



# SERVER SIDE CUSTOMIZATION for BOM CREATION and RULE HANDLER REGISTRATION in PLM TEAMCENTER-12.0

<sup>1</sup>Mr. Mohammad Mubasheruddin, <sup>2</sup>Mr. Sayyad Shafik,

<sup>1</sup>M.Tech in Mechanical Engineering, <sup>2</sup>Assistant Professor,

<sup>1,2</sup>Department of Mechanical Engineering,

<sup>1,2</sup>MPGI School of Engineering, Nanded, Maharashtra, India

**Abstract:** In present days of increasing software development environment sometimes it turns into challenging to mold the Product Lifecycle Management software as per requirement of clients. Most of the time user requires some simplified applications for its business solution and simplified process for their organization. To solve this problem server side customization is necessary means customize the server side by using Integration Tool Kit. Integration Tool Kit (ITK) is a Siemens PLM Software Tool that is used to integrate third-party or user-developed applications with Teamcenter. ITK is a low level API's built by Siemens to access Teamcenter Database. It is used for programming within Teamcenter server for customizing extension points, workflow handlers, server exits or standalone executables. Server Layer can be customized by using C and C++ languages. My aim is to create utility through ITK for creating BOM by reading data in csv and I have also registered custom Rule Handler through ITK for validation of named reference as per business requirements. For this work I have used Teamcenter 12 along with Visual Studio Enterprise 17 for server side customization. For customization of server layer according to client's requirement, I have used ITK reference guide provided by Siemens which contains all ITK API's. By using Visual Studio 17, I have written the code of utility and registered custom rule handler by referring ITK API reference guide. Utility has extension .exe so I have executed it directly in Teamcenter server after building the project and BOM is created. Handler has extension .dll so I have registered it in Teamcenter server after building the project and implemented it on a workflow to check business requirement for validation of named reference i.e. physical file is attached on dataset or not. If physical file is attached on dataset handler will return EPM\_go and workflow will be initiated else handler will return EPM-nogo and workflow will not be initiated.

**Keywords:** Product Lifecycle Management, Integration Tool Kit, Bill of Materials, Teamcenter 12, Dynamic Library, Application Programming Interface, Comma Separated Values.

## I. INTRODUCTION

Product Lifecycle Management frameworks offer assistance organizations in adapting with the expanding complexity and designing challenges of creating unused items for the worldwide competitive markets. Product Lifecycle Management ought to be recognized from product life-cycle Management (marketing). PLM depicts the designing perspective of a product, from overseeing depictions and properties of a product through its improvement and valuable life; while, PLM alludes to the commercial administration of life of a product within the commerce showcase with regard to costs and deals measures. Product Lifecycle Management is the method of overseeing the whole lifecycle of a product from initiation, through designing plan and make, to benefit and transfer of made products and also PLM manages data from raw material to finished products [1]. Product Lifecycle Management integrates data, people, processes and business systems and provides a product information backbone for companies and its enterprise. Product Lifecycle Management tool has wider scope and there are various types of Product Lifecycle Management tools that integrate the data according to company needs. Companies have decided which PLM software is better but now days Siemens PLM has covered 70% of global competitive markets [2]. Today's point of view Teamcenter has covered 70% of the global market. There are lot of Product Lifecycle Management tools available in the market such as Siemens Teamcenter, PTC Windchill, Dassault ENOVIA, SAP PLM, Oracle Agile, Aras PLM, Arena PLM, Fuse PLM, Bamboo Rose, Autodesk Vault, Autodesk Fusion Lifecycle but Siemens Teamcenter is very popular among all of this due its flexibility and user friendly interface and also lot of Out of the Box (OTB) are available in the Teamcenter.

## II. PLM CUSTOMIZATION

### 2.1 Need of PLM

Product Lifecycle Management can be utilized to build yield with consistent assets, to expand incomes or to diminish the assets used to deliver a steady yield. This all assists with improving the reality. Product Lifecycle Management encourages association to accomplish this through: Productivity upgrades, Improving advancement for new business objects (products), Decreased expenses, Increment efficiency, improved nature of business objects (products).

Integration of data is the need of global market also integrated data consumes the memory so market needs integrate data with least consumable memory. From that concern has discovered the concept of PLM and its tools.

### 2.2 Objective

This research work has developed the solution for creating custom utility and handler in Teamcenter according to our business requirement where we can implement Product Lifecycle Management in any organization.

This work has main objective of creating custom utility and handler in Teamcenter 12 by using Server Side Customization means the task which cannot be performed by default Teamcenter utilities and handlers are performed by these customized utilities and handlers.

### 2.3 Types of Customization

#### 2.3.1 Client Side Customization

The client side customization can be performed by using programming language means Java and various wizards available in Teamcenter. This type of customization can be used for including new Application in Teamcenter, such as Menu bar, Toolbar, User Interface form on client side. By using this type of customization we can easily solve the customer requirement.

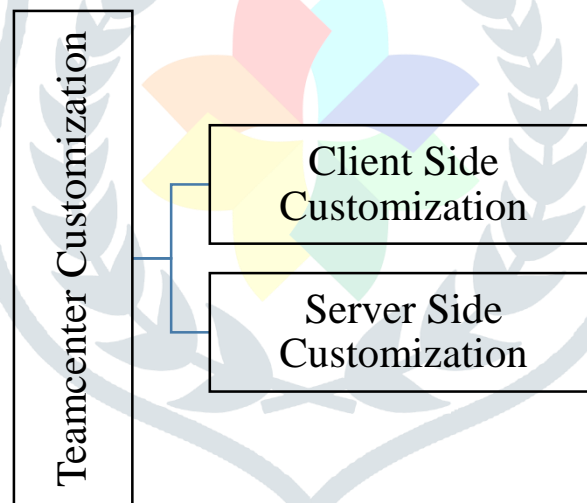


Fig -1: Teamcenter Customization types

#### 2.3.2 Server Side Customization

The server is being customized using Teamcenter API (Application program interface) called as Integrated Tool Kit (ITK), C++. Integrated Tool Kit (ITK) is a set of software tools which you can use to integrate third party or user-developed applications with Teamcenter and we can use this customization for server side issues.

### 2.4 Customization Tools

Customization in Teamcenter can be performed in different ways and we have seen below [3].

#### 2.4.1 Non-programming Customization

Non-programming customization can be performed by using the wizards available in Teamcenter and making the entries in the registry files.

## 2.4.2 Programming Customization

Programming customization can be performed by using programming languages i.e. C, C++, Java. This can be classified into server side and client side customization. Client side is customized using Java language and server side is by using ITK.

## 2.4.3 ITK (Integration Tool Kit)

The Integration Toolkit (ITK) is a set of server-side software tools that you can use to integrate third-party or user-developed applications with Teamcenter. The ITK is a set of C and C++ functions used directly by Teamcenter and UNIGRAPHICS. The Integration Toolkit is a set of software tools that act as a programmatic interface to Teamcenter. It is the means by which both internal and external applications integrate with the Teamcenter. Internal applications are those supplied such as Unigraphics. External applications (third party) are those that you decide to integrate into Teamcenter.

## 2.4.4 Portal Customization

Portal customization is performed by using Java programming. This can be used to add the new application or customize the menu bar and toolbar within Teamcenter engineering application.

## III. RESEARCH METHODOLOGY

For implementation of Product Lifecycle Management in an organization, the main concern is to mold the Teamcenter as per client's requirements. So I have created a utility and handler with implementing custom functionality using Server Side Customization i.e. which cannot be achieved by default utilities and handlers available in Teamcenter. I have made custom project in Visual Studio 17 for utility with .exe extension and name **BOM\_CSV\_UTILITY**. After building the utility I have copied the built path and executed it in Teamcenter server. I have also made custom project in Visual Studio 17 for handler with .dll extension and name **rhlrchknf**. After building the handler I have registered its dll in Teamcenter Server Site and applied it on workflow to validate named reference.

### 3.1 Project Configuration for Utility:

In this customization, projects are built for molding the Teamcenter software as per the business need and the client's need. The steps required for configuration of Visual Studio 17 with Teamcenter for utility are given below in Figure. For this I have created an empty project in Visual Studio and added source file with .cpp extension. Then I have followed the following project setting procedure for utility.

1. Right Click on the project and go to properties and select Configuration Manager and select Release in Active solution configuration and x64 in Active solution platform.

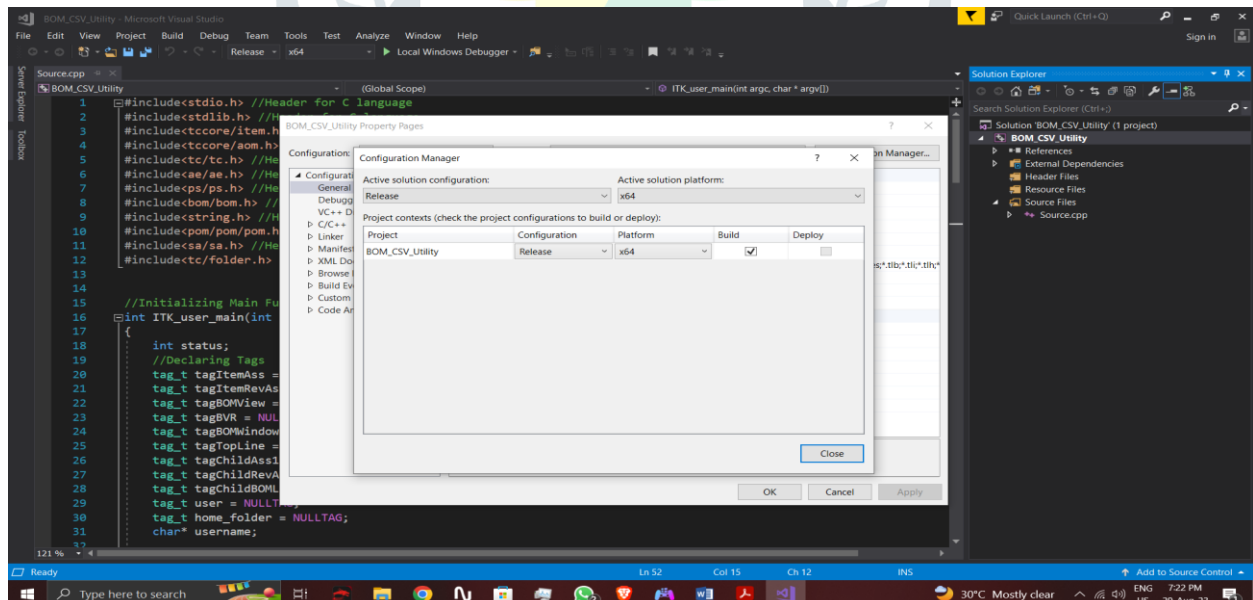


Fig -2: VS Configuration Manager

2. Select Configuration Properties and select Configuration Types as **Application (.exe)** for utility.

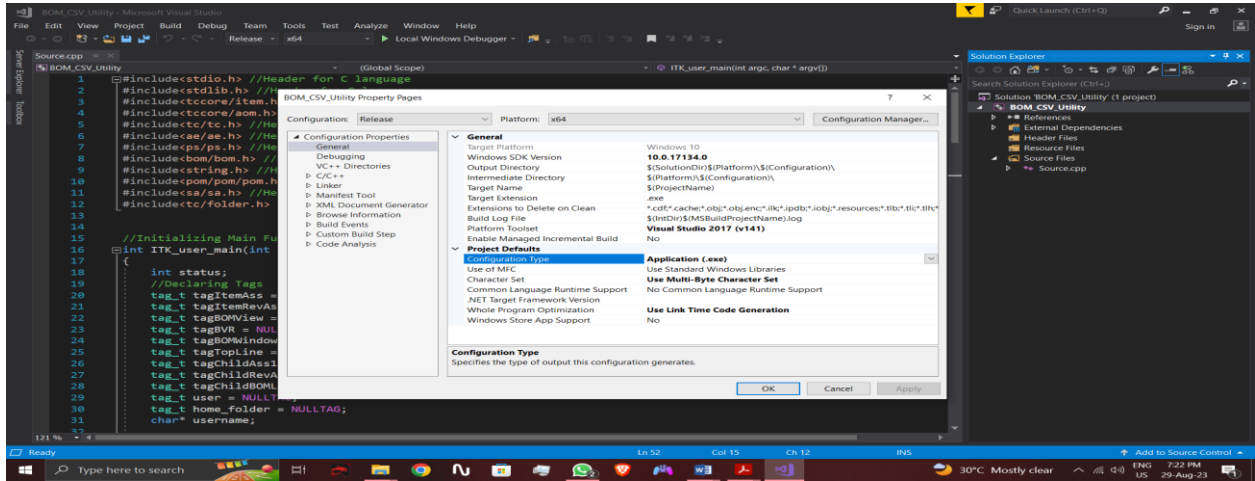


Fig -3: VS Configuration Properties

3. Now select C/C++ and then select General and then select Additional Include Libraries and give the path as follows:

- **E:\Siemens\Teamcenter12\include**
- **E:\Siemens\Teamcenter12\include\_cpp**

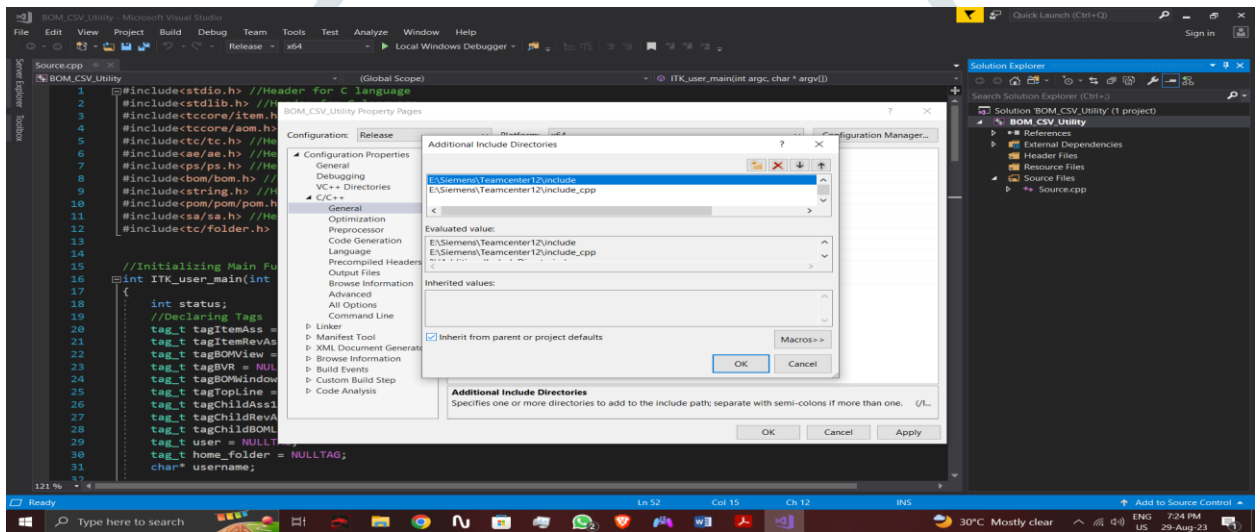


Fig -4: VS C/C++ General Configuration

4. Now select C/C++ and then select Preprocessor and then select Preprocessor Definition and give the path as follows:

- **IPLIB=none**
- **\_CRT\_SECURE\_NO\_WARNINGS**

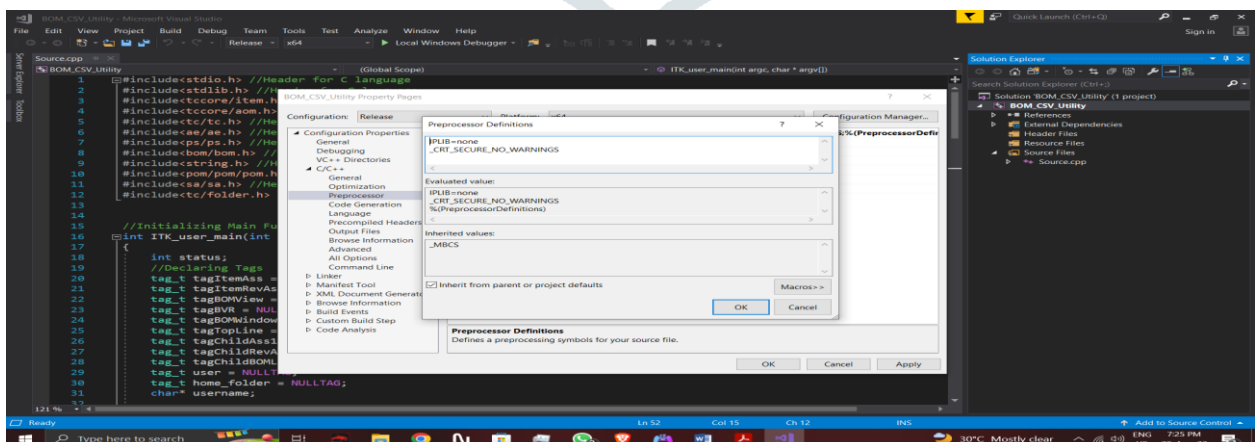


Fig -5: VS C/C++ Preprocessor Configuration

5. Now select Linker folder and select General and select Additional Libraries Directories and give the path as follows:

- E:\Siemens\Teamcenter12\lib

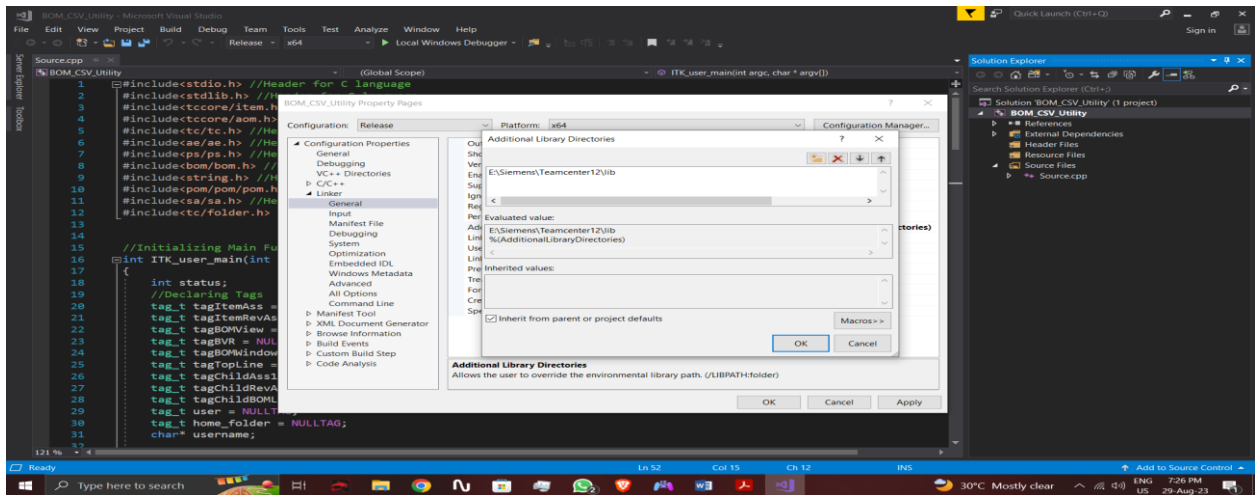


Fig -6: VS Linker General Configuration

6. Now select Linker folder and select Input and select Additional Dependencies and give the path as follows:

- E:\Siemens\Teamcenter12\lib\\*.lib
- E:\Siemens\Teamcenter12\lib\itk\_main.obj

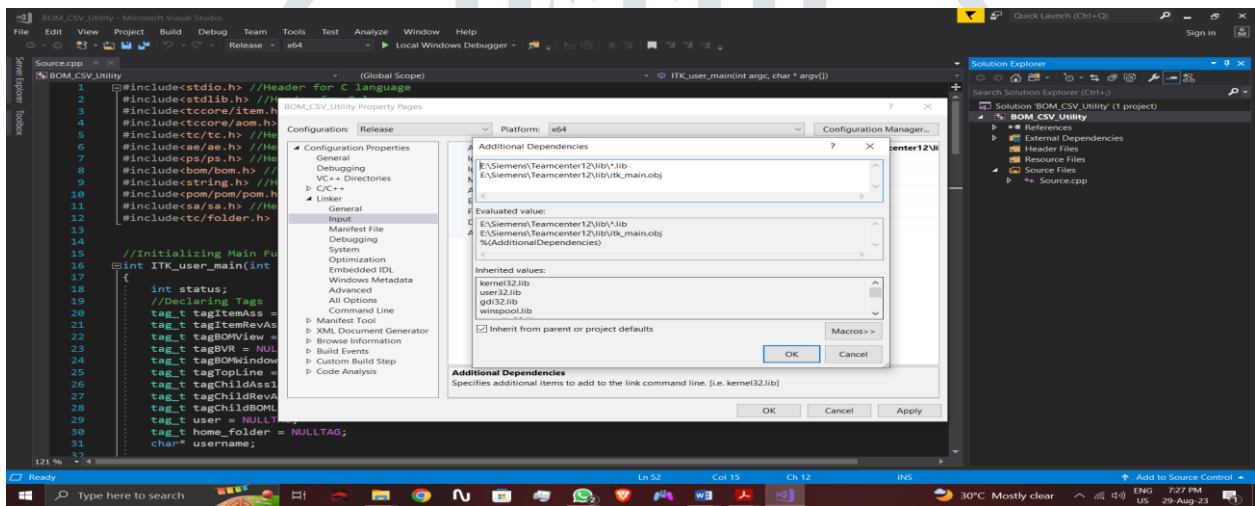


Fig -7: VS Linker Input Configuration

7. After this click Apply and afterwards click OK and Project setting for utility is completed.

### 3.2 Execution of BOM creation through csv file

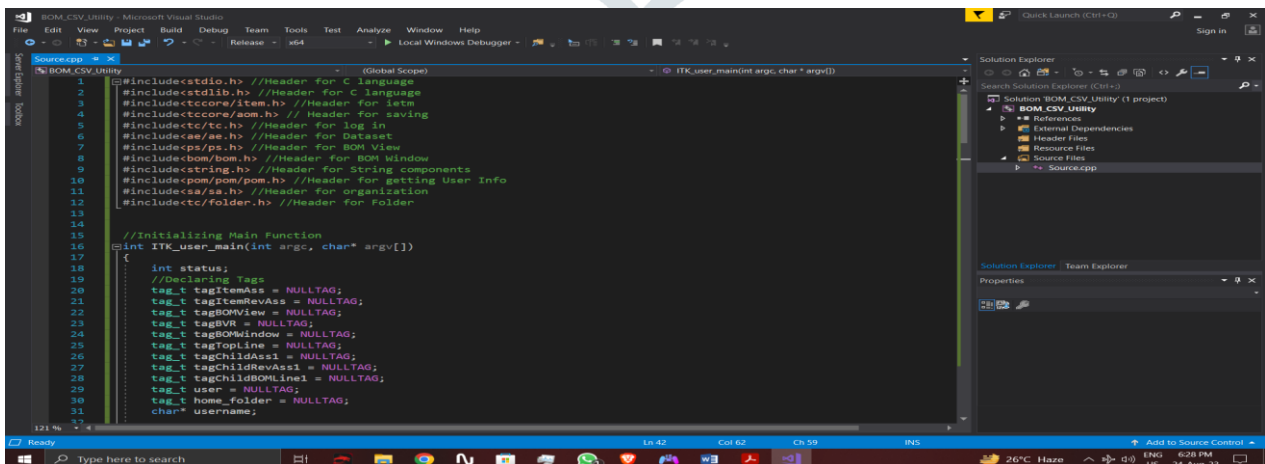


Fig -8: Declaration of Headers and tags

1. After completion of project setting I have started writing code in **source.cpp** file where I have included all headers require to call API's related to BOM and also declared tags require to pass in API's.

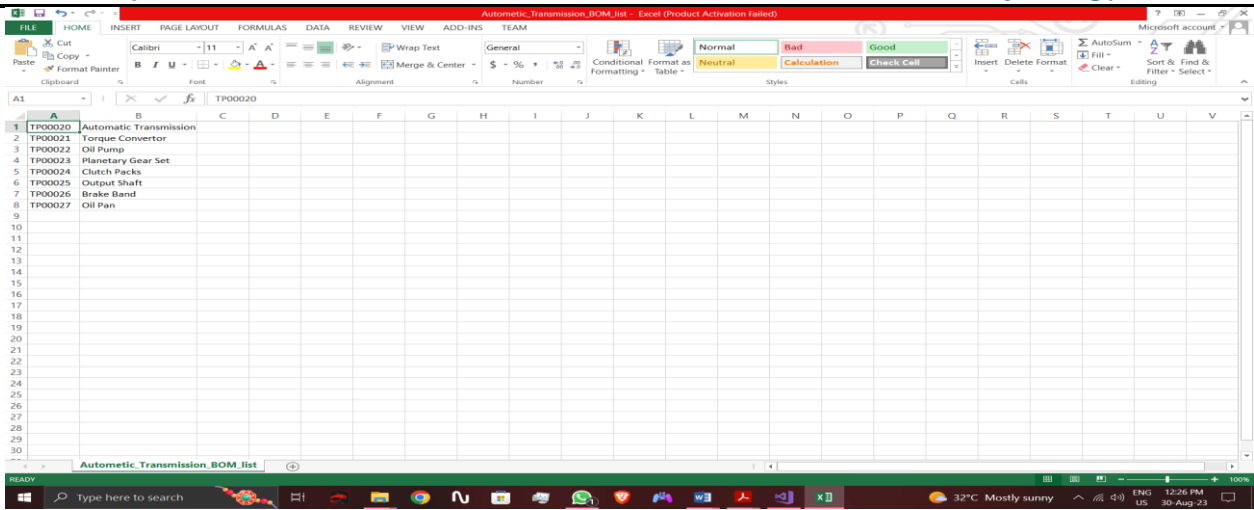


Fig -9: CSV File with BOM Data

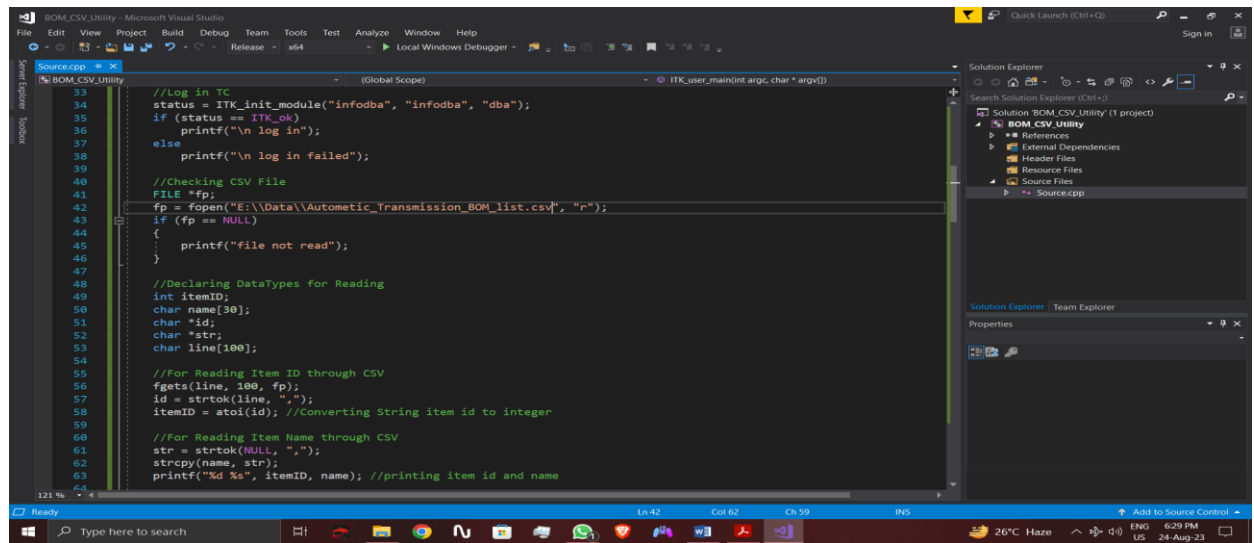


Fig -10: CSV File Read Program

- The CSV file contains BOM data so I have written the code for reading this data through CSV file read program of C language and read the Item ID and Item Name line by line.

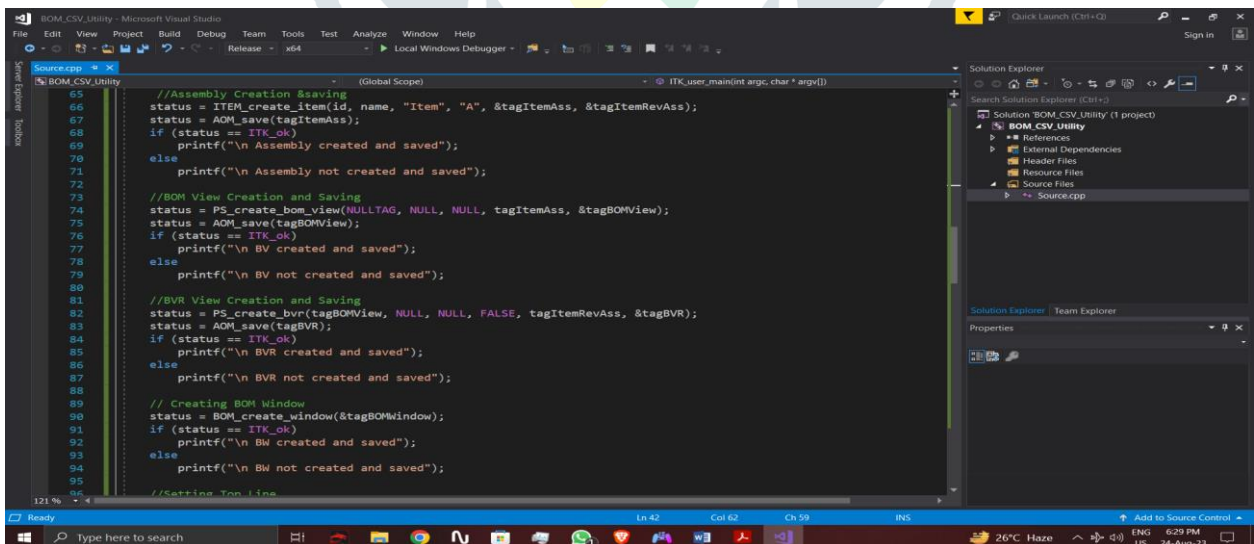


Fig -11: Parent Item Creation

3. I have created BOM View, BOM View Revision, BOM Window and I have set TopBOMLine using API's for Parent Item.

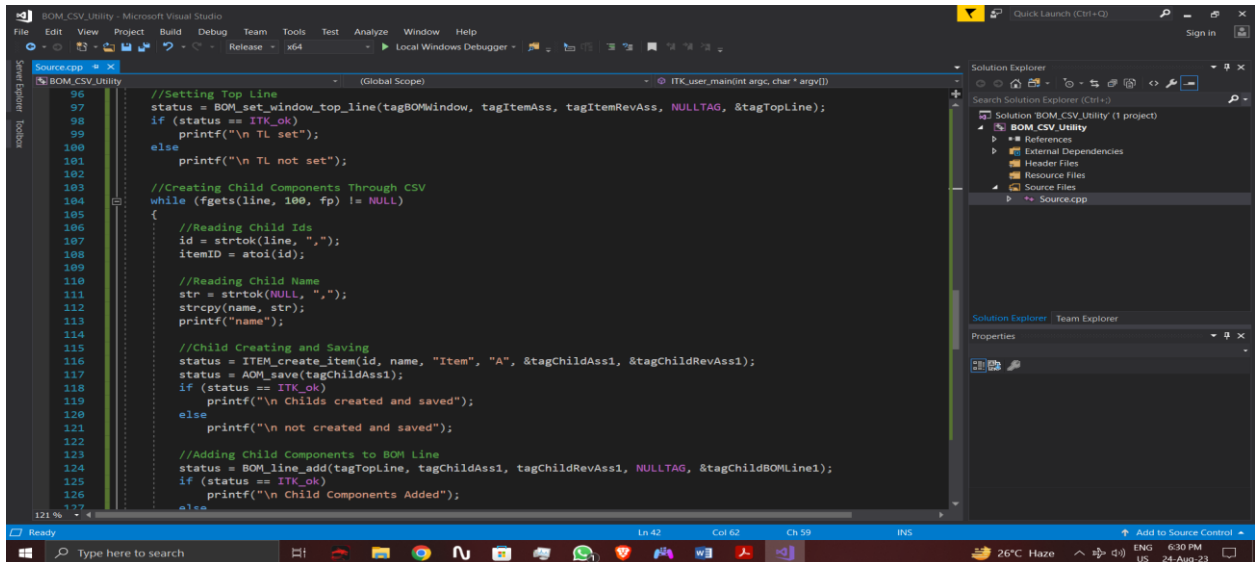


Fig -12: Child Items Creation

4. Created Child Items by using File Read code by reading second line through Item ID and Item Name using While Loop and added Child Components to the BOM Line through API.

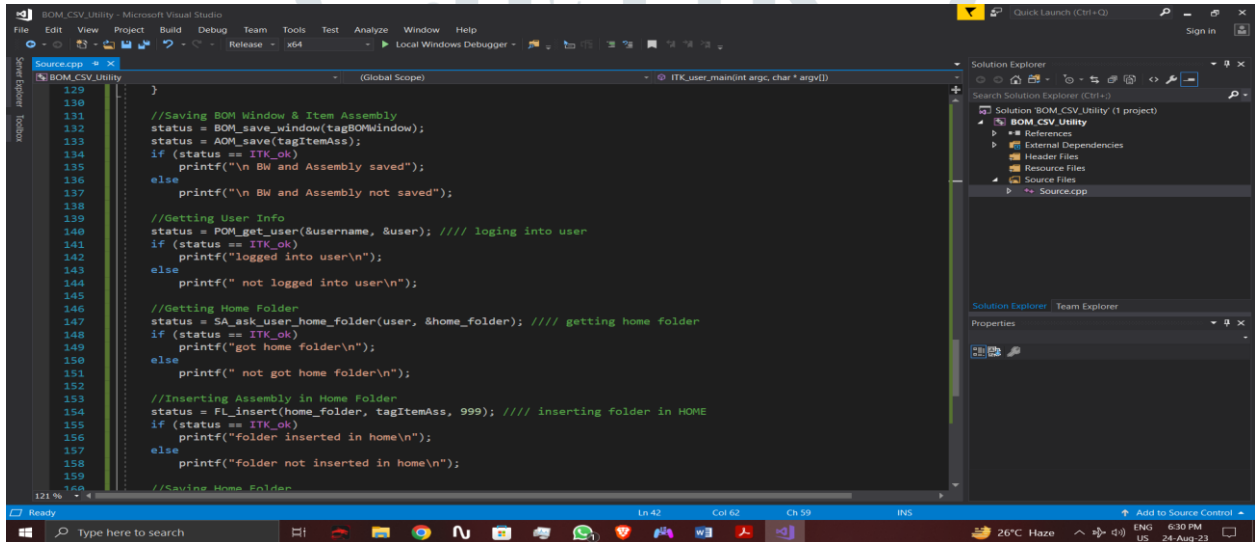


Fig -13: Adding BOM in Home Folder of Teamcenter

5. Saved BOM Window and Item Assembly and added this BOM in Home Folder of Teamcenter.

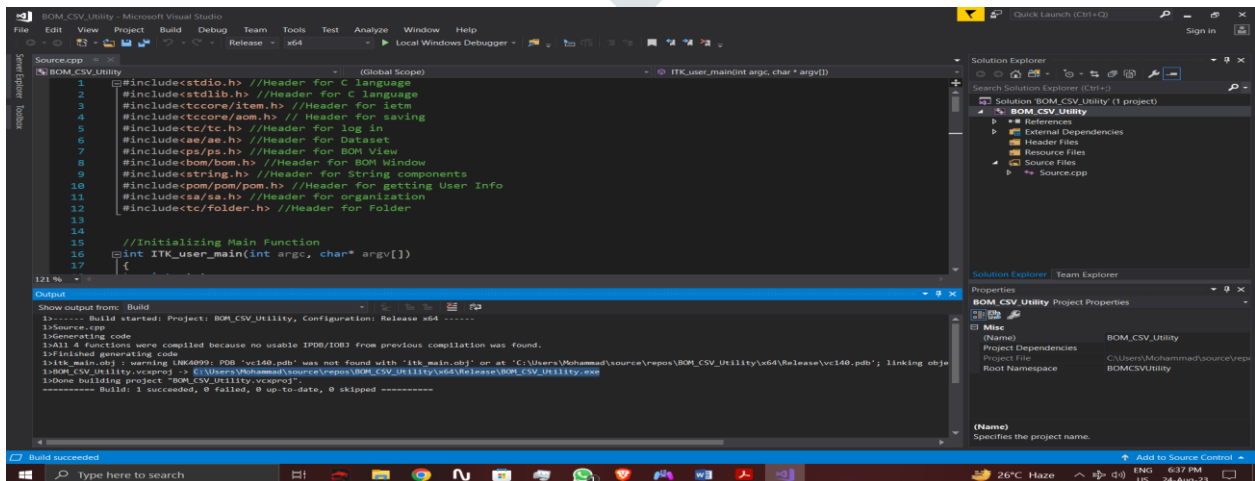


Fig -14: Build the Project

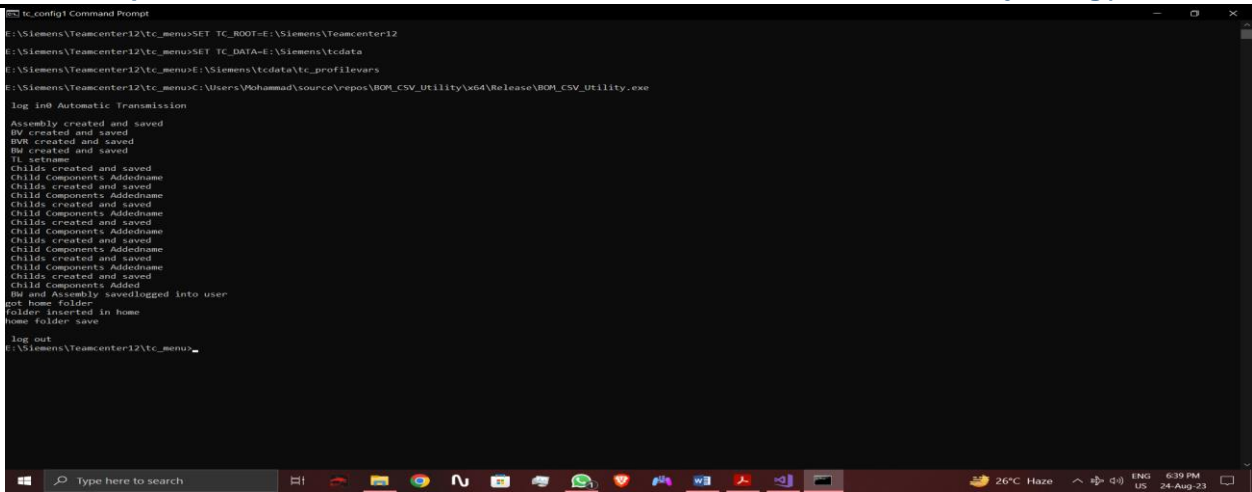


Fig -15: Run the Utility in Teamcenter Server

6. Now I have built the project in Visual Studio and copied the build path of the project. I have opened Teamcenter Server i.e. **tc\_config1 Command Prompt** and pasted the build path to run the utility successfully and BOM is created.

### 3.3 Project Configuration for Handler

The steps required for configuration of Visual Studio 17 with Teamcenter for handler are given below in Figure. For this I have created an empty project in Visual Studio and added source file with .cpp extension. Then I have followed the following project setting procedure for handlers.

1. Right Click on the project and go to properties and select Configuration Manager and select Release in Active solution Configuration and x64 in Active solution platform.

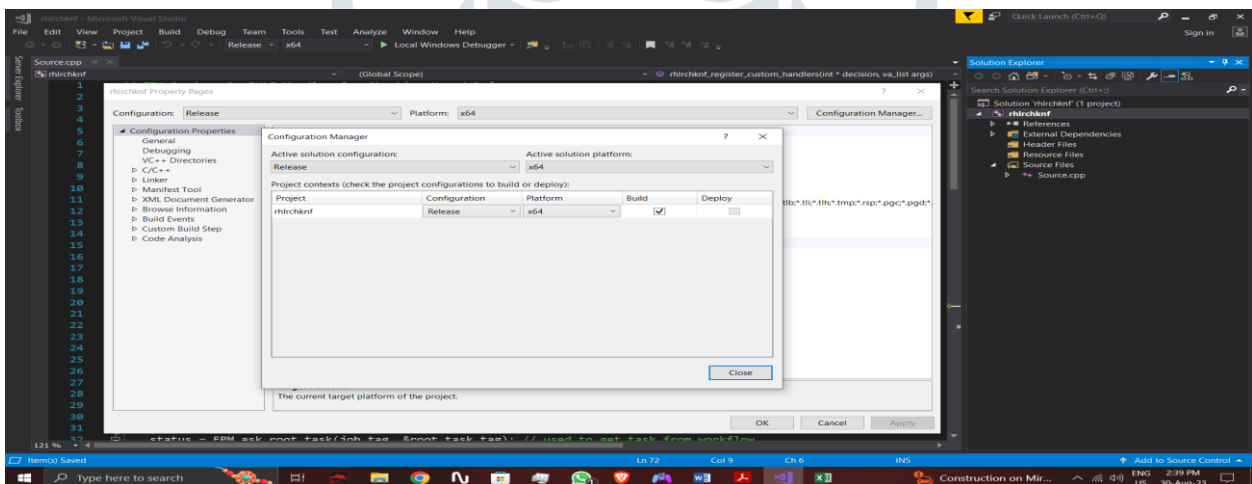


Fig -16: VS Configuration Manager

2. Select Configuration Properties and select Configuration Types as **Dynamic Library (.dll)** for handler.

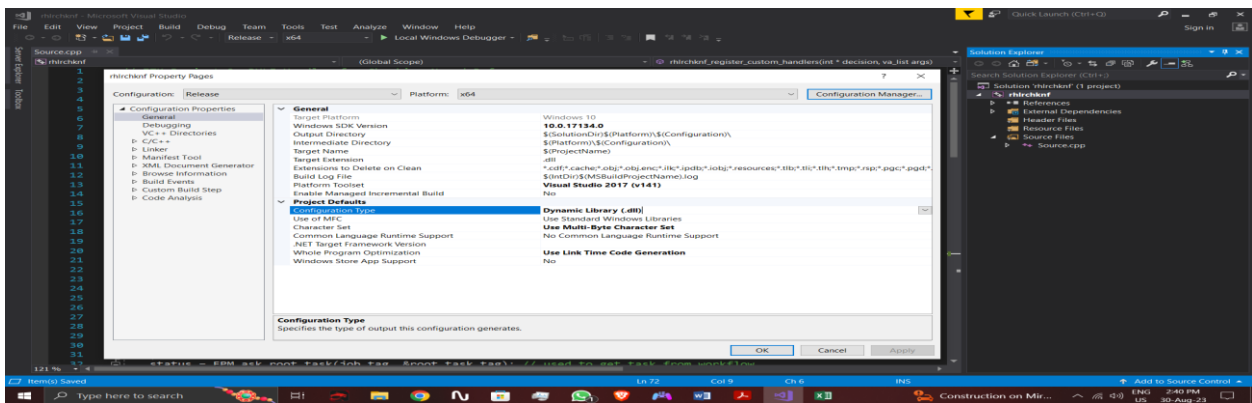


Fig -17: VS Configuration Properties

3. Now select C/C++ and then select General and then select Additional Include Libraries and give the path as follows:

- E:\Siemens\Teamcenter12\include
- E:\Siemens\Teamcenter12\include\_cpp



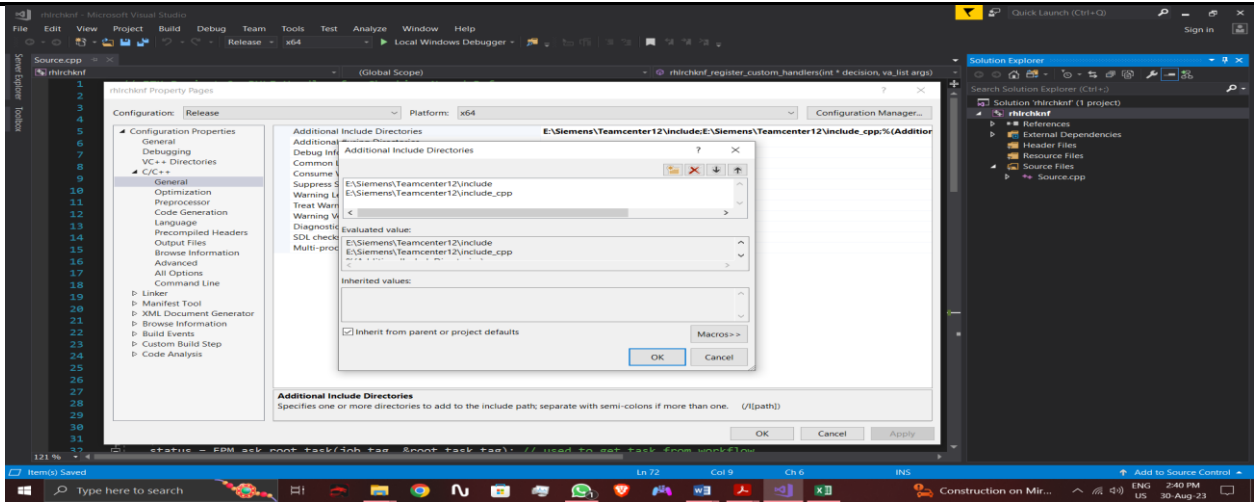


Fig -18: VS C/C++ General Configuration

4. Now select C/C++ and then select Preprocessor and then select Preprocessor Definition and give the path as follows:

- **IPLIB=none**

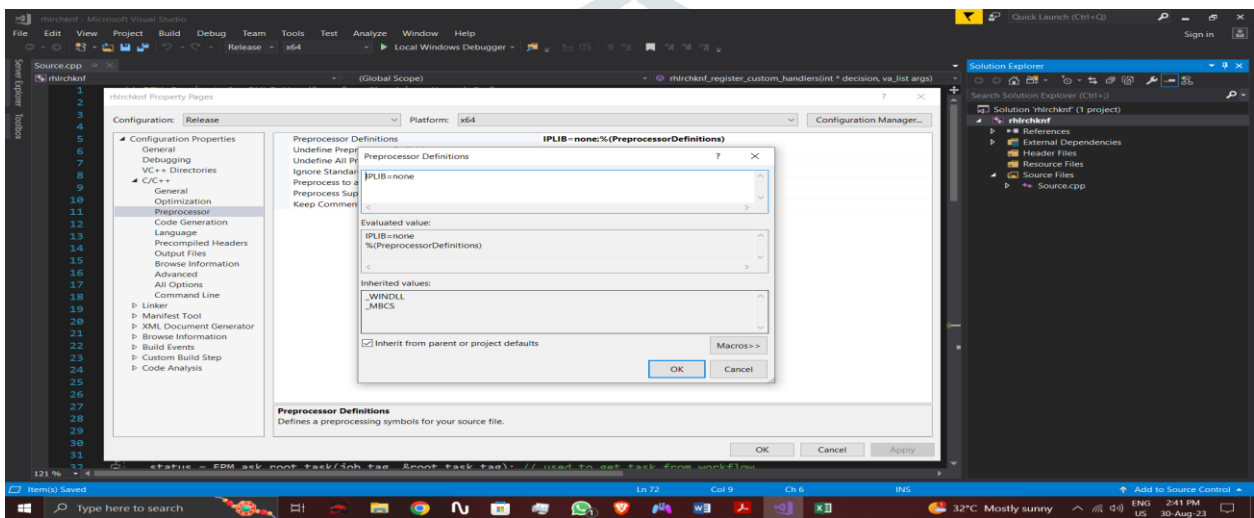


Fig -19: VS C/C++ Preprocessor Configuration

5. Now select Linker and select General and then select Output File and give the path as follows:

- **E:\Siemens\Teamcenter12\bin\rhlrckhkf.dll**

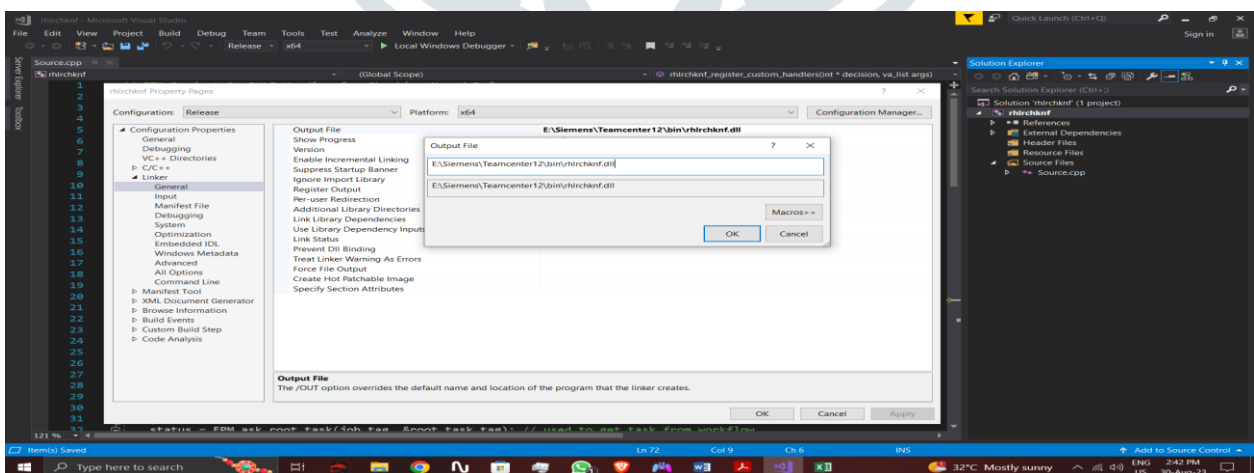


Fig -20: VS Linker General Output File Configuration

6. Now select Linker folder and select General and select Additional Libraries Directories and give the path as follows:

- **E:\Siemens\Teamcenter12\lib**

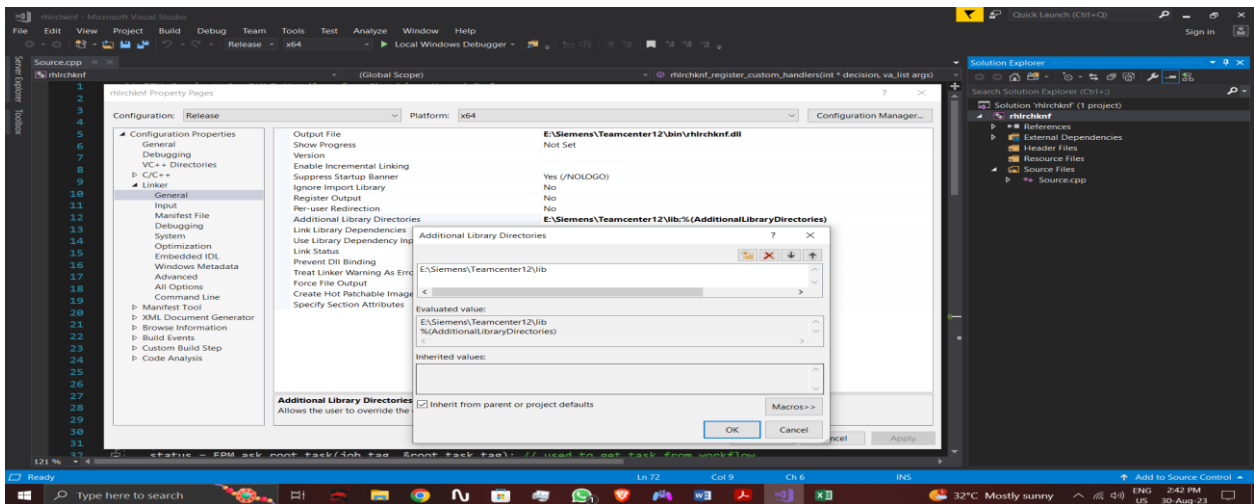


Fig -21: VS Linker General Additional Library Directories Configuration

7. Now select Linker folder and select Input and select Additional Dependencies and give the path as follows:

- **E:\Siemens\Teamcenter12\lib\\*.lib**

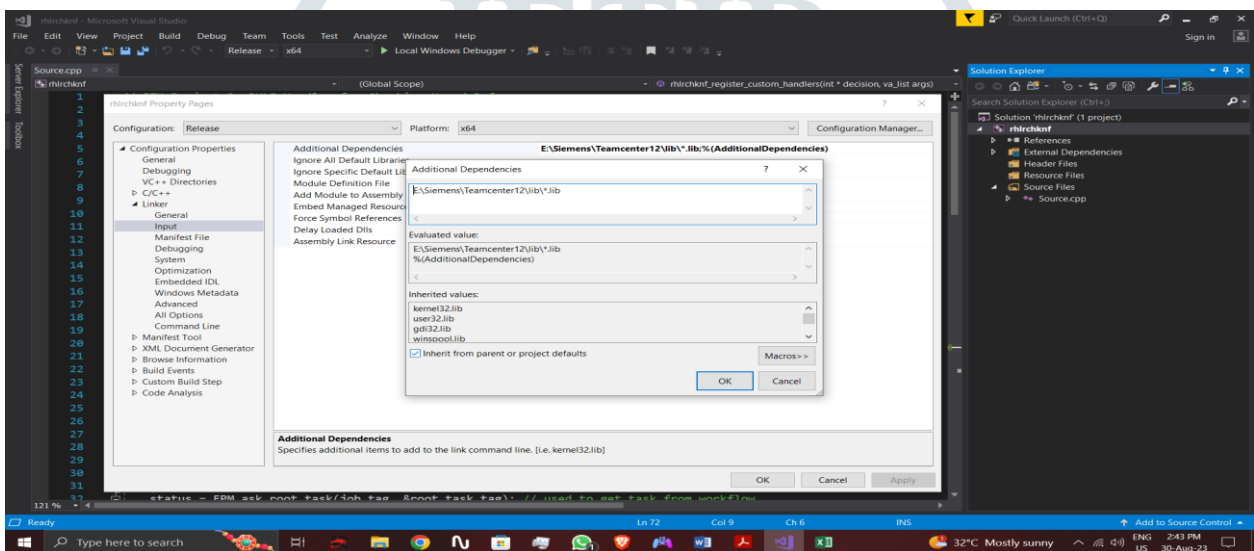


Fig -22: VS Linker Input Configuration

8. After this click Apply and afterwards click OK and Project setting for utility is completed.

### 3.4 Execution of Rule Handler for validation of Named Reference

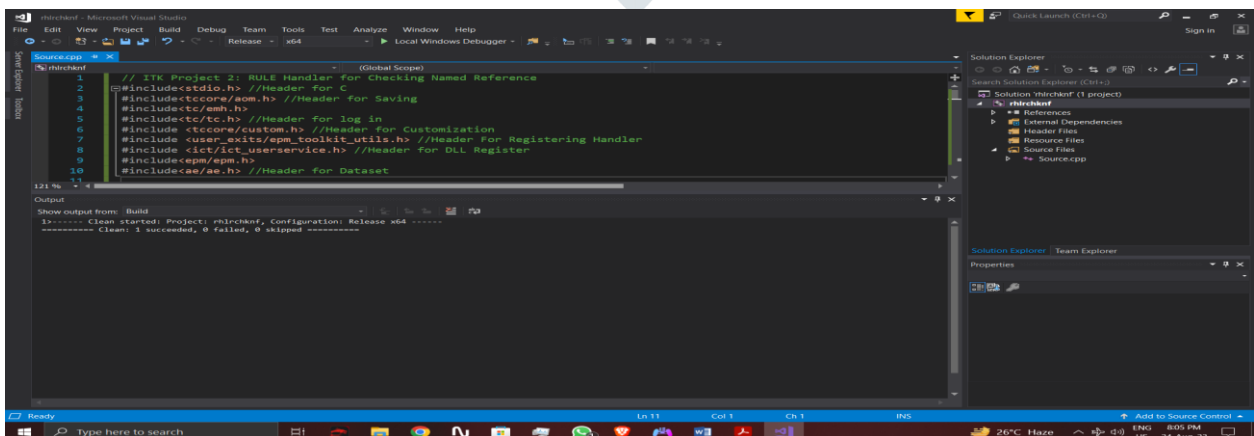
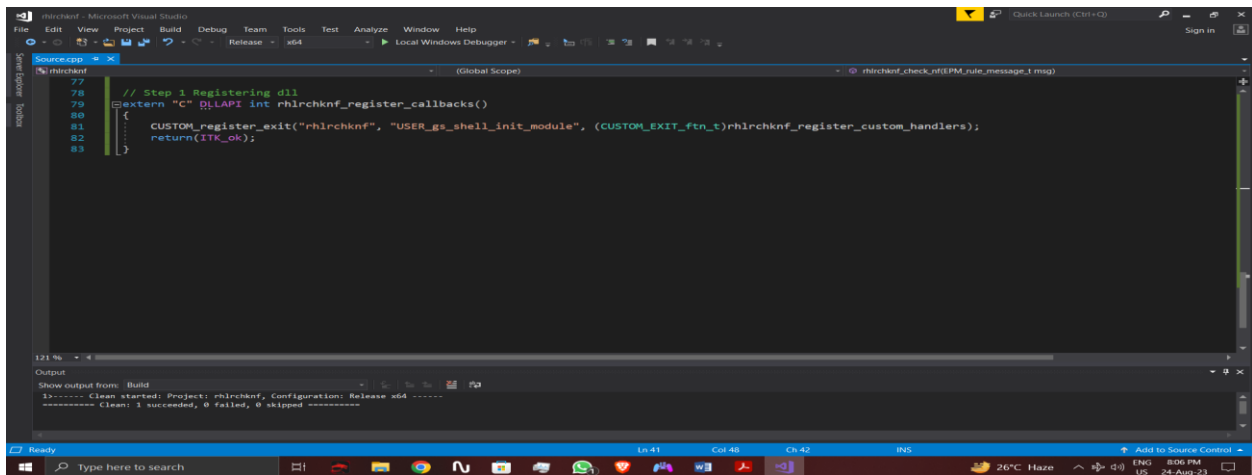


Fig -23: Declaration of Headers and tags

1. After completion of project setting I have started writing code in **source.cpp** file where I have included all headers require to call API's related to Handler and also declared tags require to pass in API's.



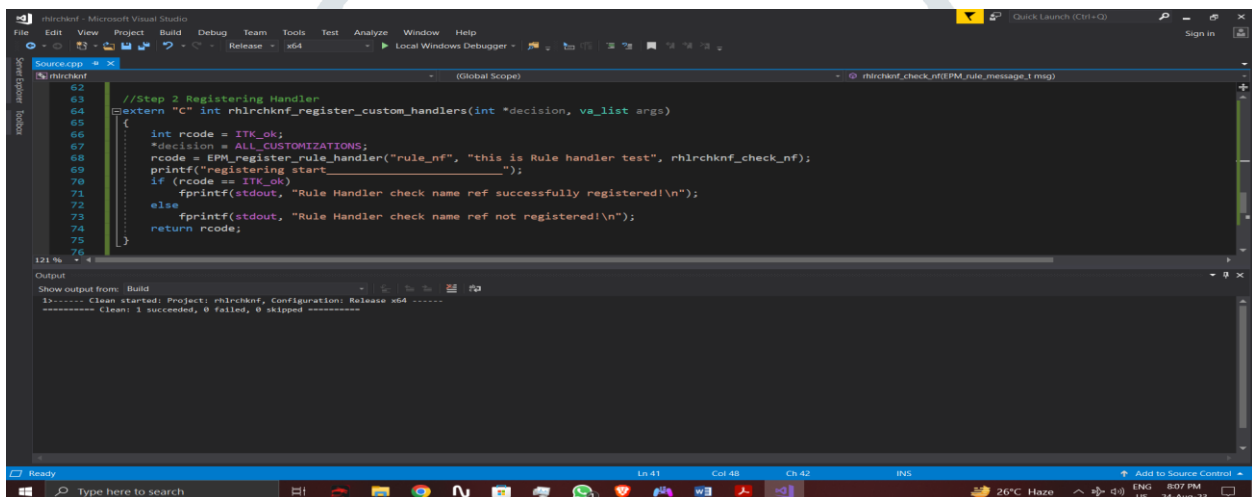
```

77 // Step 1 Registering dll
78
79 extern "C" DLLAPI int rhlrchknf_register_callbacks()
80 {
81     CUSTOM_register_exit("rhlrchknf", "USER_gs_shell_init_module", (CUSTOM_EXIT_FTN_T)rhlrchknf_register_custom_handlers);
82     return(ITK_ok);
83 }

```

Fig -24: DLL Registration Code

2. I have written the code for DLL registration by using API **CUSTOM\_register\_exit** (context, base\_ftn\_name, custom\_ftn) provided by Siemens ITK Function file.



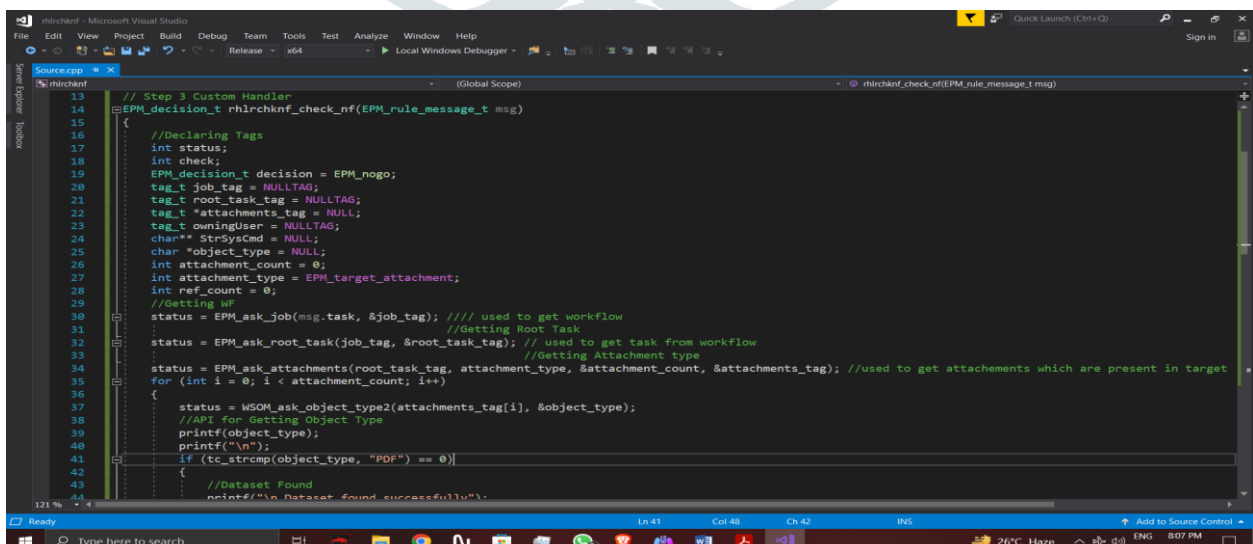
```

62 // Step 2 Registering Handler
63
64 extern "C" int rhlrchknf_register_custom_handlers(int *decision, va_list args)
65 {
66     int rcode = ITK_ok;
67     *decision = ALL_CUSTOMIZATIONS;
68     rcode = EPM_register_rule_handler("rule_nf", "this is Rule handler test", rhlrchknf_check_nf);
69     printf("registering start\n");
70     if (rcode == ITK_ok)
71         fprintf(stdout, "Rule Handler check name ref successfully registered!\n");
72     else
73         fprintf(stdout, "Rule Handler check name ref not registered!\n");
74     return rcode;
75 }
76

```

Fig -25: Rule Handler Registration Code

3. Now I have registered Rule Handler by using API **EPM\_register\_rule\_handler** (handlerName, handlerDescription, functionPointer).



```

13 // Step 3 Custom Handler
14 EPM_decision_t rhlrchknf_check_nf(EPM_rule_message_t msg)
15 {
16     //Declaring Tags
17     int status;
18     int check;
19     EPM_decision_t decision = EPM_nogo;
20     tag_t job_tag = NULLTAG;
21     tag_t root_task_tag = NULLTAG;
22     tag_t *attachments_tag = NULL;
23     tag_t owningUser = NULLTAG;
24     char** StrSysCmd = NULL;
25     char *object_type = NULL;
26     int attachment_count = 0;
27     int attachment_type = EPM_target_attachment;
28     int ref_count = 0;
29     //Getting WF
30     status = EPM_ask_job(msg.task, &job_tag); // used to get workflow
31     //Getting Root Task
32     status = EPM_ask_root_task(job_tag, &root_task_tag); // used to get task from workflow
33     //Getting Attachment type
34     status = EPM_ask_attachments(root_task_tag, attachment_type, &attachment_count, &attachments_tag); //used to get attachments which are present in target
35     for (int i = 0; i < attachment_count; i++)
36     {
37         status = WSOM_ask_object_type2(attachments_tag[i], &object_type);
38         //API for Getting Object Type
39         printf(object_type);
40         printf("\n");
41         if (tc_strcmp(object_type, "PDF") == 0)
42         {
43             //Dataset Found
44             printf("\n Dataset found successfully");

```

Fig -26: Getting Active Task Code

4. Now I have written the code for getting Workflow, Active Task and Attachments.

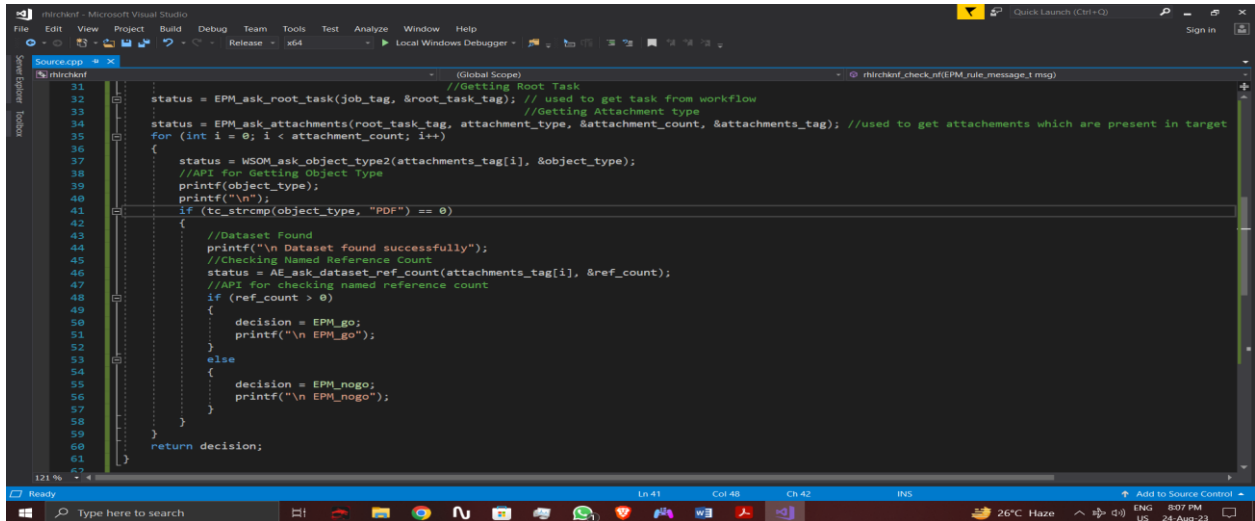


Fig -27: Comparing Object type with Item Revision

5. I have get the Object Type and compared it with Item Revision and if true then I found Dataset. Then checked Named Reference count of Dataset and if count != null then EPM\_go or EPM\_nogo.

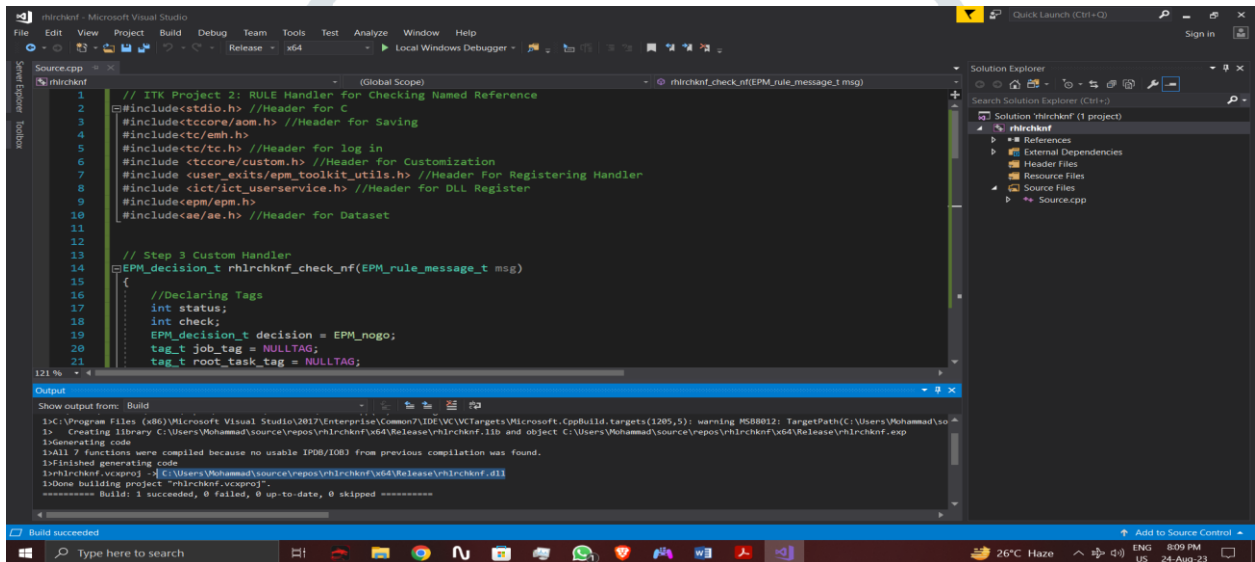


Fig -28: Build the Project

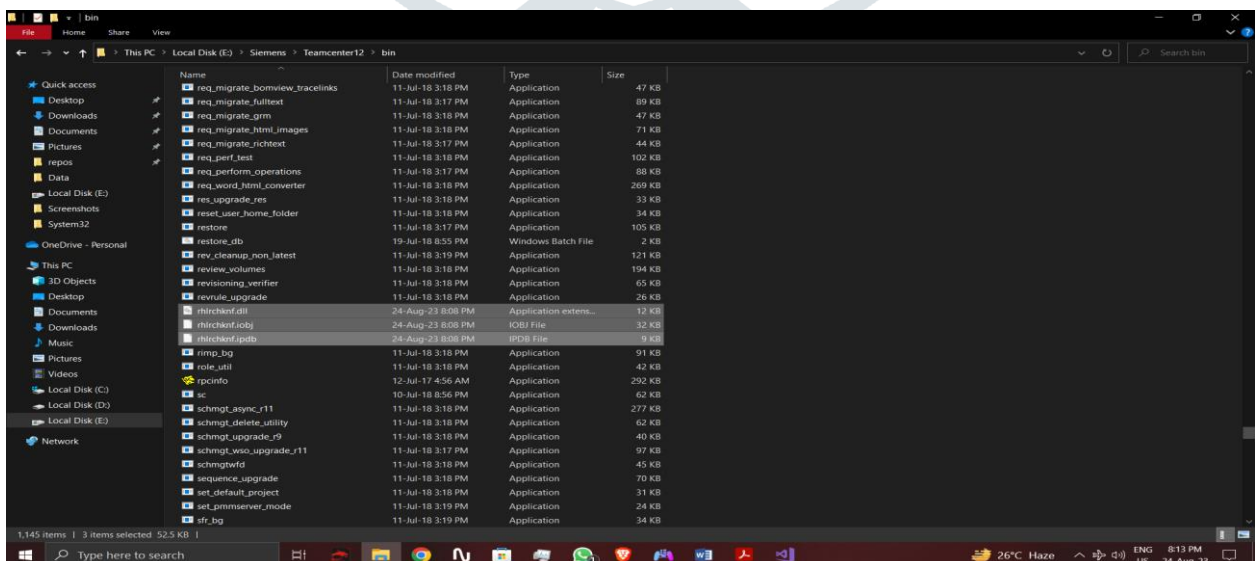


Fig -29: DLL created in bin folder of Teamcenter

- After completion of code build the project in Visual Studio. After building the project **rhlrchknf.dll** extension file is created in bin folder of Teamcenter.

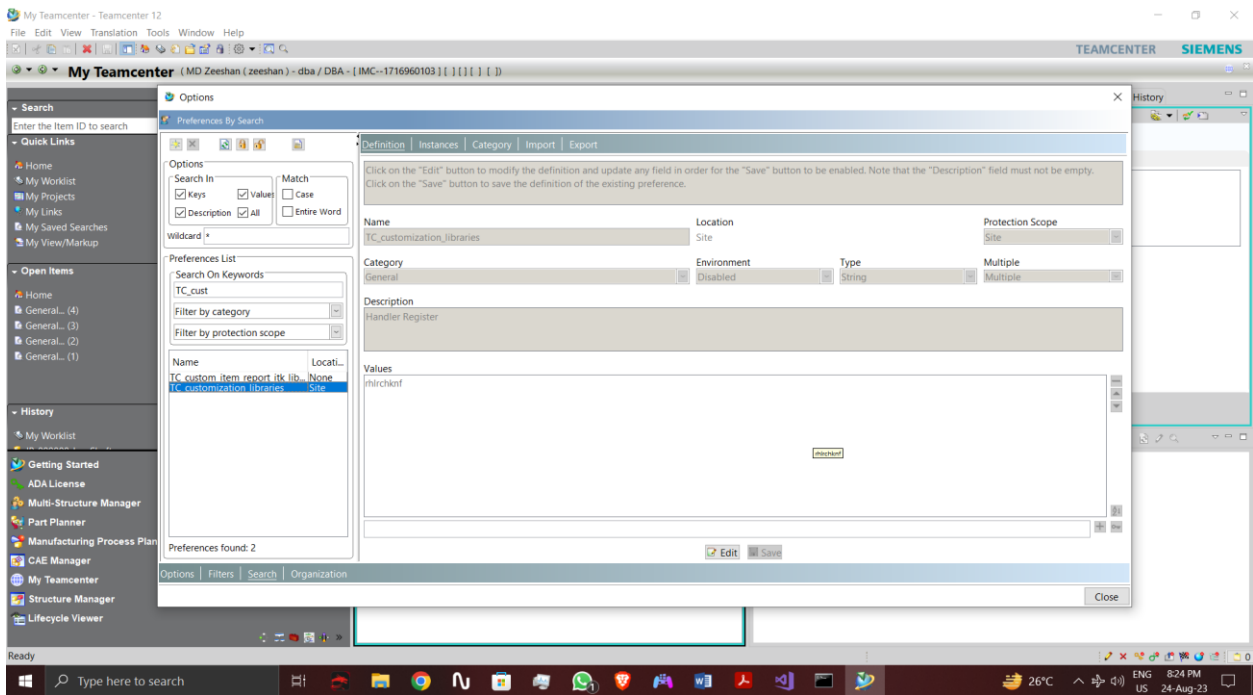


Fig -30: DLL registered in Teamcenter Server

- Copy the project name i.e. **rhlrchknf** without extension and paste it in **TC\_customization\_libraries** for registering dll in Teamcenter server and click on save and log out of Teamcenter.

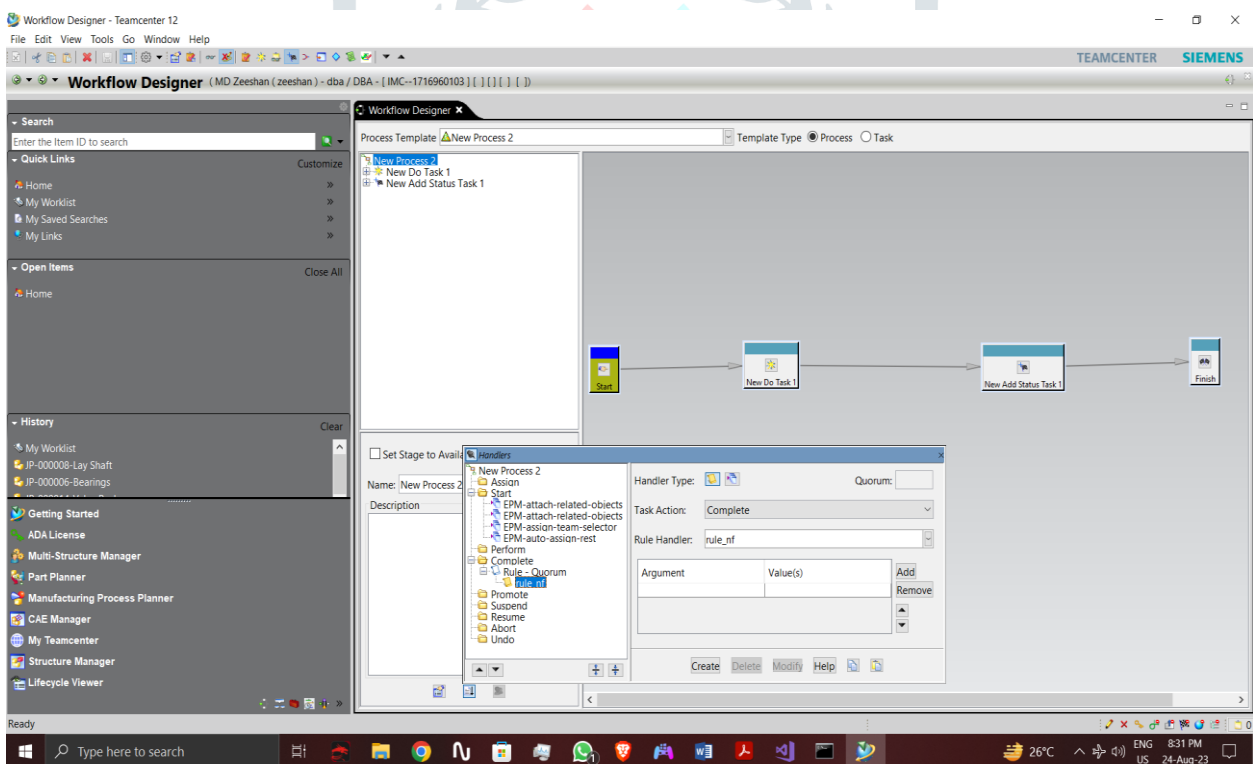


Fig -31: Checking of Custom Rule Handler registered in Teamcenter

- Log in again and our custom rule handler will be registered as shown in above figure and I have initiated it on a workflow to validate the named reference.

By this way I have implemented the BOM creation through CSV file Utility and Registered Custom Rule Handler for Validation of Named Reference in Teamcenter through Server Side Customization.

## IV. RESULTS AND DISCUSSION

### 4.1 Result for BOM creation through CSV file

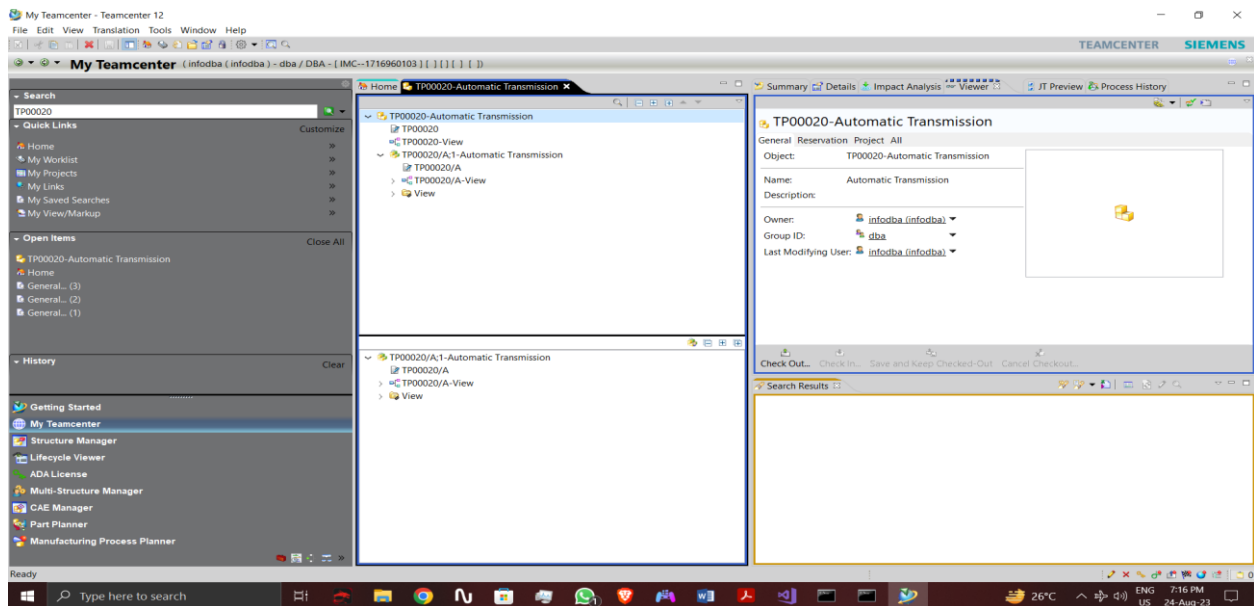


Fig -32: Checking of BOM Created through ITK in Teamcenter

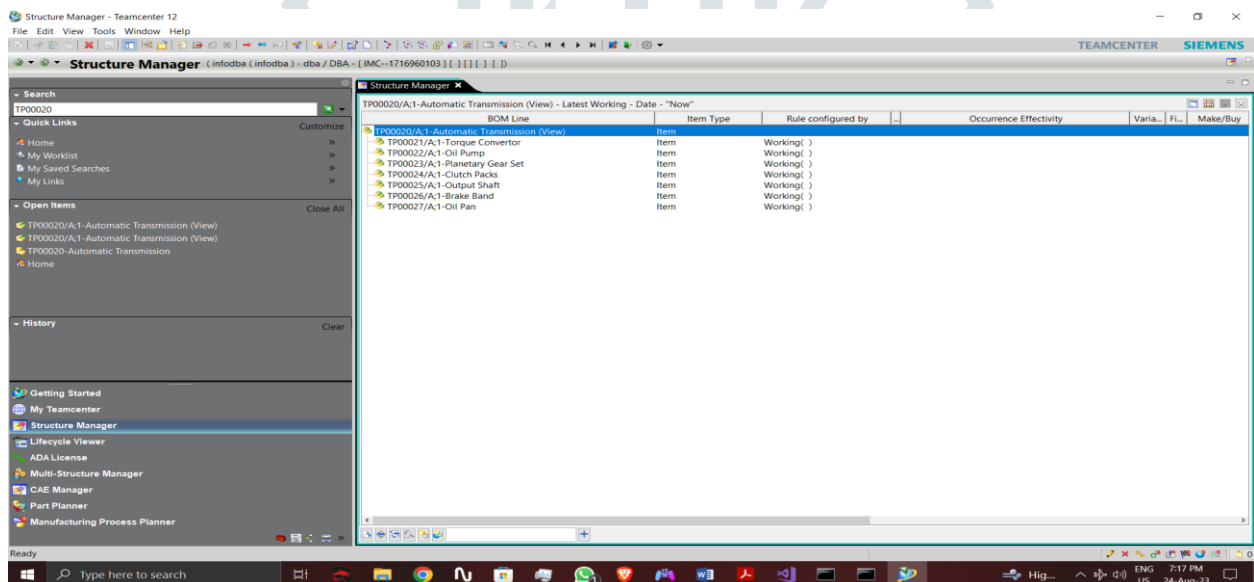


Fig -33: Loading of BOM Created through ITK in Structure Manager

As we have successfully compiled the code through Server Side Customization, then we have Log in to the Teamcenter and searched the BOM through Parent Item ID to check whether BOM is created or not. After searching the BOM through Query, we found our BOM created through code in My Teamcenter. For checking Child Items we have loaded BOM in Structure Manager and we found that BOM is loaded along with Child Items. We have fulfilled the requirement of client by creating BOM through reading CSV file.

### 4.2 Result for Rule Handler for Validation of Named Reference

As we have successfully complied the code of Rule Handler, then we have checked the results by attaching it on a Workflow and initiated the Workflow on Item Revisions for validation of Named Reference as per business requirements through client. After validation the results with and without named reference are explained below.

#### 4.2.1 Dataset on Item Revision without Named Reference

As we can notice clearly in below figure, if we are initiating workflow on Item Revision which contains Dataset attached to it without Named Reference i.e. without physical file then error is thrown stating that **Business rules for handler are not met** and workflow will not be initiated on Item Revision.

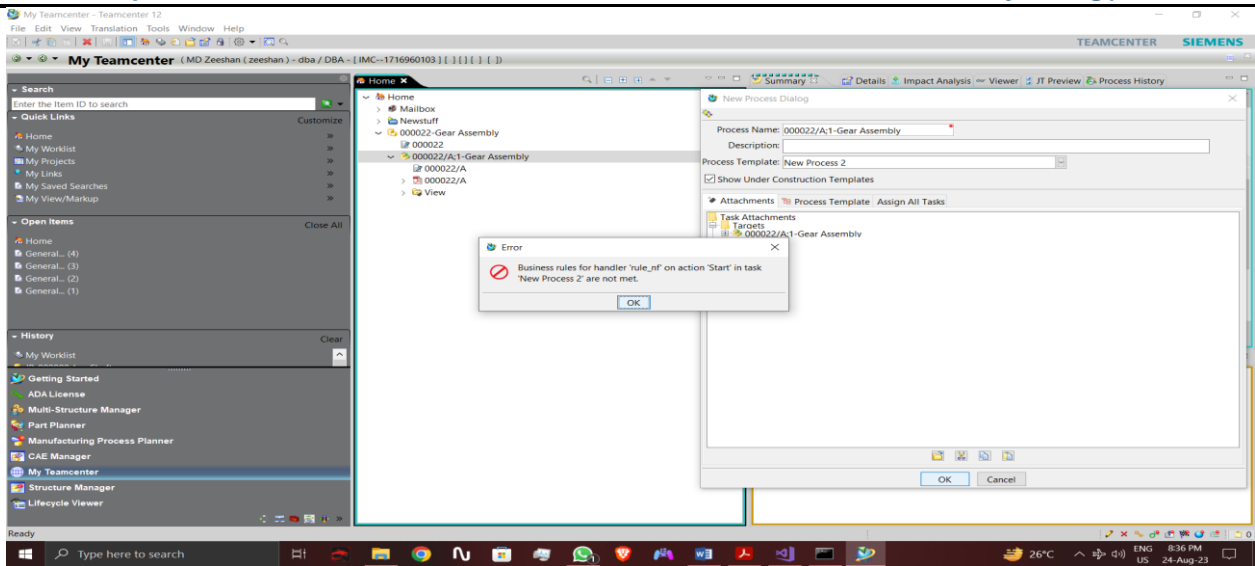


Fig -34: Workflow not initiated on Item Revision without Named Reference

#### 4.2.2 Dataset on Item Revision with Named Reference

As we can notice clearly in below figures, if we are initiating workflow on Item Revision which contains Dataset attached to it with Named Reference i.e. with physical file then workflow will be initiated on Item Revision as it meets the Business requirements and no error is thrown.

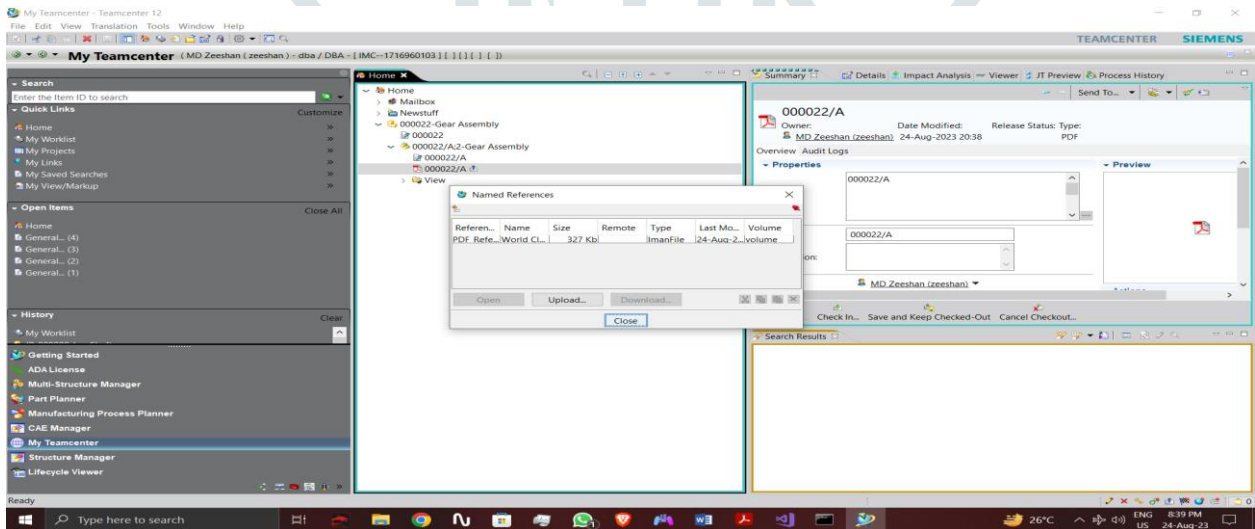


Fig -35: Dataset having Named Reference

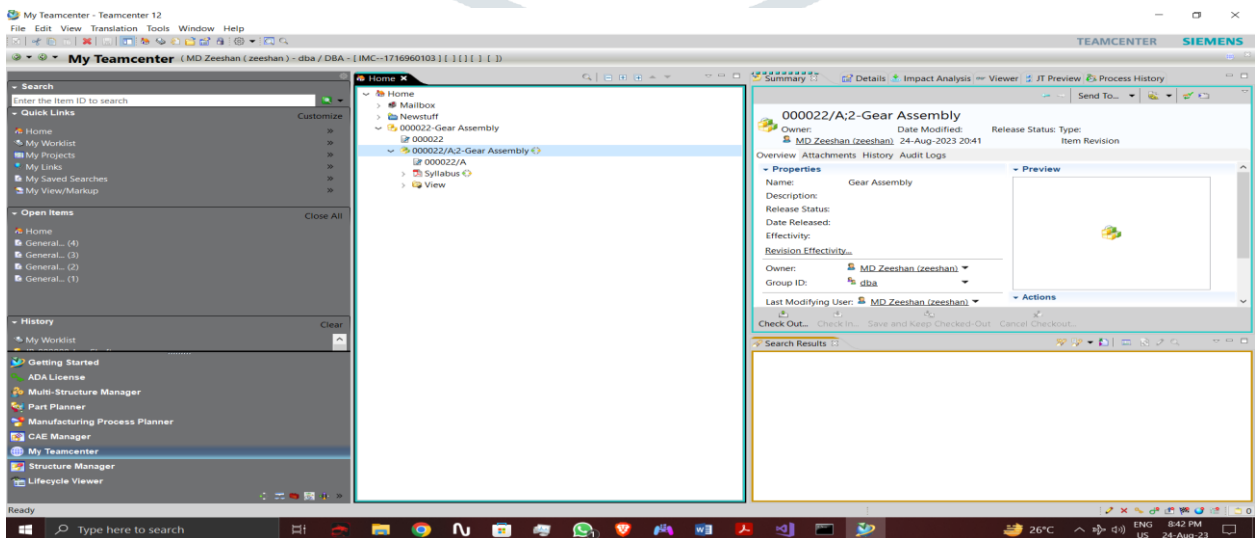


Fig -36: Workflow Initialed on Item Revision with Named Reference

**V. ACKNOWLEDGMENT**

I am grateful to Mr. Sayyad Shafik for his valuable guidance and insightful feedback throughout the research process. I would like to extend my thanks to my company for giving me an opportunity to work on this project and all the friends who have helped me in gathering data related to this work.

**REFERENCES**

- [1] Kurkin, Ondřej; Januška, Marlin (2010). "Product Life Cycle in Digital factory". Knowledge management and innovation: a business competitive edge perspective. Cairo: International Business Information Management Association (IBIMA): 1881–1886. ISBN 9780982148945.
- [2] Cunha, Luciano (20 July 2010). "Manufacturing Pioneers Reduce Costs By Integrating PLM & ERP". Onwindows.com. Archived from the original on 11 February 2017. Retrieved 7 February 2017.
- [3] Shrikant Pokale and Sawan Borul, "Client side customization for checking the user rights in Teamcenter-PLM" in International Journal of Applied Information Systems (IJAIS) Volume 5, Issue 10, August- 2013 ISSN 2249- 0868.
- [4] G. Shreenivasulu, "Eclipse Plug in based Teamcenter Customization" in International Journal of Advances in Computer Science and Technology Information Systems (IJACST) Volume 2, Issue 9, August-2013 ISSN 2320- 2602.
- [5] Teamcenter 12 Integration Tool Kit Function file, Siemens Product Lifecycle Management Software Inc., 2018.
- [6] W. Schindel, —System interactions: Making the heart of systems more visible, I Proc. of INCOSE Great Lakes Regional Conference, 2013.
- [7] P. B. Hart, —A PLM implementation for aerospace systems engineering-conceptual rotorcraft design, I vol. M.S. thesis, Dept. Mechanical. Eng., Georgia Institute of Technology., Georgia, May 2009.
- [8] Shrikant Basarkod, by PLM Neutral Customization frame work, White paper, Geometric, June 2009.

