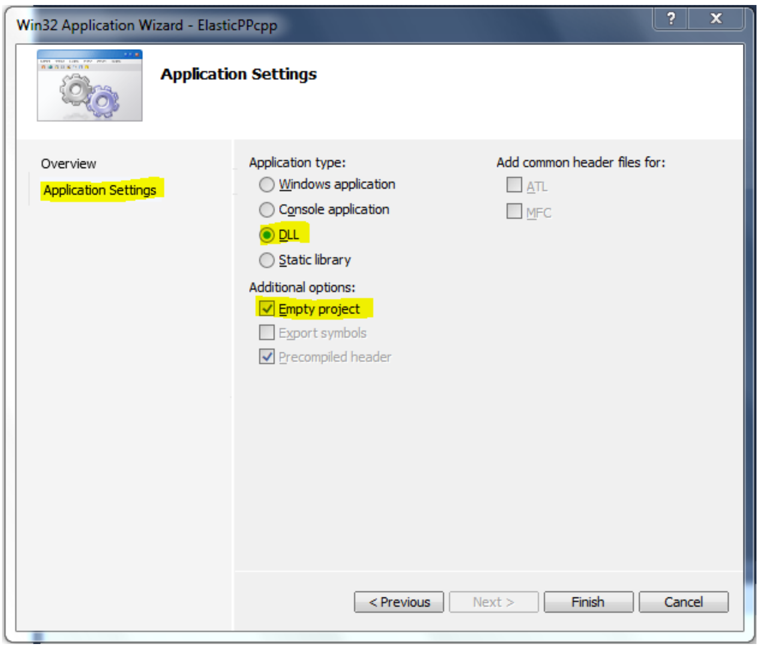
A project for creating a Win32 application, console application, DLL, or static library

Name: ElasticPPcpp
Location: [illegible]
Solution: Create new solution
Solution name: ElasticPPcpp

[✓] Create directory for solution
[ ] Add to source control

Browse... OK Cancel

New Project Window


Win32 Application Wizard - ElasticPPcpp

Application Settings

Overview
Application Settings

Application type:
- ◯ Windows application
- ◯ Console application
- ● DLL
- ◯ Static library

Additional options:
- [✓] Empty project
- [ ] Export symbols
- [✓] Precompiled header

Add common header files for:
- [ ] ATL
- [ ] MFC

< Previous Next > Finish Cancel

Win32 Application Wizard Window

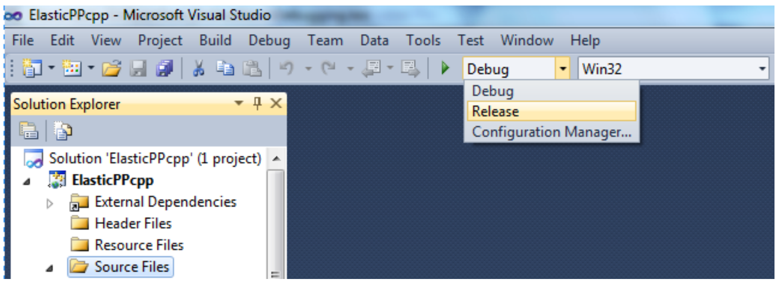
Step 3: Right click on the **Source File** $\longrightarrow$ **Add** $\longrightarrow$ **Existing Item** $\longrightarrow$ Select **ElasticPPcpp.h** and **ElasticPPcpp.cpp** $\longrightarrow$ **Add**

Step 4: **Build ⟶ Build Solution**
1 Failed *(Element API Not Found)*
This step will fail because no core directory has been included.

Step 5: Right click on the project, **ElasticPPcpp** on the solution explorer ⟶ **Properties** ⟶ **Configuration Properties** ⟶ **C/C++** ⟶ General ⟶ Additional Include Directories ⟶ choose < *edit* > from the dropdown ⟶ add the path of **core** directory, **`..\..\..\core`** or the actual path, **`OpenSees\DEVELOPER\core`** ⟶ **Ok** *Note: Core directory is found in* `OpenSees\DEVELOPER`
If you try to build now, it fails again because it is still missing the supporting files from the core directory.

Step 6: In the solution explorer, right click on **Source File** ⟶ **Add** ⟶ **Existing Item** ⟶ **Developer** ⟶ Core ⟶ select all *(ctrl+A)* ⟶ **Add**

Step 7: **Build ⟶ Rebuild Solution**
The program should now run successfully and create a DLL file in **`ElasticPPcpp\Debug`** folder on your computer. Note:
You can also create the DLL for release version of the visual studio by choosing the **Release** option on the toolbar and repeating steps, **5** and **7**. For more information on debug and release versions, see FAQ section.


ElasticPPcpp - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Test Window Help
Debug
Win32
Debug
Release
Configuration Manager...
Solution Explorer
Solution 'ElasticPPcpp' (1 project)
ElasticPPcpp
External Dependencies
Header Files
Resource Files
Source Files

Step 8: Copy the newly generated DLL file from **`ElasticPPcpp\Debug`** to **`OpenSees\DEVELOPER\material\cpp`**. This is an IMPORTANT step.

Step 9: Test the new material using an example script *example1.tcl* located in `OpenSees\DEVELOPER\material\cpp`.
• Copy the binary file, **OpenSees.exe**, to the folder `OpenSees\DEVELOPER\material\cpp`.
• Double click on the command file to open the OpenSees Command Window.
• Run the example file by giving the command:
OpenSees > source example1.tcl.
If successful, the program should exit with a 0.

Note:
• **.dll should be in the same directory as the script*
• *If you can link a digital link library using set Load Library Path, there is no need to have the DLL file in the same directory as the code.*

11.2 Alternate and simple method to add a new material *(with example)*

In the following steps, we will be going over an example to add a new thermal material to the material library in OpenSees, ***ElasticMaterialNewThermal***. The easiest way to go about this exercise is to copy an existing file **ElasticMaterialThermal** and making a suitable modification. This method holds good when you are creating a material, similar to the one that already exists. Note that all materials have some classes and methods in common, which perhaps makes this method an efficient way to add your own code.

1. Got to the appropriate folder that contains the source and the header files for the existing material. For example, in this case *ElasticMaterialThermal.cpp* and *ElasticMaterialThermal.h*. `OpenSees/SRC/material/uniaxial`

2. Make a copy of the header and source files in the same folder.

3. Name it *ElasticMaterialNewThermal.cpp* and *ElasticMaterialNewThermal.h*.

4. Add the two newly created files to the element project in the solution explorer:
   In the solution explorer, expand *material* project and thereafter expand *uniaxial* to see a list of uniaxial materials available in OpenSees. Right click on *uniaxial* $\longrightarrow$ Add $\longrightarrow$ Existing Item. Select the source and header files for the new material from `OpenSees\SRC\material\uniaxial` and click *Add*.

5. Open both the source and header files in a text editor (**Notepad ++** or Microsoft Visual Studio editor) and make changes. In this example, we make things simple just by replacing the keyword, *ElasticMaterialThermal* to *ElasticMaterialNewThermal*.
   Note: MS Visual Studio's find and replace window can be brought up by clicking *ctrl+H* on the keyboard. It consists of two options Quick Find and Quick Replace.
   Type **ElasticMaterialThermal** under *Find what:* and **ElasticMaterialNewThermal** under *Replace with:*. The window also offers to find and replace 'where?' in *Look in:*. Always keep the default Option '**Current Document**' to avoid accidental changes or deletion in the other part of the document.

6. Add a Tag to the newly created material: Open the header file *classTags.h* (`OpenSees\SRC`). Add the following line to assign a classTag.
   #define MAT_TAG_ElasticMaterialNewThermal 111
   Tag is a number (For example, 111 in the current example). This has to be unique and should not be repeated.

7. Add the material to the Visual Studio project file (*.vcxproj)
   Add the following line in *material.vcxproj* file.
   <ClInclude Include=
   "`..\..\..\SRC\material\uniaxial\ElasticMaterialNewThermal.h`" />

<ClCompile Include=
“..\..\..\SRC\material\uniaxial\ElasticMaterialNewThermal.cpp”/>
Note: When you modify the material.vcxproj file outside MS Visual Studio window, a dilog box appears with a warning message: *This file has been modified outside of the source editor. Do you want to reload it?*. Proceed with the option *Overwrite*.

8. Add a function for the newly created material in *TclModelBuilderUniaxialMaterialCommand.cpp*. It is found in `OpenSees\src\material\uniaxial`. The file is also found in the solution explorer.
   Open the file in the source editor by double clicking over it. Add the following line to declaration:
   extern void *OPS_NewElasticMaterialNewThermal(void);
   Add the following in the code:
   } else if (strcmp(argv[1],”ElasticNewThermal”) == 0) {
   void *theMat = OPS_NewElasticMaterialNewThermal();
   if (theMat != 0)
   theMaterial = (UniaxialMaterial *)theMat;
   else
   return TCL_ERROR;
   //- - - - - -End of adding identity for ElasticMaterialNewThermal

9. Rebuild only the material project: Right click on material ⟶ Project Only ⟶ Rebuild Only material.

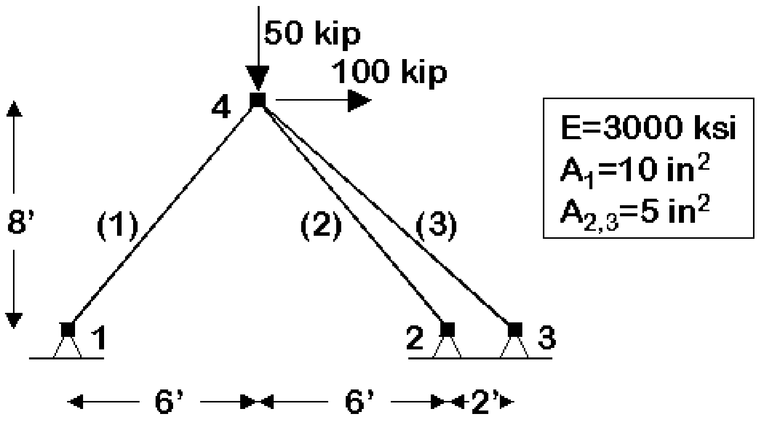
11.3 Testing the Newly Added Material *(or a piece of code)*

Step 1: Add a tcl file *test.tcl* to the in the project *openSees*. Add a line or two using the new material. For example:
model BasicBuilder -ndm 2 -ndf 3;
uniaxialMaterial ElasticNewThermal 1 20000 0.01;
Step 2: Debug the successfully built version of OpenSees with the new material to bring up the OpenSees command window.
Step 3: Source the test.tcl file by typing *source test.tcl.* If the program outputs the desired lines (if added) or exits with no errors, you have SUCCESSFULLY added a new material.

Section 12

WORKING WITH ‘quickMain’

The class ‘quickMain‘ is generally used to write source script for benchmark problems and test the reliability of the written code. The only other way which has superceded this is by adding tcl commands and writing tcl scripts for solving benchmark problems. The latter has gained popularity lately. However, early developers find the ‘quickMain’ method very convenient.[1]
OpenSees Command Language Manual shows some basic structural element examples to demonstrate the successful functioning of OpenSees framework. This section illustrates a simple linear elastic 3-bar truss structure subjected to static loads as shown in the pictures below.
Example:



The example demonstrates a typical finite element exercise by representing a truss configuration using nodes, elements, materials, loads and constraints.
Model:

The model consists of an assembly of three truss elements defined by four nodes, a single load pattern with a nodal load acting on node 4, where the three members join. The truss is maintained in equilibrium by defining constraints at three bottom nodes. All three truss elements are assumed to be made of same elastic material, which is defined by creating a single elastic material object.
Analysis

The model considered in the example is linear and is solved using *Linear Solution* algorithm. A

[1]When checking out and running opensees, always EXCLUDE the quickMain.

Load Control integrator is used as a procedure for applying load. The equations are formed using a banded system, *BandSPD* (banded symmetric positive definite). The equations are numbered using the *RCM* (Reverse Cuthill-McKee) numberer object. The constraints are represented with a *Plain* constraint handler.

Step 1: Follow all the steps under the previous sections to download, build and debug OpenSees source codes.

Step 2: Set *quickMain* as the Start Up project: right click on *quickMain* and choose *'Set as StartUp Project'*.The code for testing should be written in *main.cpp* file under *quickMain*.

Step 3: Right click on the *quickMain* class and select the option *'Project Only'* and then select *'Rebuild Only quickMain'* or simply click *F5*.

Step 4: Once the build completes, the output indicates of the build is successful or not.

Step 5: In case of successful build, click *Ctrl+F5* to open the output window.

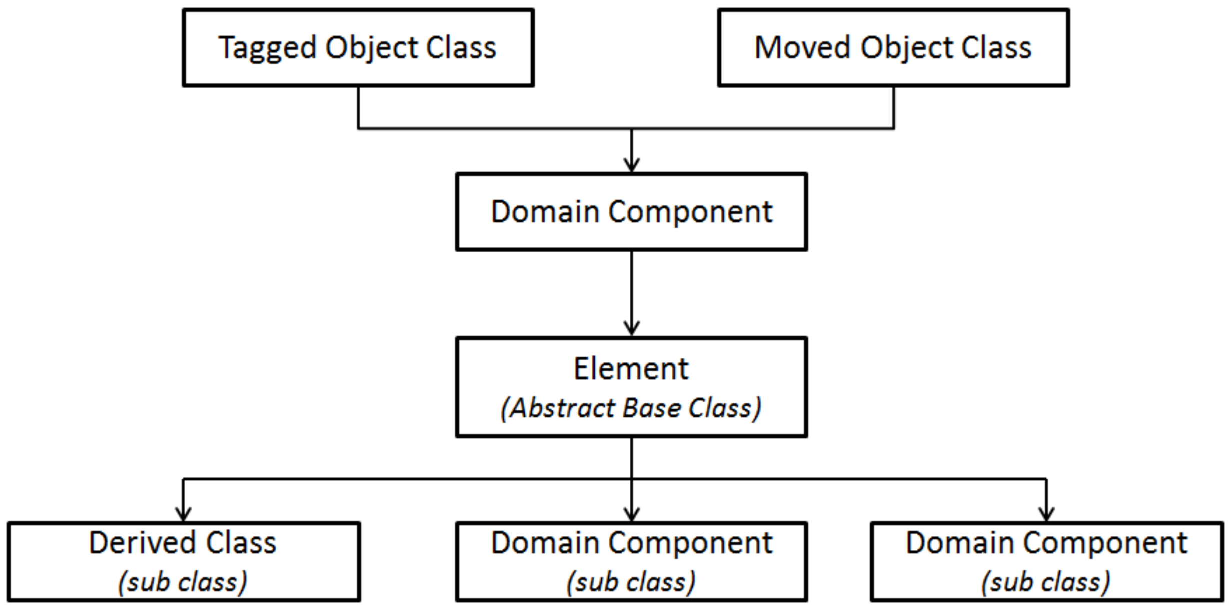
Section 13

ADDING A NEW CODE TO OPENSEES *(The quickMain Method)*

13.1 Basic Steps

1. Provide a new subclass of Element class.
2. Provide an interface function that will be used to parse the input and create a new element.

13.2 Inheritance diagram of an element class:



13.2.1 The Element Class:

```
class Element : public DomainComponent {
  public:
    Element(int tag, int classTag);
    virtual ~Element();



    // initialization
    virtual int setDomain(Domain *theDomain);

    //methods dealing with nodes and number of external dof
    virtual int getNumExternalNodes(void) const =0;
    virtual const ID &getExternalNodea(void) =0;
    virtual Node **getNodePtrs(void) =0;
    virtual int getNumDOF(void) =0;

    //methods dealing with committed state and update
    virtual int commitState(void); // called when a converged solution has been obtained for a time
step
    virtual int revertToLastCommit(void) = 0; // called when the solution algorithm has failed to
converge to a solution to a solution at a time step
    virtual int revertToStart(void); // called when model is rest to      initial conditions
    virtual int update(void); // called when a new trial step has been set at the nodes

    //methods dealing with element stiffness
    virtual const Matrix &getTangetStiff(void) =0;
    virtual const Matrix &getInitialStiff(void) =0;

    //methods dealing with element forces
    virtual void zeroLoad(void);
    virtual int addLoad(ElementLoad *theLoad, double loadFactor);
    virtual const Vector &getResistingForce(void) =0;

    //public methods for output
    void Print(OPS_Stream &s, int flag =0);
    virtual Response *serResponse(const char **argv, int argc, OPS_Stream &theHandler);
    virtual int getResponse(int responseID, Information &eleInformation);

    //method for database/parallel processing
    int sendSelf(int commitTag, Channel &theChannel);
    int recvSelf(int commitTag, Channel &theChannel, FEM_ObjectBroker &theBroker);
```

}

13.3 Example - Truss2D

In the following section we will provide all necessary code to add a new 2D planar truss element into an OpenSees interpreter. The stress-strain relationship will be provided by a UniaxialMaterial object.

13.3.1 Header

The header for the new class, which we will call Truss2D is as follows:

```
//include directives
#include<Element.h>
#include<Matrix.h>
#include<Vector.h>

//forward declarations
class UniaxialMaterial;

class Truss2D : public Element
⟨   public:
    //constructors
    Truss2D(int tag,
        int Nd1, int Nd2,
    UniaxialMaterial &theMaterial,
    double A);

    Truss2D();

    //destructor
    ~Truss2D();

    //initialization
    int setDomain(Domain *theDomain);

    //public methods to obtain information about dof & connectivity
    int getNumExternalNodes(void) const;
    const ID &getExternalNodes(void);
```

Node **getNodePtrs(void);
int getNumDOF(void);

//public methods to set the state of the element
int commitState(void);
int revertToLastCommit(void);
int revertToStart(void);
int update(void);

//public methods to obtain stiffness
const Matrix &getTangetStiff(void) =0;
const Matrix &getInitialStiff(void) =0;

//public methods to obtain resisting force
const Vector &getResistingForce(void) =0;

//method for obtaining information specific to an element
void Print(OPS_Stream &s, int flag =0);
Response *setResponse(const char **argv, int argc, OPS_Stream &s);
int getResponse(int responseID, information and &eleInformation);

//public methods for database and parallel processing

int sendSelf(int commitTag, Channel &theChannel, FEM_ObjectBroker &theBroker);
void Print(OPS_Stream &s, int flag =0);

protected:

private:
//private member functions - only available to objects of the class
double computeCurrentStrain(void) const;

//private attributes - a copy for each object of the class
UniaxialMaterial *theMaterial; //pointer to the material
ID externalNodes; //contains the IDs of end nodes
Matrix trans; //hold the transformation matrix
double L; //length of truss (undeformed configuration)
double A; //area of truss
Node *theNodes[2]; //node pointers

//static data - single copy for all objects of the class
static Matrix trussK; //class wide matrix for returning stiffness

```
    static Vector trussR; //class wide vector for returning residual
}; #endif
```

The header file defines the interface and variables for the class *Truss2D*. It defines the new class to be a subclass of the *Element* class. The public interface consists of two constructors and a destructor in addition to minimal set of methods we showed for the *Element* class. There are no protected data or methods as we do not expect this class to contain further subclasses. In private section, we define the private method, ***computeCurrentStrain()***, and we define a number of private variables and a number of static variables.
The header has a number of ***#include*** directives, one is needed for the base class and every class used as a variable in the list of data (except those that are used as pointers). For those classes which only appear as pointers in the header file **(Node, UniaxialMaterial)**, a forward declaration is all that is needed*(the include can also be used, but using the forward declaration simplifies the dependencies and reduces the amount of code that has to be recompiled later if changes are made).*

13.3.2 Implementation

In another file, ***Truss.cpp***, we place the code that details what the constructors, destructor and methods do. In addition to this, we provide an additional procedure ***OPS_Truss2D()***.
Note: It has the same name as the class with an OPS_ prefix). We will go through each part of the file.

13.3.3 Include Directives

The first part of the file contains the list of includes. It is necessary to have ***#include*** directive for each class and API file that is used within the source file (.cpp) file and is not included in the header file (*.h).

```
#include<elementAPI.h>
#include<G3Globals.h>
#include<Information.h>
#include<Domain.h>
#include<Node.h>
#include<Channel.h>
#include<Message.h>
#include<FEM_ObjectBroker.h>
#include<UniaxialMaterial.h>
#include<Renderer.h>
#include<ElementResponse.h>
```

```
#include<math.h>
#include<stdlib.h>
#include<string.h>
```

13.3.4 Static Variables

Next step is to initialize static variables. For the given example, we ate using **two** static-variables (objects shared by two Truss2D object that is created), one to return the tangent matrix and and the other to return the resisting force.

```
//initialize the class wide variables
Matrix Truss2D::trussK(4,4);
Vector Truss2D::trussR(4);
```

13.3.5 Constructors

After the list of includes, we provide the **two** constructors. The constructors are rather simple. They just initialize all the data variables defined in the header. Note that it is important to set all the initial pointer values to **0**.
The first constructor is the one most typically used. The arguments provide the elements tag, the tags of the two end nodes, the elements area and a copy of the element's material.

The codes in the constructor does the following:

1. The **elements tag** and a **0** ate passed to the Element constructor.
2. The material pointer **theMaterial**, is set to a copy of the material obtained from the material that is passed in the arguments.
3. The **externalNodes** array is set to be an array of size 2 and its values are set to the nodal tags of two nodes.
4. The **theNodes** array component are set to be **0**.

It should be noted that the static variables dealing with length, tranformations and the nodes are set to **0** in the constructors. They will be filled in when the **setDomain()** method is invoked on the object.

```
Truss2D::Truss2D(int tag,
              int Nd1, int Nd2,
              UniaxialMaterial &theMat,
              double a)
:Element(tag, 0),
externalNodes(2),
trans(1,4), L(0.0), A(a)
{
  //get a copy of the material object for your own use
  theMaterial = theMat.getCopy();   if(theMaterial == 0) {

opserr << "FATAL TrussCPP::TrussCPP() - out of memory, could not get a copy of the Material\n";
exit(-1);
{

//fill in the ID containing external node info with node id's
if(externalNodes.Size() != 2)
    opserr << "FATAL TrussCPP::TrussCPP() - out of memory, could not create an ID of size 2
\n";
    exit(-1); }

externalNodes(0) = Nd1;
externalNodes(0) = Nd2;

theNodes[0] = 0;
theNodes[1] = 0;
}
```

The second constructor is called when parallel processing or the database feature of the OpenSees application is used. Its purpose is to create blank Truss2D objects, that will be filled in when the recvSelf() method is invoked on the object.

```
Truss2D::Truss2D()
:Element(0,0),
theMaterial(0),
externalNodes(2),
trans(1,4), K(0.0), A(0,0)
{
    theNodes[0] = 0;
    theNodes[1] = 0;
}
```

13.3.6 Destructor

Then we provide the destructor. In the destructor, all memory that the Truss2D created or was passed to it in the constructor must be destroyed. For our example, we need to invoke the destructor on the copy of the material object.

```
Truss2D::~Truss2D()
{
    if (theMaterial != 0)
        delete theMaterial;
}
```

setDomain() initialization method

The **setDomain()** method is invoked when the truss element is being added to the domain. It is in this method that most of private variables of the project are determined. The method returns **0 if successful** or **a negative number if NOT**. In the method, we obtain pointers to the end nodes, nodal coordinates are obtained and the element length and the transformation materix is set once the coordinates have been obtained.

```
void
Truss2D::setDomain(Domain *theDOmain)
{
    //check Domain is not null - invoked when object removed from a domain
    if(theDomain == 0), {
        return;
    }

    //first ensure the nodes exist in Domain and set the node pointers
    Node *end1Ptr,*end2Ptr;
    int Nd1 = externalNodes(0);
    int Nd2 = externalNodes(1);
    end1Ptr = theDomain− >getNode(Nd1);
    end2Ptr = theDomain− >getNode(Nd2);
    if end1Ptr == 0 {
        opserr << “WARNING Truss2D::setDomain() - at truss” <<this− >getTag()<<“node”
<< Nd1 << does not exist in domain\n”;
        Nd1 << “ does not exist in domain \n”;
        return;
don’t go any further - otherwise segmentation fault occurs
}
    if end2Ptr == 0 {
        opserr << “WARNING Truss2D::setDomain() - at truss” <<this− >getTag()<<“node”
<< Nd2 << does not exist in domain\n";
        Nd2 << " does not exist in domain \n";
        return;
don't go any further - otherwise segmentation fault occurs
    }
    theNodes[0] = end1Ptr;
    theNodes[1] = end2Ptr;
    //call the domain component class method THIS IS VERY IMPORTANT!!!
    this− >DomainComponent::setDomain(theDomain);

    ensure connected nodes have correct number of dof's
    int dofNd1 = end1Ptr− >getNumberDOF();
    int dofNd2 = end2Ptr− >getNumberDOF();
    if ((dofNd1 != 2) |||| (dofNd2 != 2)) lbrace
      opserr << "Truss2D::setDomain(): 2 dof required at nodes\n";
      return;
    }

    //now determine the length and transformation matrix
    const Vector &end1Crd = end1Ptr− >getCrds();
    const Vector &end2Crd = end2Ptr− >getCrds();

    double dx = end2Crd(0)-end1Crd(0);
    double dy = end2Crd(1)-end1Crd(1);

    L = sqrt(dx*dx + dy*dy);

        if (L ==0) {
      opserr << "WARNING Truss2D::setDomain() - Truss2D" <<this− >getTag() << "has
zero length\n";
      return;
//don't go any further - otherwise divide by 0 error occurs
    }

    double cs = dx/L;
    double sn = dy/L;

    trans(0,0) = -cs;
    trans(0,1) = -sn;
    trans(0,2) = cs;
    trans(0,3) = sn;
rbrace
```

13.3.7 Methods dealing with nodes

Next comes four rather simple methods that return basic information about the element nodes. These are one line methods that does not need any explanation.

```
int
Truss2D::getNumExternalNodes(void)const
{
    return 2;
}

constID &
Truss2D::getExternalNodes(void)
{
    return externalNodes;
}

Node**
Truss2D::getNodePtrs(void)
{
    return theNodes;
}

int
Truss2D::getNodePtrs(void)
{
    return 4;
}
```

13.3.8 Methods dealing with current state

```
int
Truss2D::commitState()
lbrace
    return theMaterial- >commitState();
}

int
```

```
Truss2D::revertToLastCommit()
lbrace
    return theMaterial− >revertToLastCommit();
}

int
Truss2D::revertToStart()
lbrace
    return theMaterial− >revertToStart();
}

int
Truss2D::update()
lbrace
    textbf//determine the current strain given trial displacement at the nodes
    double strain = this− >computeCurrentStrain();

    //set the strain in the materials
    theMaterial− >setTrialStrain(strain)

return 0;
{
```

13.3.9 Methods to return Tangent Matrix

In both methods we obtain appropriate tangent from the material and use this to return the transformed matrix.

```
const Matrix &
Truss2D::getTangentStiff(void)
{
    if (L == 0.0)lbrace //length = zero - problem in setDomain() warning message already printed
        trussK.Zero();
        return trussK;
    }

    //get the current E from the material for the last updated strain
    double E = theMaterial− >getTangent();

    //form the tangent stiffness matrix
    trussK = trans^trans;
    trussK *= A*E/L;

    //return the matrix
    return trussK;
}

const Matrix &
Truss2D::getInitialStiff(void)
{
    if (L == 0.0)lbrace //length = zero - problem in setDomain() warning message already printed
        trussK.Zero();
        return trussK;
    }

    //get the current E from the material for the last updated strain
    double E = theMaterial− >getInitialTangent();

    //form the tangent stiffness matrix
    trussK = trans^trans;
    trussK *= A*E/L;

    //return the matrix
    return trussK;
}
```

13.3.10 Methods to return resisting force

In this method, we obtain the stress from the material and use this to return the transformed force vector.

```
const Vector &
Truss2D::getResistingForce()
{       if(L == 0.0) { //length == 0, problem in setDomain()
        trussR.Zero();
        return trussR;       }

    //want: R = Ku - Pext

    //force = F * transformation
    double force = A*theMaterial− >getStress();
    for (int i=0;i¡4;i++)
```

```
        trussR(i) = trans(0,i)*force;

    return trussR;
}
```

13.3.11 Methods dealing with output

Information is obtained by the user when the print command is invoked by the user and also when the user issues the recorder command. When the Print command is invoked, Print method is also invoked. This method simply prints information about the element, and then asks the material to do likewise.

```
void
Truss2D::Print(OPS_Stream &s, int flag)
{
   s << “Element: ” <<this− >getTag();
   s << “type: Truss2D iNode: ” << externalNodes(0);
   s << “jNode ” << externalNodes(1);
   s << “Area: ” << A;
   s << “\t Material” << *theMaterial;
}
```

There are two methods used by the element recorders.

1. The first method, **setResponse()**, is called then the recorder is created. The element informs the recorder that it can respond to a request of that type, if it cannot respond to the request, it returns a 0, otherwise it returns a response object. The response includes a pointer to the element, an integer flag used to ID the response when the getResponse() method is called, and a Vector detailing the size of the response.

2. The second method, getResponse(), is called by the recorder when it is actually recording the information.

```
Response *
Truss2D::setResponse(const char **argv, int argc, OPS_Stream &output)
{
   Response *theResponse = 0;

   output.tag(“ElementOutput”);
   output.attr(“eleType”,this− >getClassType());
   output.tag(“ElementOutput− >”);
```

```
    int numNodes = this− >getNumExternalNodes();
    const ID &nodes = this− >getNumExternalNodes();
    static char nodeData[32];

    for(int i=0;i¡numNodes;i++) {
        sprintf(nodeData, “node%d”,i+1);
        output.attr(nodeData,nodes(i));
}

    if(strcmp(argv[0],“force”) == 0 || strcmp(argv[0],“forces”) == 0 ||
        (strcmp(argv[0],“globalForce”) == 0 || (argv[0],“globalForces”) == 0) {
    const Vector &force = this− >getResistingForce();
    int size = force.Size();
    for(int i=0;i¡size;i++) {
        sprintf(nodeData,“P%d”,i+1);
        output.tag(“ResponseType”,nodeData);
    }

theResponse = new ElementResponse(this, 1, this− >getResistingForce());
}

else if((strcmp(argv[0],“dampingForce”) == 0 || strcmp(argv[0],“forces”) == 0)
{
    const Vector &force = this− >getResistingForce();
    int size = force.Size();
    for(int i=0;i¡size;i++) {
        sprintf(nodeData,“P%d”,i+1);
        output.tag(“ResponseType”,nodeData);
    }
theResponse = new ElementResponse(this, 2, this− >getResistingForce());
} else if (strcmp(argv[0],“axialForce”) == 0)
        return new ElementResponse(this, 3, 0.0);

  output.endTag();   return theResponse;
}

int
Truss2D::getResponse(int responseID, Informatuon &eleInfo)
{
  double strain;

  switch(responseID) {
```

```
  case -1;
      return -1;
  case 1: //global forces
      return eleinfo.setVector(this− >getResistingForce());
  case 2:
      return eleinfo.setVector(this− >getRaleighDampingForces());
  case 3:
      return eleinfo.setDouble(A*Material− >getStress());
  default:
      return 0;
  }
}
```

13.3.12 Methods dealing with databases and parallel processing

Two methods, one each for database and parallel processing have been provided, if the user specifies database or parallel processing feature in OpenSees. If neither is to be used, OpenSees returns a negative value in both the methods. The idea is that, the elements 'packs' its information using Vector and ID objects, which is further sent to a Channel Object. On the other hand, the blank element will receive the Vector and ID data, 'unpack's it and assigns the variables.

```
int
Truss2D::sendSelf(int commitTag, Channel &theChannel)
{
    int res;

    //we don't check if dataTag == 0 for the Element
    //objects as that is taken care of in a commit by the domain
    //object. Therefore, checking need not be done while sending the data
    int dataTag = this− >getDbTag();

    //Truss2D packs its data into a Vector and sends this to theChannel
    along with the DbTag and the commit tag passed in the arguments

    Vector data(5);
    data(0) = this− >getTag();
    data(0) = A;
    data(2) = theMaterial− >getClassTag();
    int matDbTag = theMaterial− >matDbTag();
    NOTE: We do have to ensure that the material has a database
    tag if we are sending to a database channel.
    if (matDbTag == 0) {
        matDbTag = theChannel.getDbTag();
        if (matDbTag != 0) {
          theMaterial->setDbTag(matDbTag);
    }
    data(3) = matDbTag;

    res = theChannel.sendVector(dataTag, commitTag, data);
    if(res<0) {
      opserr << "WARNING Truss2D::sendSelf() - failed to send Vector\n";
      return -1;
    }

    //Truss2D then sends the tags of its two end nodes
    res = theChannel.sendVector(dataTag, commitTag, externalNodes);
    if(res<0) {
      opserr << "WARNING Truss2D::sendSelf() - failed to send ID\n";
      return -2;
    }

    //finally Truss2D invokes its material object to show up
    res = theMaterial.sendSelf(commitTag, theChannel);
    if(res<0) {
      opserr << "WARNING Truss2D::sendSelf() - failed to send the Material\n";
      return -3;
    }

    return 0;
{

int
Truss2D::recvSelf(int commitTag, Channel &theChannel, FEM_ObjectBroker &theBroker)
{
    int res;
    int dataTag = this->getDbTag();


    //Truss2D creates a Vector, receives a Vector and then sets the
    internal data with the data in the Vector
```

```
Vector data(5);
res = theChannel.recvVector(dataTag, commitTag, data);
if(res<0) {
  opserr << “WARNING Truss2D::recvSelf() - failed to receive Vector\n”;
  return -1;
}

this− >setTag((int)data(0));
A = data(1);
//Truss2D now receives the tags of its two external nodes
res = theChannel.recvVector(dataTag, commitTag, externalNodes);
if(res<0) {
  opserr << “WARNING Truss2D::sendSelf() - failed to send ID\n”;
  return -2;
}

//we create a material object of the correct type,
//sets its database tag and commands this new object to show up,
int matClass = data(2);
int matDb = data(3);

theMaterial = theBroker.getNewUniaxialMaterial(matClass);
if(theMaterial == 0); {
  opserr << “WARNING Truss2D::recvSelf() - failed to create a Material\n”;
  return -3;
}

//we set the dbTag before we receive the material - this is important
theMaterial− >setDbTag(matDb);
res = theMaterial− >recvSelf(commitTag, theChannel, theBroker);
if (res<0) {
  opserr << “WARNING Truss2D::recvSelf() - failed to receive the Material\n”;
  return -3;
}
```

Section 14

ADDING A Tcl COMMAND TO OPENSEES

*For questions on this section, email Xu Dai at **x.dai@ed.ac.uk** or Liming Jiang at **liming.jiang@ed.ac.uk***

Tcl (originally from Tool Command Language, but conventionally spelled “Tcl” rather than “TCL”; pronounced as “tickle” or “tee-see-ell”) is a scripting language created by John Ousterhout. Originally “born out of frustration”, according to the author, with programmers devising their own languages intended to be embedded into applications, Tcl gained acceptance on its own. It is commonly used for rapid prototyping, scripted applications, GUIs and testing. Tcl is used on embedded systems platforms, both in its full form and in several other small-footprint versions.
Currently Tcl 8.5 is being used by OpenSees, and useful links include:
Tcl/Tk documentation: **http://www.tcl.tk/doc/**
Tcl commands list: **http://www.tcl.tk/man/tcl8.5/TclCmd/contents.htm**
Tcl Tutorial: **http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html**

14.1 Creating your own command

Enumerated below are typical command lines used to create Tcl commands in **TclHeatTransferModule** class.

1. Use Tcl_CreateCommand (This is the standard API of Tcl) to add a new command- FireModel:
   Tcl_CreateCommand(interp,“FireModel”,(Tcl_CmdProc*)TclHeatTransferCommand_add FireModel,(ClientData)NULL, NULL);
2. Specify the Procedure corresponding to this keyword:
   int TclHeatTransferCommand_addFireModel(ClientData clientData, Tcl_Interp *interp, int argc, TCL_Char **argv)(. . . ·)

3. TclModelBuilder or domain classes hold the tags and the pointers of materials, elements, firemodels and so on, using ArrayOfTaggedObjects. Alternatively MapofTaggedObjects is responsible to storing the pointers for auto-loop search and application.
   theFireModels = new ArrayOfTaggedObjects(10);
4. "new" is C++ standard command for creating a pointer to the memory space where the corresponding class is stored:
   theFireModel = new LocalizedFireEC1(FireModelTag, crd1, crd2, crd3, D, Q, H, lineTag);
5. Adding and returning the object pointer:
   theTclHTModule-¿addFireModel(theFireModel);
   FireModel *theFireModel = theTclHTModule-¿getFireModel(FireModelID);
6. Get integer input from the interpreter, similarly Tcl_GetDouble is for obtaining double type value from the interpreter:
   if (Tcl_GetInt(interp, argv[2], &FireModelTag) != TCL_OK) {. . . ·}
7. Checking the keyword of command:
   else if(strcmp(argv[1],"parametric") == 0 || strcmp(argv[1],"Parametric") == 0) {. . . ·}
8. Printing out warning or error:
   opserr << "WARNING invalid thermal inertia of the compartment boundaries" << endln;
9. Argc: number of arguments from a line-input in the command line prompt, Argv[n]: the (n+1)th argument (component in the input arrary)
   i.e., Node 1 1 0; Argc=4, Argv[3]=0

14.2 Example for adding Tcl commands for fire models

```
//Add Fire Model
int
TclHeatTransferCommand_addFireModel(ClientData clientData, Tcl_Interp *interp, int argc, TCL_Char
**argv)



// checking the TclHTModule exists or not
if (theTclHTModule == 0)
opserr << "WARNING current HeatTransfer Module has been destroyed - HTPattern\n";
return TCL_ERROR;
}
// checking the HTDomain exists or not
if (theHTDomain == 0)
opserr << "WARNING no active HeatTransfer Domain - HTPattern\n";
return TCL_ERROR;
}
```

```
//create a pointer to base class
FireModel* theFireModel=0;
int FireModelTag = 0;

//get the fireModel tag;
//Tcl_GetInt is a Tcl standard function for getting the integer input from the interpreter, if it
successfully receives the integer, the function will return TCL_OK, otherwise it will cause an error
and return TCL_ERROR
if (Tcl_GetInt(interp, argv[2], &FireModelTag) != TCL_OK)
opserr << “WARNING:: invalid entity tag for defining HT constants: ” << argv[1] << “\n”;
return TCL_ERROR;
}

int count=2;
//It is a common approach to use count recording the current location of argument.
Thus the count will add 1 after checking the argument.
// The standard format of TCL command will be like “ Firemodel standard 1 ” // or “Firemodel
localised 1 origin $locx $locy $locz firepars
//standard fire curve;
if(strcmp(argv[1],“-standard”) == 0 || strcmp(argv[1],“standard”) == 0 || strcmp(argv[1],“Standard”)
== 0){
      theFireModel = new NorminalFireEC1(1 , 1);

    }
    //hydroCarbon fire curve;
    else if(strcmp(argv[1],“hydroCarbon”) == 0 ||
strcmp(argv[count],“HydroCarbon”) == 0){
      theFireModel = new NorminalFireEC1(1 , 3); // hydrocarbon fire tag is 3;
    }
    //Paramtetric fire
    else if(strcmp(argv[1],“parametric”) == 0 || strcmp(argv[1],“Parametric”) == 0){
        double thi=0; double avent=0; double hvent=0; double atotal=0; double afire=0; double
qfire=0; double Tlim=0;
        count++;
        if(argc==10){
            if (Tcl_GetDouble(interp, argv[count], &thi) != TCL_OK) {
            opserr << “WARNING invalid thermal inertia of the compartment boundaries” <<
endln;
            opserr << “ for HeatTransfer fire model: ” << argv[1] << endln;
            return TCL_ERROR;
            } count++;
            if (Tcl_GetDouble(interp, argv[count], &avent) != TCL_OK) {
            opserr << “WARNING invalid total area of vertial openings on walls” << endln;
            opserr << “ for HeatTransfer fire model: ” << argv[1] << endln;
            return TCL_ERROR;
            } count++;
            if (Tcl_GetDouble(interp, argv[count], &hvent) != TCL_OK) {
            opserr << “WARNING invalid weighted average of window heights on walls” << endln;
            opserr << “ for HeatTransfer fire model: ” << argv[1] << endln;
            return TCL_ERROR;
            } count++;
            if(Tcl_GetDouble(interp, argv[count], &atotal) != TCL_OK) {
            opserr << “WARNING invalid total area of the compartment(including walls)” <<
endln;
            opserr << “ for HeatTransfer fire model: ” << argv[1] << endln;
            return TCL_ERROR;
            } count++;
            if (Tcl_GetDouble(interp, argv[count], &afire) != TCL_OK) {
            opserr << “WARNING invalid area of the floor with fire” << endln;
            opserr << “ for HeatTransfer fire model: ” << argv[1] << endln;
            return TCL_ERROR;
            } count++;
            if (Tcl_GetDouble(interp, argv[count], &qfire) != TCL_OK) {
            opserr << “WARNING invalid total design fire” << endln;
            opserr << “ for HeatTransfer fire model: ” << argv[1] << endln;
            return TCL_ERROR;
            } count++;
            if (Tcl_GetDouble(interp, argv[count], &Tlim) != TCL_OK) {
            opserr << “WARNING invalid time levels corresponds to different fire growth rate” <<
endln;
            opserr << “ for HeatTransfer fire model: ” << argv[1] << endln;
            return TCL_ERROR;
            }
        }
        else
            opserr<< “WARNING:: Defining Parametric fire: “<<argv[2]<<” recieved insufficient
arguments” << ”\n”;

        theFireModel = new ParametricFireEC1(FireModelTag, thi, avent, hvent, atotal, afire, qfire,
Tlim);
    }
    //localised fire curve;
    else if(strcmp(argv[1],“localised”) == 0 ||
strcmp*argv[1],“Localised”) == 0){

    count++; //count should be updated
    double crd1=0.0; double crd2=0.0; double crd3=0.0;
    double D=0; double Q=0; double H=0; int lineTag=0;


    //Add a tag for location of origin;
    if(strcmp(argv[count],“-origin”) == 0 || strcmp(argv[count],“origin”) == 0){
      count++;
      if (Tcl_GetDouble(interp, argv[count], &crd1) != TCL_OK) {
      opserr << “WARNING invalid x axis coordinate of fire origin” << endln;
      opserr << “ for HeatTransfer localised fire model: ” << argv[count] << endln;
      return TCL_ERROR;
      }
      count++;
      if (Tcl_GetDouble(interp, argv[count], &crd2) != TCL_OK) {
      opserr << “WARNING invalid y axis coordinate of fire origin” << endln;
      opserr << “ for HeatTransfer localised fire model: ” << argv[count] << endln;
      return TCL_ERROR;
      }
      count++;

      if (Tcl_GetDouble(interp, argv[count], &crd3) == TCL_OK) {
      //if the z loc is successfully recieved, count should be added with 1;
      count++;
      }
      else
      {
        //it possible not to have a z loc for localised fire definition
        opserr << “WARNING invalid z axis coordinate of fire origin” << endln;
        opserr << “ for HeatTransfer localised fire model: ” << argv[count] << endln;
        crd3=0.0;
      }
    }
    //end of fire origin, waiting for firePars;
    if(strcmp(argv[count],“-firePars”) == 0 || strcmpargv[count],“firePars”) == 0){
      count++;
      if (Tcl_GetDouble(interp, argv[count], &D) != TCL_OK) {
      opserr << “WARNING invalid diameter of the fire source” << endln;
      opserr << “ for HeatTransfer localised fire model: ” << argv[count] << endln;
      return TCL_ERROR;
```

```
            }
            count++;
            if (Tcl_GetDouble(interp, argv[count], &Q) != TCL_OK) {
            opserr << “WARNING invalid rate of heat release” << endln;
            opserr << “ for HeatTransfer localised fire model: ” << argv[count] << endln;
            return TCL_ERROR;
            }
            count++;
            if (Tcl_GetDouble(interp, argv[count], &H) != TCL_OK) {
            opserr << “WARNING invalid distance between the fire source and the ceiling” <<
endln;
            opserr << “ for HeatTransfer localised fire model: ” << argv[count] << endln;
            return TCL_ERROR;
            }
            count++;
            //detect argument for linetag;
            if(argc-count>0){
                if (Tcl_GetInt(interp, argv[count], &lineTag) == TCL_OK) {
                opserr << “WARNING invalid central line tag ” << endln;
                opserr << “ for HeatTransfer localised fire model: ” << argv[count] << endln;
                return TCL_ERROR;
                }
            }
            else {
                opserr << “Central line tag for localised fire “<< argv[1]<<”is set as default:3” <<
endln;
                lineTag=3;
            }
        }
        else {
            opserr<< “WARNING:: Defining Localised fire ”<<argv[2]
<<“ expects tag:-firePars or firePars” << “\n”;          }

theFireModel = new LocalizedFireEC1(FireModelTag, crd1, crd2, crd3, D, Q, H, lineTag);
      }
      //else ———-
      else{
        opserr<<“WARNING unknown fire model type”<< argv[1];
        opserr<<“- for fire model ”<<FireModelTag;
        opserr<<“ valid types: standard, hydroCarbon,parametric, localised.. \n”;
        return TCL_ERROR;
      }
```

```
if(theFireModel!=0){ theTclHTModule->addFireModel(theFireModel);     }
else
   {
      opserr<<"WARNING: TclHTModule fail to add FireModel: "<<argv[1]<<endln;
   }

   return TCL_OK;

}
```

Section 15

RUNNING OPENSEES IN LINUX

15.1 General Linux Commands *(useful while running OpenSees)*

sudo - “**s**uper **u**ser **do**” or “**s**ubstitute **u**ser **do**” allows user to run programs with the security privileges of another user.

sudo -i - “**s**uper **u**ser **do -i**” or “**s**ubstitute **u**ser **do -i**” allows user to access the root directory. *(most applications run without root. However, some programs demand the root access)*

md - “**m**ake **d**irectory” command allows user to create a new directory / folder.

cd - “**c**hange **d**irectory” command allows user to change the directory. i.e., move from one directory to another or simply navigate along a path.

pwd - “**p**resent **w**orking **d**irectory” allows user to view the current working directory.

ls - “**l**i**s**t” command allows user to view the files and directories of the current folder.

ls -l - “**l**i**s**t -l” command allows user to view the permission of files and directories of the current folder.

ls -l ‘filename with extension’ - “**l**i**s**t -l *filename with extension*” command allows user to view the permission of files of the current folder.

ls -ld ‘foldername’ - “**l**i**s**t -ld *foldername*” command allows user to view the permission of folder.

15.1.1 Identity used for obtaining permissions or permission types

r - User has the permission to read the contents of the directory/file (using *ls* command). The number equivalent of this permission is 4.

w - User has the permission to write into the directory/file (create files and directories using *mkdir*). The number equivalent of this permission is 2.

x - User has the permission to enter the directory (change directory using *cd*command. The number equivalent of this permission is 1.

15.1.2 Permission identity for user groups

u - Owner permission only
g - Group permission along with owner permission
a or o - Permission for all users

15.1.3 Permutation of numbers used to set permissions

0 no permission
1 execute
2 write
3 write and execute $(2 + 1)$
4 read
5 read and execute $(4 + 1)$
6 read and write $(4 + 2)$
7 read, write and execute $(4 + 2 + 1)$
chmod $a + x$ - "**ch**ange **mod**e **a+x**" command allows user to set permission to execute a command for all users and *(**a** represents all users and **x** represents permission to execute).*
chmod 777 - "**ch**ange **mod**e **777**" command allows all users to read, write and execute a command.
cd .. - "change directory .." command allows user to navigate to the previous directory in the path.
cd ˜ - "change directory " command allows user to move back to the home directory.
clear - this command clears the screen in the working terminal.
echo $PATH - this command shows the library paths.
echo $LD\_LIBRARY\_PATH$=. - this command allows the user to set the library path to .
echo $LD\_LIBRARY\_PATH$=. - this command allows the user to set the library path to .
***'program name'* - -version** - this command allows the user to check the current version of an installed program. eg., gcc - -version.

15.2 Installing Necessary Packages to run OpenSees

Step 1. Open Terminal

Step 2. Download and install the subversion package key in the following command at the terminal
sudo apt-get install subversion

Step 3. Checkout OpenSees from the subversion *(choose one of the two)*
Berkeley Version

```
svn co svn://opensees.berkeley.edu/usr/local/svn/OpenSees/trunk@5363 OpenSees
```

Edinburgh Version

```
svn co https://svn.ecdf.ed.ac.uk/repo/see/OpenSeesEd/OpenSees/
```

Step 4: Create two directories *(bin and lib)* under the home directory

```
mkdir bin
mkdir lib
```

Step 5: Install the following packages
make
gcc
g++
gfortran
mysql-server
tcl8.5
tcl8.5-dev
tk8.5
tk8.5-dev
libelf-dev
libgl1-mesa-dev
libglu1-mesa-dev
libpng-dev
Run the following in the terminal

```
sudo apt-get install make gcc g++ gfortran mysql-server tcl8.5 tcl8.5-dev tk8.5 tk8.5-dev libelf-dev libgl1-mesa-dev
libglu1-mesa-dev libpng-dev
```

Step 6: Change the read permissions of the OpenSees directory

```
sudo chmod 777 OpenSees
```

Step 7: Key in the command to make the solution of OpenSees

```
sudo make
```

15.3 Adding Your Own Code to OpenSees

Software: GNU Compiler Collection (GCC)

Step 1. Open a new terminal $(Ctrl + Alt + T)$

Step 2. Checkout the DEVELOPER folder by typing in the following command

```
svncosvn://opensees.berkeley.edu/usr/local/svn/OpenSees/trunk/DEVELOPER
```

Step 3. Type the command ***cd DEVELOPER***

Step 4. Type the command ***cd material***

Step 5. Type the command ***cd cpp***

Step 6. Try to run the example by typing the following command

```
/home/'username'/bin/OpenSees example1.tcl
```

Note:
** Run this from the path of the cpp folder*
** Run OpenSees from where it exists on your computer*
** replace 'username' by the username of your computer*

The build fails with the following error:

```
WARNING could not create uniaxialMaterial ElasticPPcpp while executing
"uniaxialMaterial ElasticPPcpp 1 3000 0.001"
(file "example1.tcl" line 22)
```

Step 7. Type the command ***make*** or ***sudo make***

Step 8. Repeat the command ***/home/'username'/bin/OpenSees example1.tcl***

If the error persists,

Step 9. Type the command export `LD_LIBRARY_PATH=.`

Step 10. Repeat the command `/home/'username'/bin/OpenSees` *example1.tcl*

Section 16

FAQ

Q.Which version of Visual Studio should be used for compiling OpenSees?
A. Initial version of OpenSees framework is built on Visual Studio 2005. The best recommended versions of Visual Studio to run OpenSees are MS Visual Studio 2008 and MS Visual Studio 2010.

Q.Which version of Tcl/Tk should be used for compiling OpenSees?
A. The current stable version of Tcl/Tk mentioned on the OpenSees website is Version 8.5.16. OpenSees is compatible with 32 bit (x86) version of Tcl/Tk. Incompatibility is observed in 64 bit version. We recommend using 32 bit version of Tcl/Tk.

Q.Where should the Tcl/Tk be installed?
A. Tcl/Tk should be installed in `C:\ProgramFiles\Tcl`. This folder does not exist and it has to be created. If you choose the default, the program will be installed in `C:\Tcl` and it will not be identified by OpenSees.

Q.Where do I find the Microsoft Visual Studio solution file (OpenSees.sln) for OpenSees?
A. The solution file (OpenSees.sln) is found in `OpenSees\Win32`.

Q.The solution file I have shows 8 / 9 / 10 on its icon. What does this mean?
A. The numbers found on the solution file icons represent the version of visual studio that they have been built in.

Q.The solution file I have is built on a previous version of Visual Studio. What do I do?
A. Double click on the solution file (OpenSees.sln) to open it in microsoft visual studio. If Visual Studio detects that the project or file was created in an earlier version of Visual Studio, the **Visual Studio Conversion Wizard** opens. Follow the on-screen instructions to complete the wizard. Creating a backup of the solution is Optional.

Q.I don't see a Solution Explorer toolbar when I open the project using OpenSees.sln file. I see a class view instead. Where is the Solution Explorer?
A. Sometimes, visual studio may display a class view by default. You may choose to view the

solution explorer instead. Choose **View** from menu bar and select **Solution Explorer** to display it on the left panel of Microsoft Visual Studio window.
Q.I don't see a **Build** menu on my menu bar. What do I do?
A. This is because your visual studio is configured to Basic Settings by default. Enable expert settings. From the menu bar, click on **Tools** ⟶ **Settings** ⟶ **Expert Settings**.

Q.I have built the solution successfully. What next?
A.Click on **Debug** option in the menu bar and select **Start Debugging** or click on the solid green arrowhead (Start Debugging) found on the debugging toolbar or simply hit **F5** on your keyboard to debug and generate the binary file **OpenSees.exe**

Q.My debugging is successful and my opensees command window is now Open. But where do I find the binary file **OpenSees.exe**?
A. OpenSees builds the binary file in `OpenSees\Win32\bin` folder.

Q.I tried running some example scripts using the binary file generated by MS Visual Studio after debugging. It takes too long to run them. Why?
A. This is because your binary was generated using **Debug** version. Generate a release version by selecting the **Release** option in the solution configurations dropdown menu. This option is found on the debug toolbar.

Q.What is the difference between **Debug** and **Release** versions?
A. The main difference between the **Debug** version and the **Release** version is the debug configuration generates debug information for your program and disables compiler optimizations while the release configuration enables compiler optimizations. The debug binaries tend to be huge as they carry debug information with them which is very useful if the program crashes. However, the release binary is small and contains no crash information. If you want to save time, use the release version of the binary as it is ideal for end users. If your program crashes and you want some clues about it and you don't care about the time it takes to run, use debug version.

Q.My program takes a very long time to run. Is there a way I can save time?
A. The speed of the build is directly proportional to the hardware and memory of your computer. It may take very long time to run if you have a slow processor and low RAM. OpenSees framework consists of many projects as seen in the solution explorer when you open the *OpenSees.sln* in MS Visual Studio. To save the time while building the solution or debugging, you may run only the project of your interest, or the project in which the error occurred and was solved. To run a single project, Right Click on the <Project Name> ⟶ Project Only ⟶ Build Only <Project Name>.
For example: Right click on **SIFBuilder** ⟶ Project Only ⟶ Build Only **SIFBuilder**
Note: If it takes unusually long time to run, you may check the version compatibility, library paths and preprocessor tags. One of these might be missing or input wrongly. In linux distributions, an error in the makefile may prolong the processing time and lead to a system freeze / crash.

Q.The process of debugging is often frustrating. Are there any tips to increase the efficiency?
A. Don't be intimidated upon seeing a lot of red lines in the output window. **READ THE ERROR CAREFULLY!**. Many times, the error may be a syntax error due to a missing *bracket*, misspelt *Keyword(s)*, etc. This can be annoying especially when the program is very long. MS Visual studio identifies such typos by underlining them using *error markers*. If you hover your mouse pointer over this curvy line, the detailed information about the error appears at the tooltip.
Utilize the interactive capabilities provided by MS Visual Studio. Right click on the error and jump to the code line in the program containing error by selecting '*Go To Location*'. This will sift through all the files in the project and pop out the one that contains the error. This feature can be repeated for many instances to navigate throughout the framework. **Q.**I can't find the source and header files for example ElasticPPcpp. Where are they located?
A. The files are located in `OpenSees\DEVELOPER\material\cpp`. Also, the folder DEVELOPER contains example files for element, integrator, material, recorder and system.
Q.The test file, *example1.tcl* gives an error while running. What do I do?
A. Check the following:
1. Has the DLL for newly generated element/material been moved into the same folder?
The newly generated DLL file has to be moved into the same folder as example file to be recognized and linked to the script.
2. Does the *example1.tcl* file contain the line of code with newly added element / material?
Tcl commands are generated for newly written piece of code. The command line should be included in the test file.
eg:
uniaxialMaterial ElasticPPcpp 1 3000 0.001
If an error is generated, check if the command line is commented out. Remove the comment (#).
eg:
# uniaxialMaterial ElasticPPcpp 1 3000 0.001

REFERENCE

Discovering OpenSees: Surfing the Waves of OpenSees by FMK
◄ **Presentation** ►   ◄ **Video** ►

OpenSees: Compile Under Linux [Debian 7 x32 & 64]*(in Chinese)*

Silvia Mazzoni, Frank McKenna, Michael H. Scott, Gregory L. Fenves, et al., "**Open System for Earthquake Engineering Simulation User Command-Language Manual**". Pacific Earthquake Engineering Research Centre, University of California Berkeley 2009.