

Project #1

Build Server: Operational Concept Document

CSE-681 SOFTWARE MODELLING AND ANALYSIS

Instructor- Dr. Jim Fawcett

AKSHITHA KOGANTI

Contents

Executive Summary	3
Introduction	4
Concept and key architectural idea	5
Application Activities	5
Partition	7
Mock Client.....	7
Mock Repository	7
File Collection	8
Parser.....	8
Build Automation.....	8
Logger	8
Uses	8
Critical Issues	9
References	10
Prototype	11
Console output.....	11
Build logs	13

Executive Summary

This document represents the Operational Concept of Build Server. The most commonly heard dialogue from the developers when the build fails in the production is that “It works on my machine”. In this case the software malfunctions only in a certain environment and the hidden dependencies might not be known to the developer for whom everything runs fine in his local machine. If the project builds and runs on 2 machines, it is more likely that it will run on many other machines than if it has only been tested on the developer’s machine. So even if there is only one developer working on a project, a build server has its value.

Build server is an automated tool that builds the test libraries. The repository sends the build request and the code to the build server. Then the build server will parse the build request and attempt the build. If the build fails, it will send the notification to the client else it will send the libraries, test request to the Test Harness and build log to the repository.

The purpose of this project is to implement a remote build server which will build the code as per the build request. This project will provide the capability to:

- Creates a temporary directory
- Accept the build request and code from the repository
- Saves the code and build request in the temporary directory
- Attempts to build
- If build succeeds send the libraries and test request to Test harness
- If build fails send the notification to the client
- Create a build log and send it to the repository
- Once the process is done remove the code and build request received in the temporary directory

The intended users to use this application are:

- Project Managers - to view the results of the latest build so that they will have a clear view on progress of the project.
- Developers – to free up the local machines and to get back to it only when the build failed notification is received.
- QA team – to check performance of the system with the logs.

Below are some of the critical issues associated with the project, and solutions are discussed in detail in further sections

- Integration between the interfaces (Repository and Build Server, Build Server and Test harness)
- If the ongoing project is very large and complex, the last build might fail and we may need to submit a new DLL. So, the question is – do we need to rebuild the entire system again?
- How is the performance and storage in case of server overload?
- Can there be instances of security breaches
- In case of build failure, how soon should the temporary directory be removed after creating the build logs?
- If the build request received is of incorrect format or a corrupted file? (<Tested>testcase1.zip</Tested>).

The document further provides discussions on inter-module interactions and explains the flow of activities with diagrams. The document lists some use cases of the application and suggests a few solutions for critical issues within the application.

Introduction

Build server is an automated tool that builds test libraries. The build server starts with a clean slate, so no unapproved configurations or artifacts are present. Since the code is pulled from the repository, only committed code will end up in the release version. It increases predictability by enforcing source control and making it possible to

identify the issues. It also notifies clients quickly if there are conflicts or missing dependencies.

Concept and key architectural idea

In software engineering, continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day. The main aim of the build server is to identify the missing dependencies or any build errors during integration. In this application, we will be using C# language with .NET framework 4.6 and Visual Studio 2017. The build, test requests will be using the xml format to communicate the information with in the interfaces.

Application Activities

In figure1 below shows the activities performed in the build server when a request is received from the repository.

- Repository will send the code files and the request xml to the build server.
- A temporary directory should be created.
- Accept the files and store them in the temporary directory.
- Parse the build request and check which compiler should be used.
- Check if the build logs already exist
- If build log doesn't exist then an entire build is needed.
- If build logs exist then we need to check the status of the last build.
- If last build failed, then only the new DLLs must be build or else whole build is required.
- Attempts build.
- If it is successful then send the test request and libraries to the test harness and execute the test harness.
- If it fails then send a notification to the client about the build failure
- Send the build logs to the repository

- Remove the temporary directory

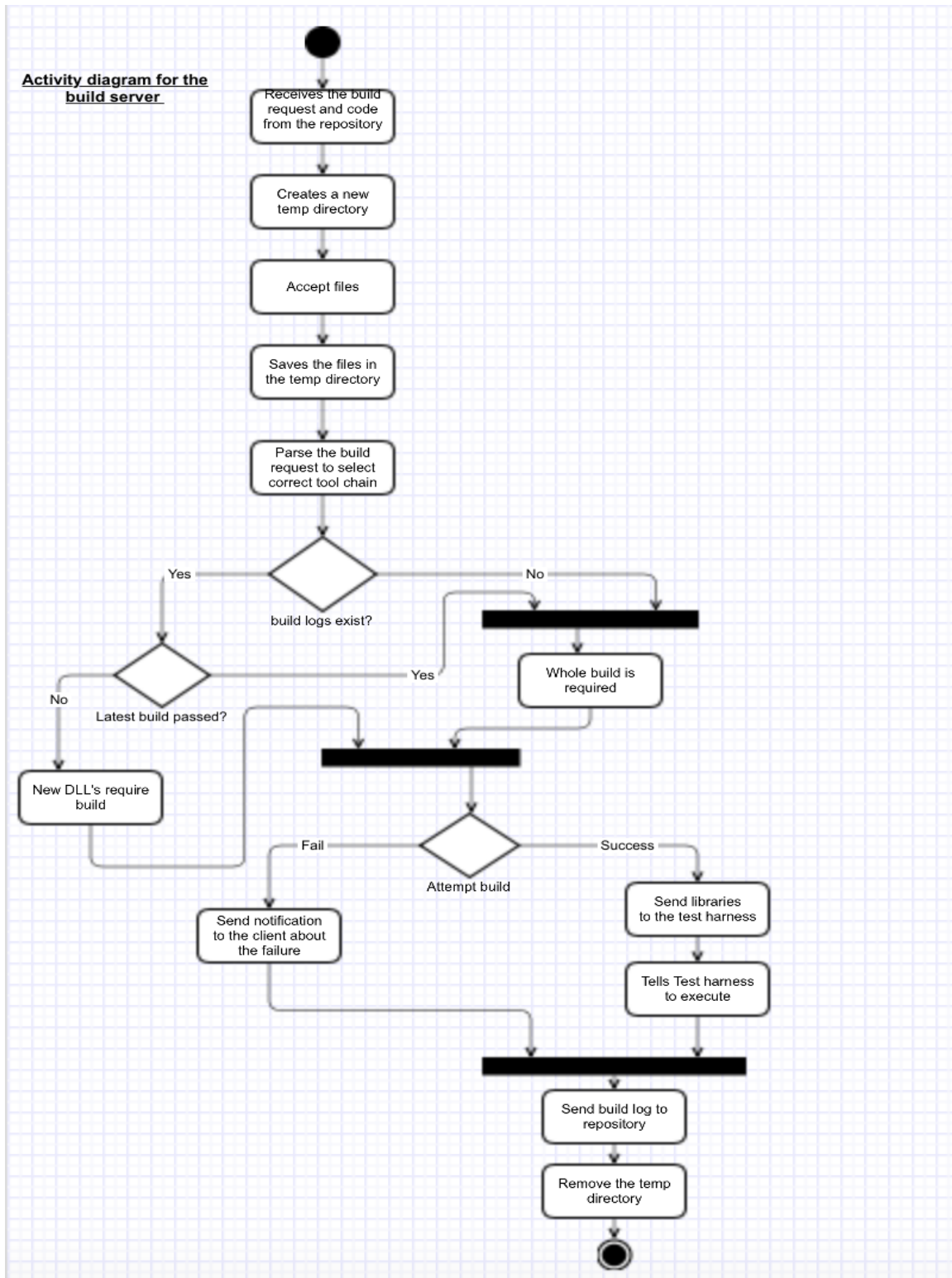


Figure 1: activity diagram for the build server

Partition

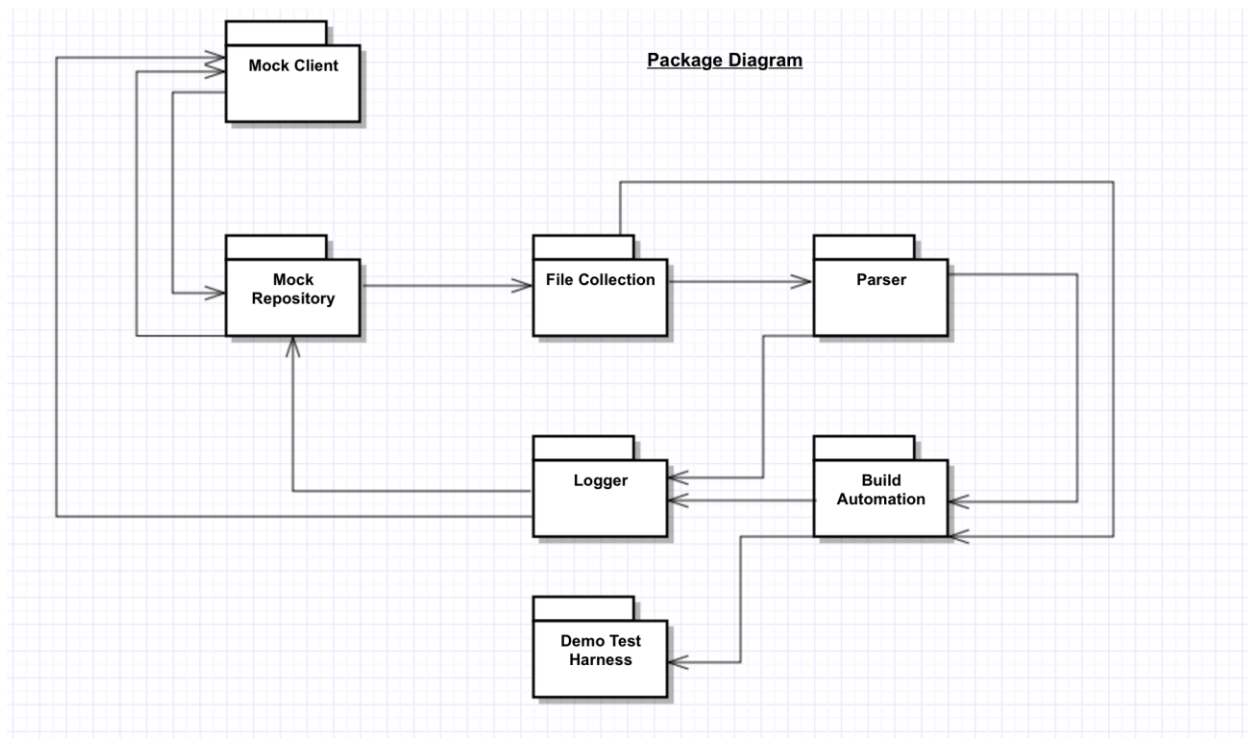


Figure 2: Package diagram

Mock Client

This package handles the users to communicate with the system. It takes the input from the users. Its activities are: sending test request to the repository, sending code to the repository, requesting logs and test results from the repository, Displaying the logs and test results and Displaying the notifications.

Mock Repository

This package is responsible to store all the files and request received from the Demo client, logger in corresponding directories. It should be able to parse the test request and send the build request and code files to the file collection module.

File Collection

This package handles accepting all the files sent by the repository, creating a temporary directory and storing the files received in it. It should send the request to parser and files to build automation. It should delete the temporary directory when a message is sent from the logger so that server usage can be freed up.

Parser

This package handles accepting the xml request from the file collection and to parse it. It should be able to choose which are the files mentioned in the test cases and inform Build automation about the compiler to use.

Build Automation

This package handles accepting the files from file collection and the information from the parser about the compiler. It should try the build. If the build is success then it should save the logs in the logger and send the test request, libraries to the test harness module. If the build fails then it should save the build logs in the logger.

Logger

This package is responsible to send the build logs to the Mock repository. It should be able to send the notification about the build to the client.

Uses

- In large and complex projects might have a lengthy build time. A build server can speed the development process by freeing up the resources on developer's local

machines and send the notifications to the developer about the success and failure of the build.

- Time needed for rebuild after fixing the defect is decreased. If there is a defect (in build) in part of a large system then there is no need to rebuild the entire system once it is fixed.
- If the project has considerable number of automated test cases which may need more time to run, the build server can offload this task from the developer's workflow. Else, the developer might choose not to run these tests very often which in turn increases the feedback cycle between bug being introduced and detected

Critical Issues

- If a defect in part of a large system (where build time very large) is found during the build then once it is fixed, it takes a lot of time to rebuild the entire system again. Sometimes if the schedule is very tight it might not be possible to rebuild the entire system again.
Solution: By using the dynamic link libraries we can just fix the defect and store the new DLL in the repository. During the next build, the new library will be implicitly loaded.
- Data storage of the temporary directories in the Build server? – if there are 20 clients accessing the build server at the same time and each client is having a code files of more than 10GB then will it be possible to let the access to the other clients? It will affect the performance or speed of the system.
Solution: if the server is busy then it can send the notification to the client with new requests to wait for some time or it should be able to bring up another server.
- Integration between the interfaces (Repository and Build Server, Build Server and Test harness)
Solution: Use the same XML Parser in the required modules.

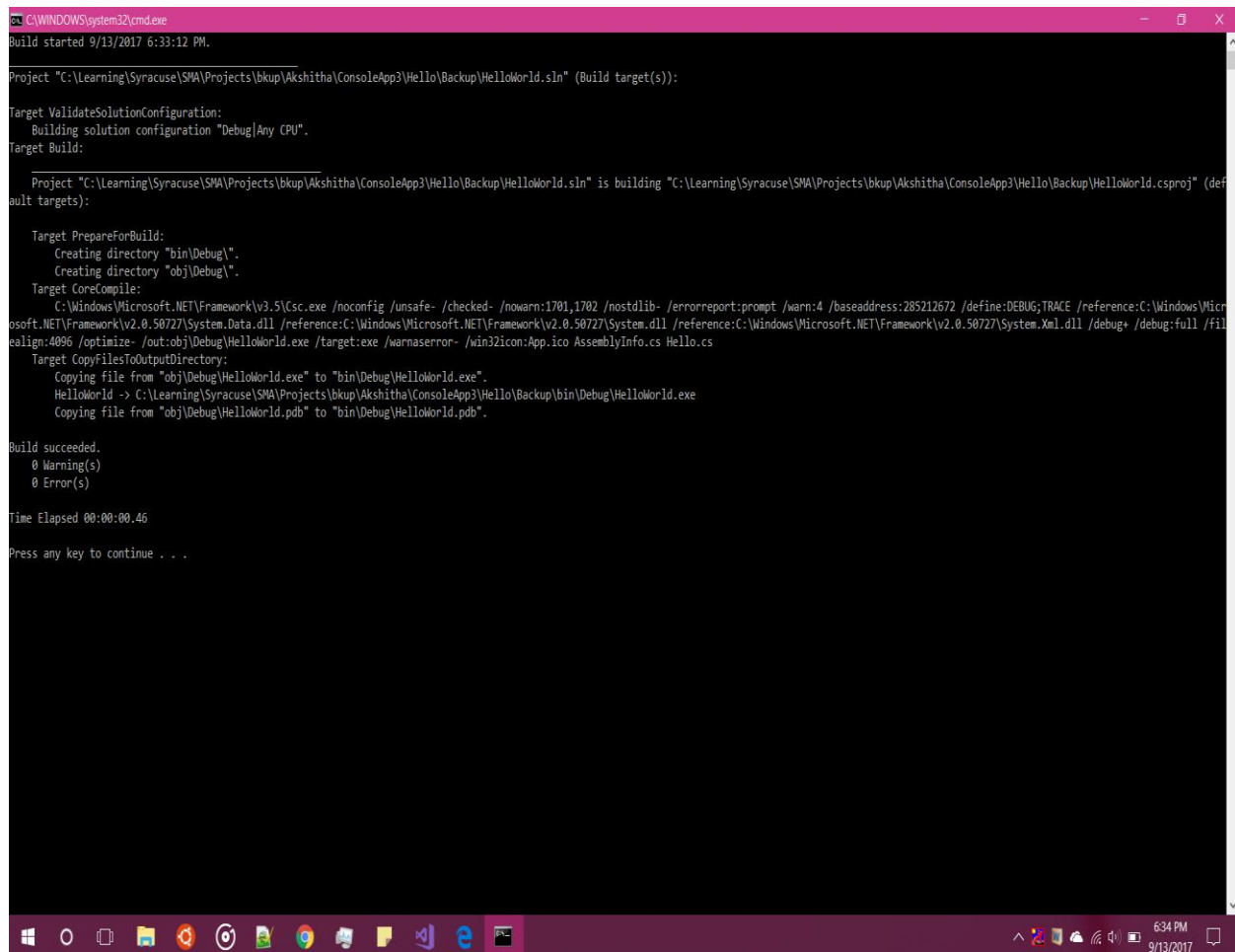
- Security – if client1 request build logs and test results for the test request submitted by the client 2 then it is a security breach. Breaches of security could include damages to the system, loss or theft of data, and compromise of data integrity.
Solution: We must allow only authorized users to access our repository. Multiple levels of accesses for managers and developers depending on their needs should be developed. Users should only be allowed to access as much functionality as they are authorized.
- Should the temporary directory be removed immediately after creating the build logs in the build failure case?
Solution: Before starting the whole build, build server will check if the previous build failed. If the previous build failed then it will just rebuild the new DLL. So, the temporary directory can be deleted once the build log is sent to the repository.
- If the build request received is of incorrect format or a corrupted file?
(<Tested>testcase1.zip</Tested>)
Solution: before accepting the files check the extension of the files, if they are with code, library files or the logs then only accept the files.

References

- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2017.htm>
- https://en.wikipedia.org/wiki/Continuous_integration
- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm>
- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project4-F2017.htm>
- <http://ecs.syr.edu/faculty/fawcett/handouts/webpages/blogOCD.htm>

Above log provides the information regarding the following:

- Build start time
- Files that are getting build
- Build configuration
- Compiler information
- Build status
- Execution time



```
C:\WINDOWS\system32\cmd.exe
Build started 9/13/2017 6:33:12 PM.

Project "C:\Learning\Syracuse\SMA\Projects\bkup\Akshitha\ConsoleApp3\Hello\Backup\HelloWorld.sln" (Build target(s)):

Target ValidateSolutionConfiguration:
  Building solution configuration "Debug|Any CPU".
Target Build:

  Project "C:\Learning\Syracuse\SMA\Projects\bkup\Akshitha\ConsoleApp3\Hello\Backup\HelloWorld.sln" is building "C:\Learning\Syracuse\SMA\Projects\bkup\Akshitha\ConsoleApp3\Hello\Backup\HelloWorld.csproj" (default targets):

  Target PrepareForBuild:
    Creating directory "bin\Debug\".
    Creating directory "obj\Debug\".
  Target CoreCompile:
    C:\Windows\Microsoft.NET\Framework\v3.5\Csc.exe /noconfig /unsafe- /checked- /nowarn:1701,1702 /nostdlib- /errorreport:prompt /warn:4 /baseaddress:285212672 /define:DEBUG;TRACE /reference:C:\Windows\Microsoft.NET\Framework\v2.0.50727\System.Data.dll /reference:C:\Windows\Microsoft.NET\Framework\v2.0.50727\System.dll /reference:C:\Windows\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll /debug+ /debug:full /filealign:4096 /optimize- /out:obj\Debug\HelloWorld.exe /target:exe /warnaserror- /win32icon:App.ico AssemblyInfo.cs Hello.cs
  Target CopyFilesToOutputDirectory:
    Copying file from "obj\Debug\HelloWorld.exe" to "bin\Debug\HelloWorld.exe".
    HelloWorld -> C:\Learning\Syracuse\SMA\Projects\bkup\Akshitha\ConsoleApp3\Hello\Backup\bin\Debug\HelloWorld.exe
    Copying file from "obj\Debug\HelloWorld.pdb" to "bin\Debug\HelloWorld.pdb".

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:00.46

Press any key to continue . . .
```

Build logs

The below is the build log for the failure case, in which you can notice that there are 3 errors and the status of the build.

```

Build started 9/13/2017 7:17:32 AM.
Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\CSharpDemos.sln" (Build target(s)):
Target ValidateSolutionConfiguration:
  Building solution configuration "Debug|Any CPU".
Target Build:
  C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\CSharpDemos.sln.metaproj : error MSB3202: The project file "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\BasicClass\BasicClass.csproj" was not found.
Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\CSharpDemos.sln" is building "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\ClassRelationships\ClassRelationships.csproj" (default targets):
Target ResolveProjectReferences:
  Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\ClassRelationships\ClassRelationships.csproj" is building "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\DemoExtensions\DemoExtensions.csproj" (default targets):
Target GenerateTargetFrameworkMonikerAttribute:
  Skipping target "GenerateTargetFrameworkMonikerAttribute" because all output files are up-to-date with respect to the input files.
Target CoreCompile:
  C:\Windows\Microsoft.NET\Framework\v4.0.30319\Csc.exe /noconfig /nowarn:1701,1702 /nostdlib+ /platform:anycpu32bitpreferred /errorreport:prompt /warn:4 /define:DEBUG;TRACE /highentropyva+ /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\Microsoft.CSharp.dll" /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\mscorlib.dll" /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\System.Core.dll" /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\System.Data.dll" /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\System.Data.Entity.dll" /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\System.Data.Linq.dll" /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\System.Net.Http.dll" /reference:"C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\System.Xml.Linq.dll" /debug+ /debug:full /filealign:512 /optimize- /out:obj\Debug\DemoExtensions.exe /subsystemversion:6.00 /target:exe /utf8output DemoExtensions.cs Properties\AssemblyInfo.cs "C:\Users\Komal\AppData\Local\Temp\NETFramework,Version=v4.5.2,AssemblyAttributes.cs"
  DemoExtensions.cs(34,7): error CS1041: Identifier expected; 'static' is a keyword
  DemoExtensions.cs(34,14): error CS1518: Expected class, delegate, enum, interface, or struct
  Done building target "CoreCompile" in project "DemoExtensions.csproj" -- FAILED.
  Done building project "DemoExtensions.csproj" -- FAILED.
Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\ClassRelationships\ClassRelationships.csproj" is building "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\DemoExtensions\DemoExtensions.csproj" (default targets):
Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\CSharpDemos.sln" is building "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\Generics\Generics.csproj" (default targets):
Target ResolveProjectReferences:
  Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\Generics\Generics.csproj" is building "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\DemoExtensions\DemoExtensions.csproj" (default targets):
Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\CSharpDemos.sln" is building "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\EnumerableTypes\EnumerableTypes.csproj" (default targets):
Target ResolveProjectReferences:
  Project "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\EnumerableTypes\EnumerableTypes.csproj" is building "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\DemoExtensions\DemoExtensions.csproj" (default targets):
Done building target "Build" in project "CSharpDemos.sln" -- FAILED.
Done building project "CSharpDemos.sln" -- FAILED.
Build FAILED.
C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\CSharpDemos.sln.metaproj : error MSB3202: The project file "C:\Users\Komal\Documents\SMA\CSharpDemos\CSharpDemos\BasicClass\BasicClass.csproj" was not found.
DemoExtensions.cs(34,7): error CS1041: Identifier expected; 'static' is a keyword
DemoExtensions.cs(34,14): error CS1518: Expected class, delegate, enum, interface, or struct
0 Warning(s)
3 Error(s)

```