VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Nam Hai Vu

TEAMCENTER ENVIRONMENT REPORTER

Information Technology

2014

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Nam Hai Vu
Title	Teamcenter Environment Reporter
Year	2014
Language	English
Pages	66 + 6 Appendices
Name of Supervisor	Ghodrat Moghadampour

The objective of this thesis was to develop an application which can seek customizations in a Teamcenter environment and create a report about them. Teamcenter is a tool for product data management. Its customizations are Teamcenter library, plugin, workflow, query, preference, XML rendering stylesheet and data model template.

The idea of this project is originated from IDEAL PLM to have a tool for collecting Teamcenter customization data. It gives project managers an overview of completed customizations when working with a new environment project. Based on the report, they can estimate working times needed for the whole project more accurately and efficiently.

The Teamcenter Environment Reporter consists of two parts, a Windows application implemented in C# .NET and a helper module implemented in C language. The helper module works as an intermediator which takes care of communication between the Windows application and Teamcenter. The application uses Teamcenter libraries and utilities to process data. At the end of the process, required data will be delivered to the user as a report in HTML format and as a zip package containing all accumulated customizations.

All require functions for the application were successfully implemented and all objectives were achieved. The application is presented to IDEAL PLM and waiting to be used in the near future.

Keywords Teamcenter, Teamcenter customization, Environment reporter

PREFACE

It has been three years since I started my degree in Information Technology at Vaasa University of Applied Sciences. During the time, I have not only learned invaluable IT knowledge such as software developing and testing, but I also have improved my interpersonal skills every time working in team with other classmates in a project. Combining everything together, I successfully manage to deliver my final project as my thesis.

The project, started from July 2014 and finished in October 2014, was inspired by my CTO Olli Väätäinen at IDEAL PLM. In the company, I have got invaluable first-hand working experiences in software industry. Therefore, I would like to thank Olli for giving me this very interesting chance. Also, my thank goes to Dr. Ghodrat Moghadampour for his supervising of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES, TABLES AND CODE SNIPPETS

LIST OF APPENDICES

ABBREVIATIONS

API	Application programming Interface
BW	Background Worker
COTS	Commercial off the shelf
CSS	Cascading Style Sheet
DLL	Dynamic Link Library
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
ITK	Integration Toolkit
JAR	Java archive
OOTB	Out of the box
PDM	Product Data Management
PLM	Product Lifecycle Management
SDK	Software Development Kit
SOA	Service Oriented Architecture
TC	Teamcenter
UI	User Interface
WF	Workflow
XML	Extensible Markup Language

1 INTRODUCTION

The Teamcenter Environment Reporter is a desktop application for the company IDEAL PLM, to support users when working with Teamcenter customizations.

1.1 Client Organization (IDEAL PLM)

IDEAL PLM is a Finnish company established in 1992. The company provides services related to PLM software and systems, including implementation, integration, software licensing and training. Currently, the company has more than 80 employees, located in five different offices around Finland (Vantaa, Tampere, Turku, Seinajöki and Vaasa) and Russia (Saint Petersburg). IDEAL PLM is the representative of Siemens PLM Software and Siemens COMOS products in Finland and Russia.

Numerous of projects have been either in progress or delivered to customers that varies from small company to large enterprises. /1/

1.2 Teamcenter

Teamcenter is one of Siemens product lifecycle management (PLM) applications. The software provides product data management (PDM) that helps users manage and control all information related to the products. In addition, Teamcenter also provides manufacturing data in the context of the product life cycle. /2/

1.3 Current situation

When starting a new project for a company, project managers and engineers often get trouble locating previous customizations and estimating needed working time for the project. They would have no idea about previous customizations because of missing or undetailed documentation.

The solution is to make a tool to help project managers and engineers have more detailed view of the new projects thus better monitoring and reducing time when determining work and time for the new projects.

1.4 Teamcenter Environment Reporter

Teamcenter Environment Reporter is a windows application that can collect all customizations of Teamcenter such as DLL files, JAR files, Preferences, Queries, Workflows, XML Rendering Stylesheets and make a document report for the accumulated customizations. In the end, a zip package contains all collected data will be delivered to users and data can be imported to other environments.

The application supports current Teamcenter versions and provide clear and easy to use GUI for users.

The application is going to be used within the company starting from next year (2015).

2 TECHNOLOGY

The application is built based on .NET Framework and programmed in C# language. The user interface (UI) is designed using Windows Forms which is included in .NET Framework. To gather some information, some Teamcenter APIs and built-in XML Parser are used.
When the application completes collecting information, a report page will be displayed as a web page using HTML, CSS with the zip file of customizations.

2.1 Teamcenter

The standard installation of Teamcenter is a powerful application that helps to manage product data. Part of Teamcenter's power is its adaptability to each company's work processes and user interface preferences. /3/

2.1.1 Teamcenter API

Teamcenter provides various ways to customize depending on business requirements, among which is Teamcenter API, an Integration Toolkit (ITK) provided by Siemens PLM Software. ITK is based on C and used for server customization. There are different ITK interfaces categorized by functionality for Teamcenter customization.

Unlike C, all ITK functions return value `ITK_ok(0)` if successful or an integer error code to inform if unsuccessful. Beside normal C data types, ITK also uses its own special data type: `tag_t`, object in ITK defined by tag of C type. It is a run time unique identifier isolates the client code using the object from its representation in memory, making it safe from direct pointer access. /4/

Although there are numerous ways of customizing in Teamcenter, the tool only focuses on main and frequently-used customizations. In the following sections they are explained in detail.

2.1.2 Teamcenter Customizable components

Below are all components that allow to be customized in Teamcenter.

Teamcenter DLL file is a library which contains ITK functions written in C and C++ that perform in Teamcenter server side.

Teamcenter plugin JAR file: The Teamcenter rich client is built on Eclipse rich client platform (RCP) framework. Thanks to the framework, now by using Eclipse plugin files (JAR) the Teamcenter rich client can be customized. /5/

Teamcenter workflow is an automation of business procedures that manages products' processes. Using workflows will improve efficiency and performance of product life cycle management. /6/

Teamcenter query is used to find information in Teamcenter, a search criteria will be inserted. Query is saved criteria that accelerate searches. /7/

Teamcenter Preferences are special environment variables stored in the Teamcenter database. Customize preferences can change Teamcenter application behaviors. /8/

Teamcenter data model template contains a data model definition about business objects and rules. Teamcenter can install many Data model templates to meet business requirements.

Teamcenter XML Rendering Stylesheets are XML documents stored in XMLRenderingStylesheet dataset. It defines how a dialog or panel is displayed in Teamcenter.

2.2 .NET Framework

The .NET Framework is a programming framework for Windows application developed by Microsoft.

Two main components of .NET framework are:

- Common Language Runtime (CLR): A runtime environment that manages compiling, executing, debugging, security…
- .NET Framework class library (FCL): A library includes pre-made classes and interfaces for users.

The .NET Framework has some main features:

- Cross Language Interoperability: Allow and support two different programming languages on the same data structure.
- Multiple platforms development: Now .NET provides different types of application:
    - Web application
    - Web service
    - Desktop application
    - Mobile application
- Memory management: CLR is responsible for managing memory (allocating when needed and freeing up memory when done) with help of .NET Garbage Collector. /9/

2.3 C#

C# is a programming language developed by Microsoft and included in .NET Framework SDK. C# syntax is similar to C and Java. C# is object oriented programming language, that means OOP concepts are encapsulation, inheritance, and polymorphism are supported. C# runs on the .NET Framework with the help of CLR to compile to native machine language. /10/

2.4 Visual Studio

Visual Studio is an integrated development environment by Microsoft to develop applications in multi platforms. It contains code editor with ItelliSense to help writing codes faster and more accurately.

Windows Forms that helps building GUI is included in .NET Framework with the support of Visual Studio in order to help programmers in designing interface simply just by dragging and dropping. /11/

2.5 HTML & CSS

Hyper Text Markup Language is a markup language for making Web pages. HTML describes the structure and layout of a web page by using defined elements and attributes. /12/

HTML5 is the latest version of the HTML standard with strong multimedia support for computers and mobile devices. This HTML5 is now the recommendation of the World Wide Web Consortium (W3C).

CSS is the language for describing the presentation of Web pages, including colors, layout, and fonts. CSS is used to format HTML elements in order to build Web pages with special styles.

CSS can be written inside HTML as an internal component or external as a .css file. The latter way is more useful because it would be easier for reading and maintenance. /12/

2.6 XML

The Extensible Markup Language (XML) is a markup language like HTML but using own designed elements instead of using pre-defined ones. XML is targeted to describe text data and widely used in web services. /13/

XML Parser: To read or write XML document, written XML parser is used to do such things to save time and reduce work.

.NET Framework provides several classes for parsing XML data such as XMLReader, XMLDocument, LINQ to XML. They have their own advantages and disadvantages that grant programmers flexibility in usage.

3 TEAMCENTER ENVIRONMENT REPORTER

This part covers the specification of Teamcenter environment reporter application that is resulted from collecting and analyzing requirements.

3.1 General description

The Teamcenter Environment Reporter will only be used by IDEAL PLM employees who do not have adequate documentation or have documentation but not completed one about previous customizations have been done in Teamcenter previously. The resolution is developing an application can collect all of previous customizations and make a brief report.

The application should fulfil the following requirements:

- User friendly: No need to be complicated, because it will be hard for everyone to remember console command or name of Teamcenter objects. A desktop application with simple GUI will fulfill this requirement.
- Selection for customizable parts: Users can select what type of customizations they want to extract, the options are: DLL, JAR, XML Rendering Stylesheet, Workflow, Query and Preference.
- Report and files: A report as well as a zip file containing all customization information is generated after each use for fulfilling this requirement
- Performance: The application performance is required to work in good amount of time even in a complicated system.
- Maintenance and upgrade: The new Teamcenter version is coming so the application needs to be well designed with clear specification. Moreover, the message of errors is required for bug fixing.

After analyzing all requirements, the following technical specification can be considered.

3.2 Functional description

The most important functionality of the application is collecting customizations and making a report for it. Another aspect is GUI's usefulness and simplicity.

There are requirements with different priority levels needed to be considered and described as the below quality function deployment.

1. Must have
    a. Create a report.
    b. Create a zip file.
    c. Have a user manual.
2. Should have
    a. Selectable specific options.
    b. A friendly GUI.
3. Nice to have
    a. Exception handling.
    b. Smart collecting

The below figure 1 is use case diagram of user interface.

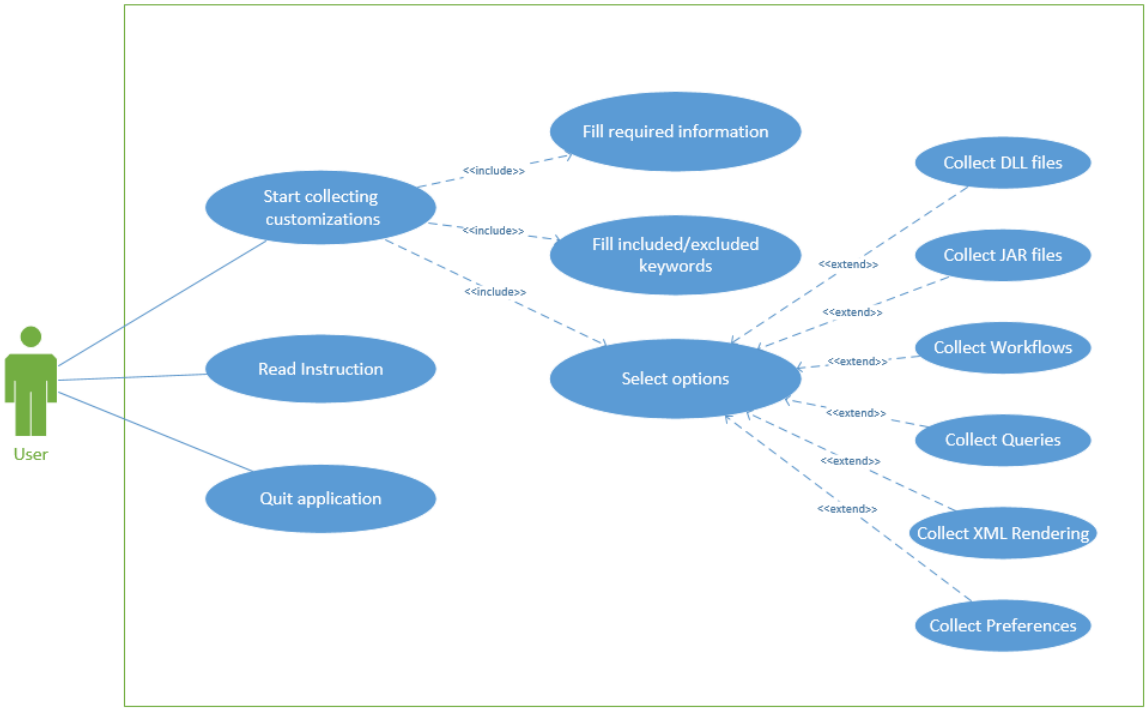


Figure 1: List of functions provided by the application

The application usage is straight forward therefor e the use case is also simple and short.

- Before clicking the Start button, users have to fill in all required information first:
    - Fill in required information for logging into Teamcenter, e.g. Teamcenter version, account credentials.
    - Input the keywords for comparing filter.
    - Select what kind of customizations users want to extract, e.g. DLL or JAR.
- Read Instruction: There is an instruction to guide new users how to use the application.
- Quit application: Stop running application and exit.

This is required information related to Teamcenter environment that users need to know and fill before using the application

Table 1: Required information

Required Information	**Description**	**Required for**
Teamcenter version	Target version of running Teamcenter	Preferences Queries Workflows XML Rendering Stylesheets
User credentials	Username, password, group for logging in to Teamcenter	Preferences Queries Workflows XML Rendering Stylesheets
Teamcenter configuration path	Path for Teamcenter configuration batch file	DLL files JAR files Preferences Queries Workflows XML Rendering Stylesheets

Besides, the application needs keywords that are put by users for filtering customizations. There are two types of keywords: included and excluded keyword.

Included keywords: Search for all customizations contains included keywords.

Excluded keywords: After collecting all customizations with included keywords, exclude the customizations containing excluded keywords.

There are 7 available options which correspond to 7 customizations for users to select:

Table 2: User selectable option

Options	**Description**
DLL files	DLL file path, last modified date
JAR files	JAR file path, last modified date

Preferences	Preference name, value, description
Queries	Query name, description
Workflows	Workflow file name
XML Rendering Stylesheet	XML file name
Templates	Data model template name

Export to zip package option: This is an option if users want to have a zip file compressing all customized files which are found and reported by the application. If not selected, only a report is output.

3.3 Application Class View

Because different types of customizations can be made, different classes are designed for each type of customization. In addition, other classes for the application's configuration, such as Teamcenter path or user credentials, are also needed and described in following sections.

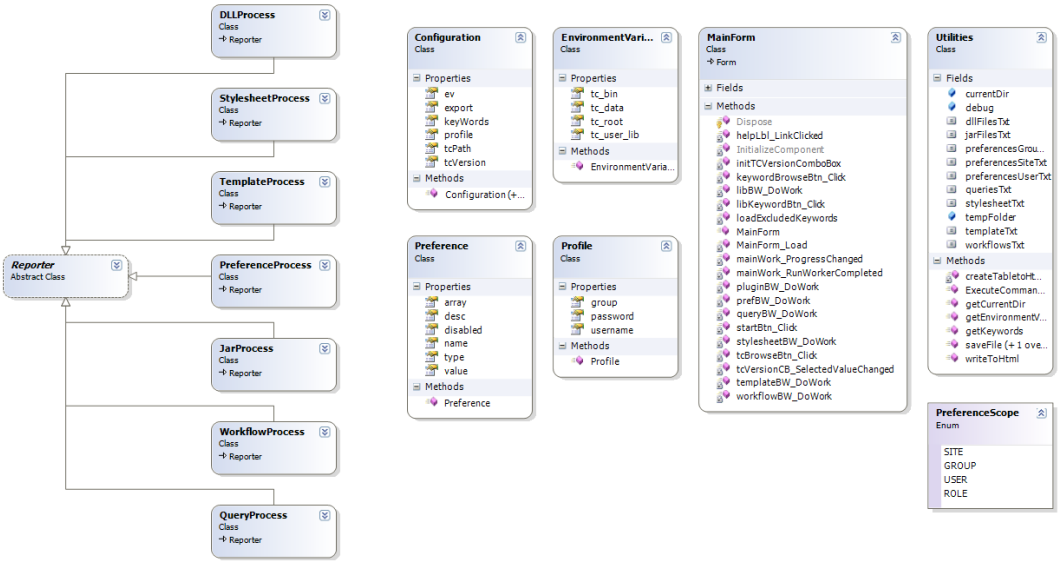


Figure 2: Application class diagram

To avoid strong coupling, classes must be separated and classified into two groups: UI and logic. This means that the UI class is at the front and is not relevant to process works. On the other hand, logic classes take care of the whole behind business logic. The separation will make the application become more of a loose coupling when being developed.

3.3.1 Individual class view

- **Preference class**

  Preferences are used as environment variables for Teamcenter. Two important components of preference are preference name and preference value. Preference value can be in different types such as string, array string and number. With the support of Teamcenter, preferences can be exported and imported as XML files. Hence, a class for Teamcenter preference instance is created with similar properties to preference XML attributes.

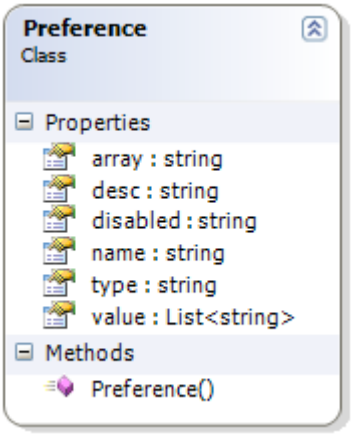


Figure 3: Preference class

- **Configuration class**

  In order to gather all users' settings right after clicking Start button, Configuration instance is created to store all current settings for logic to process. The current settings are Configuration's properties: EnvironmentVariable, export, List keywords, profile, Teamcenter path, and Teamcenter version.

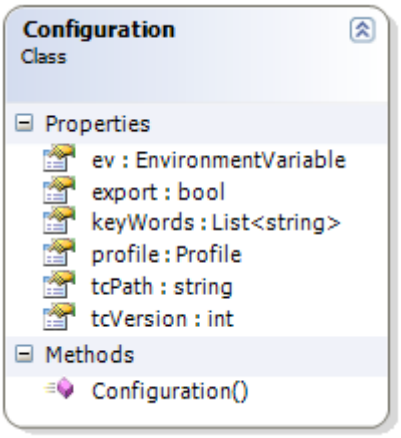


Figure 4: Configuration class

- **Profile class**

  Teamcenter utilities require username, password and group when executing commands in order to extract data from Teamcenter database.

This class stores users' credentials like username, password and group when logging into Teamcenter.

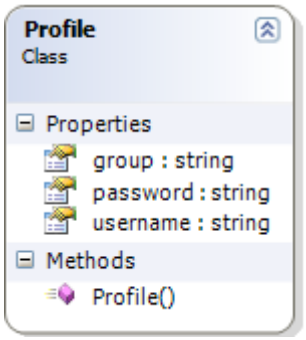


Figure 5: Profile class

- **EnvironmentVariable class**

  Unlike preferences, environment variables for Teamcenter are normal system variables, for example, Teamcenter directory, Teamcenter data directory, etc. These environment variables will be used to locate compulsory directories. This class stores needed Teamcenter environment variables.

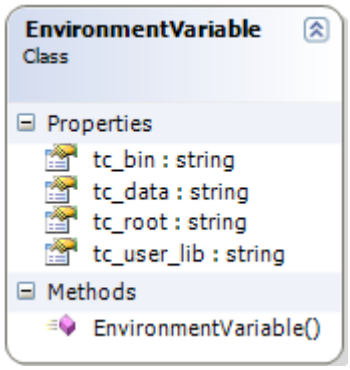


Figure 6: EnvironmentVariable class

- **Reporter class**

  Reporter class is an abstract class. It includes protected attributes for its children. A normal method `readConfig()` is used to get user input values from UI and populate them to its attributes. Abstract method `report()` is for being overridden by inherited classes. It will then return `List<string[]>` type that needed for the final HTML report.

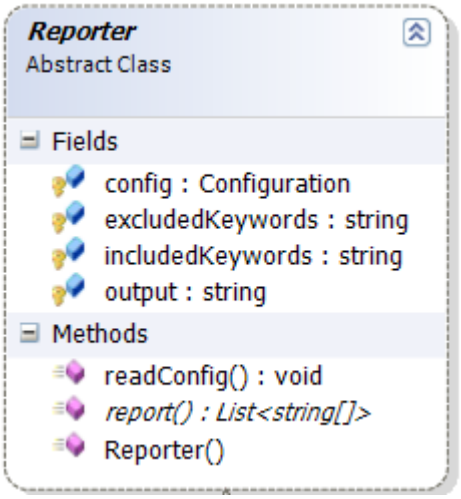


Figure 7: Reporter class

These below classes play a role as processing logic behind UI. All of them inherit abstract class Reporter and override `report()` method.

DLLProcess class

This class is responsible for getting all DLL files from TC_BIN environment variable and filter these using provided keywords. It also copies filtered files to new location for packaging.

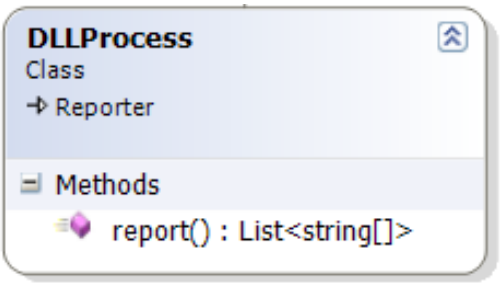


Figure 8: DLLProcess class

JarProcess class

This class is responsible for getting all JAR files from plugin folder of Teamcenter and filter them using provided keywords. It also copies filtered files to new location for packaging.

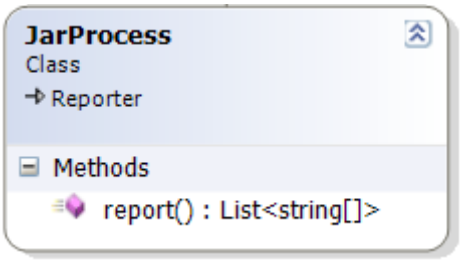


Figure 9: JarProcess class

PreferenceProcess class

Teamcenter preferences will be exported into an XML file. Processing, parsing, filtering XML data is the responsibility of this class.

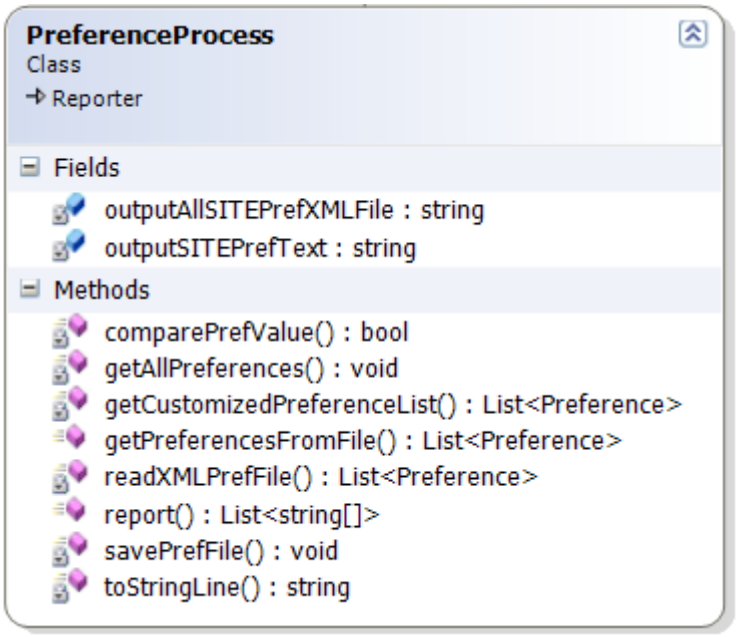


Figure 10: PreferenceProcess class

QueryProcess class

Query is a string used for searching specific data from the Teamcenter database with flexibility. Similar to preference class, this class accumulates queries from XML data.

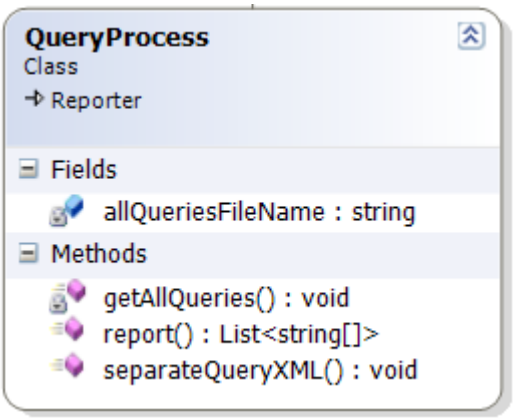


Figure 11: QueryProcess class

StylesheetProcess class

XML Rendering Stylesheet can be customized to create or change view of displaying Teamcenter data for example an item's properties. XML Rendering stylesheet methods and properties form this class.

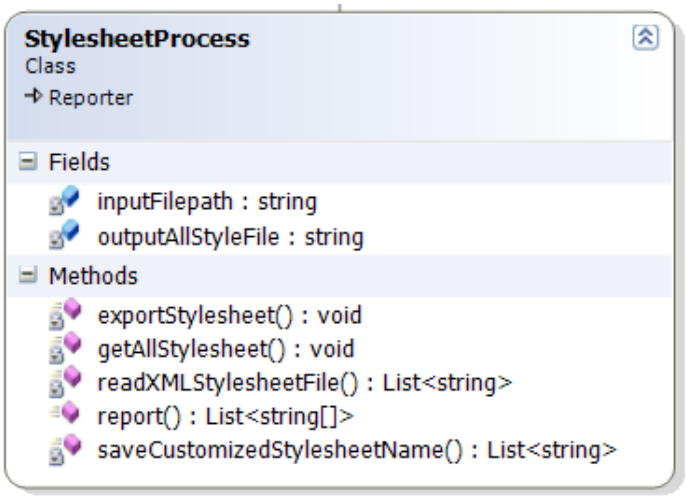


Figure 12: StylesheetProcess class

TemplateProcess class

Dependent Data model templates for Teamcenter are stored in XML files. This class deals with parsing XML data and make it available in the report.

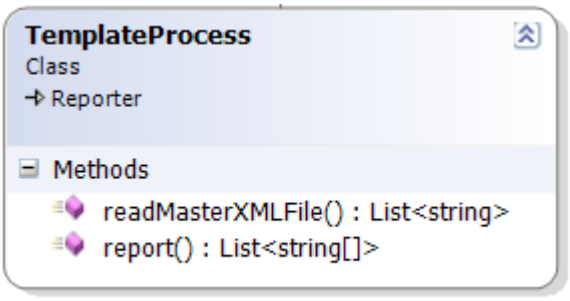


Figure 13: Template class

WorkflowProcess class

Resembling other customizable modules, workflow can be exported into an XML file with all information and is saved as XML data. This class handles exporting workflow data and processing it for the report.

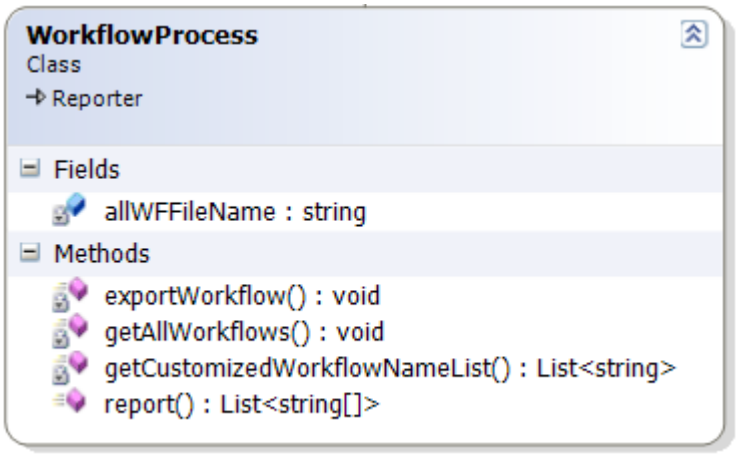


Figure 14: WorkflowProcess class

Utilities class

Utilities class is a group of frequent utility functions and constant variables. Methods in this class are defined public and static. Some functions are overloaded each other to be flexible in used.

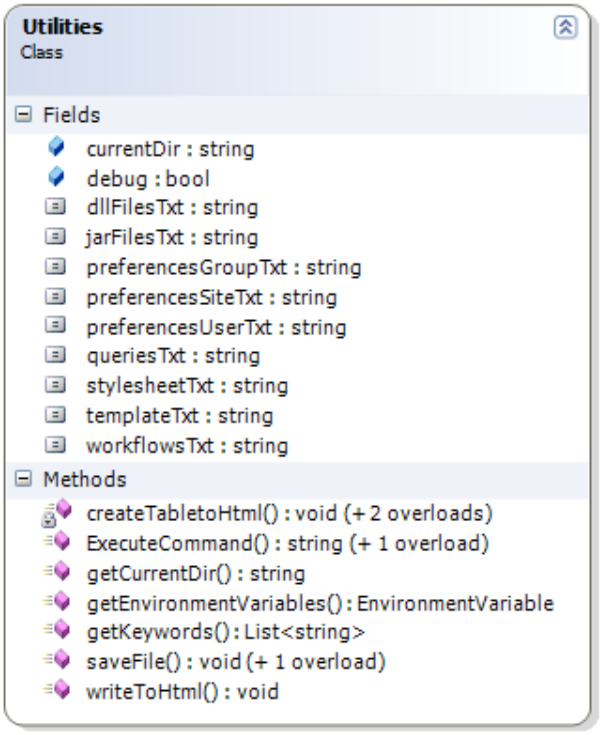


Figure 15: Utilities class

UIForm class

The GUI class contains event handlers for UI controls such as button, textbox, checkbox.

3.3.2 Task Class View

There are 6 tasks for 6 corresponding available options and a task for collecting all required information:

The application will create a new instance of class Configuration from user's inputs including TC account information, environment variables and application

configurations.

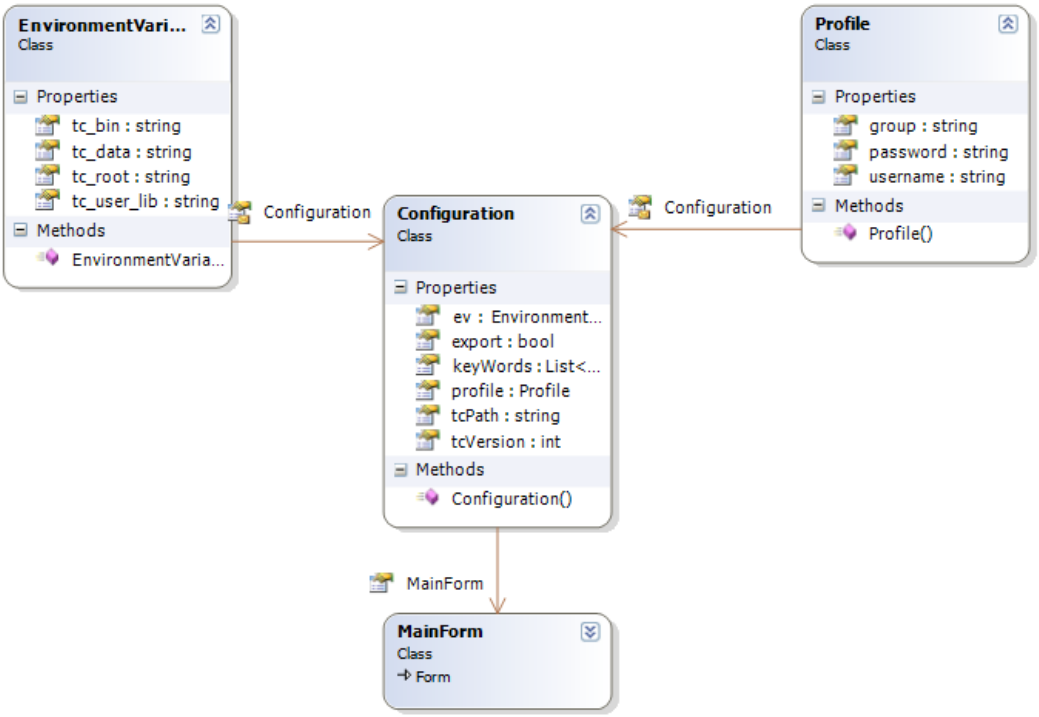


Figure 16: Configuration and its relation class diagram

The application creates a new thread and this will be responsible for extracting all Teamcenter customization information and filter them using provided keywords. The logic for this is placed in below classes called Process classes. These inherits Reporter class that is defined as an abstract class.

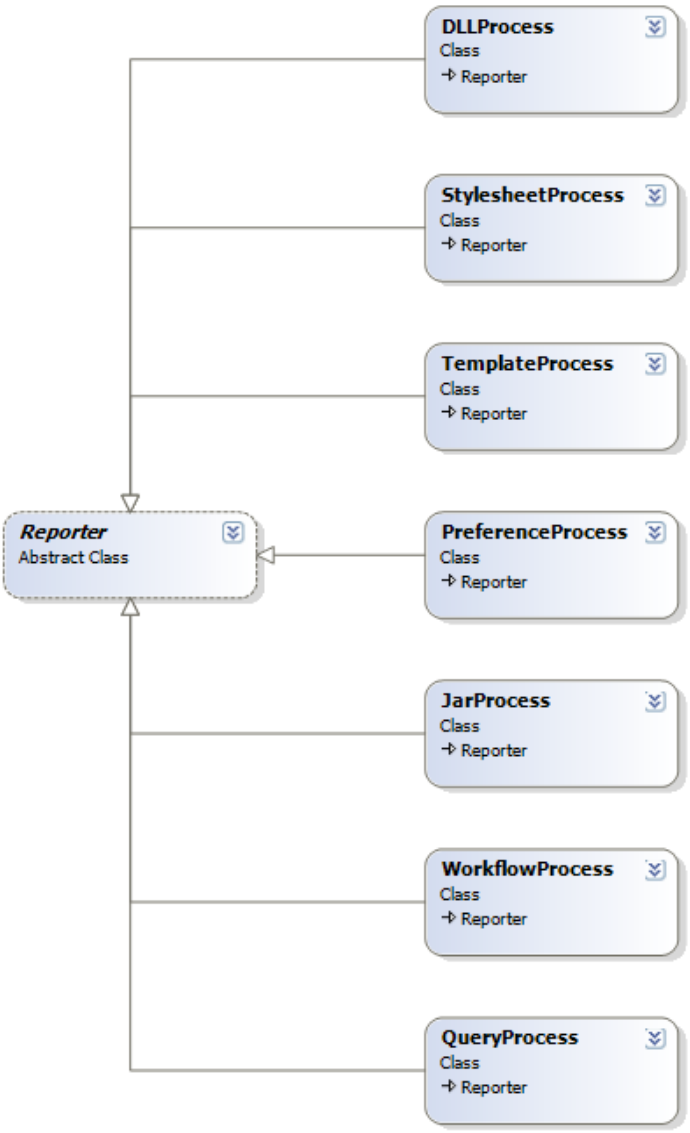


Figure 17: Process classes diagram

3.4 Detailed functional description

Sequence diagram is to describe interactions between the application's components with each other by passing messages among them.

3.4.1 Collecting DLL

The figure below describes how collecting DLL function works.

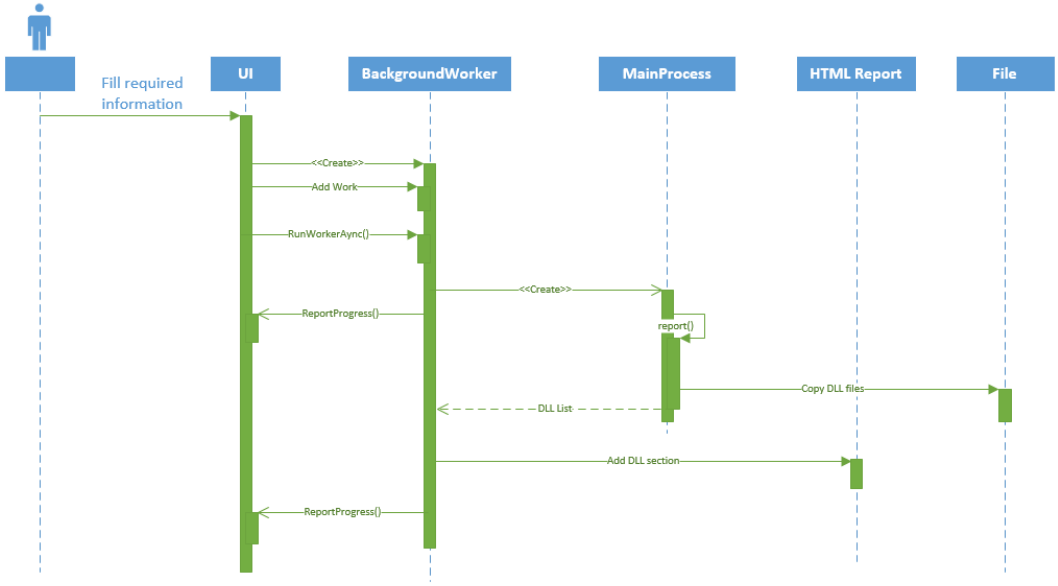


Figure 18: Collecting DLL function

1. The user selects a DLL option, clicks the Start button in UI: A new instance of a background worker is created and added the DLL work.
2. The BW starts running.
3. The BW reports progress to the UI.
4. A new instance of DLLProcess invokes overridden `report()` method. The return is a list and file paths of collected DLL files.
5. The BW adds DLL list to the HTML report.
6. The BW finishes and reports work progress to the UI.

3.4.2 Collecting JAR

The figure below describes how collecting JAR function works.

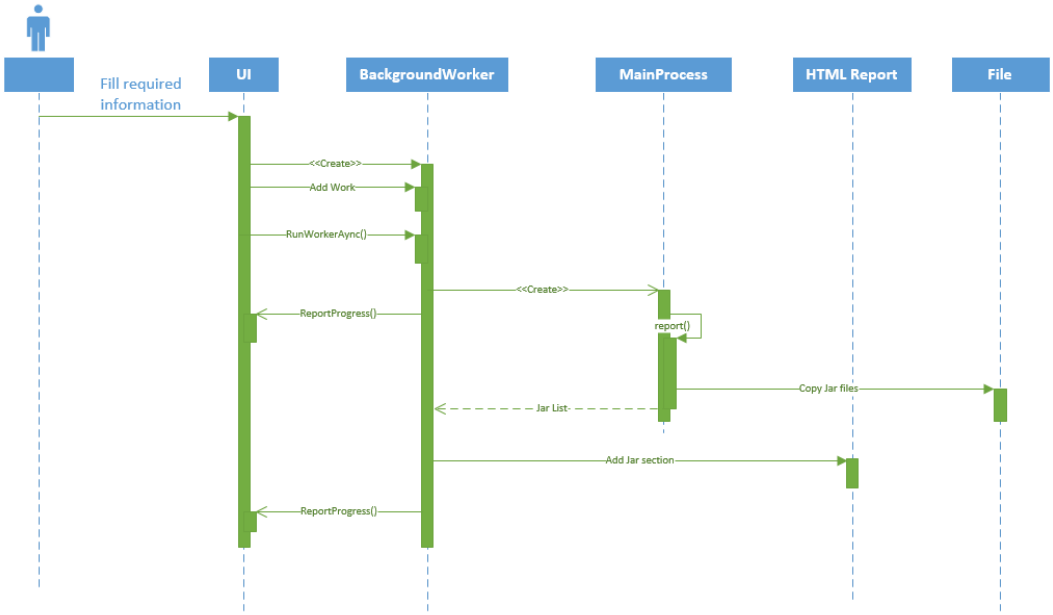


Figure 19: Collecting JAR diagram

1. The user selects the JAR option, clicks the Start button in the UI: A new instance of a background worker is created and added the JAR work.
2. The BW starts running.
3. The BW reports progress to the UI.
4. A new instance of the JARProcess invokes overridden `report()` method. The return is a list and file paths of collected the JAR files.
5. The BW adds the JAR list to the HTML report.
6. The BW finishes and reports work progress to the UI.

3.4.3 Collecting Preference

The figure below describes how collecting Preference function works.

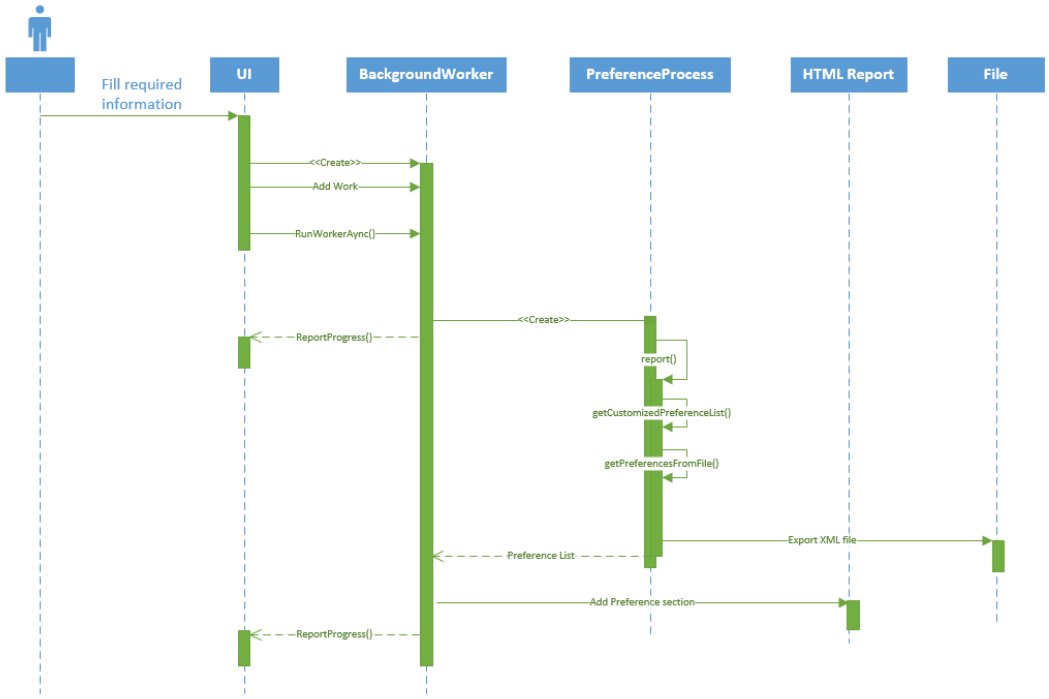


Figure 20: Collecting Preference diagram

1. The user selects Preference option, clicks the Start button in the UI: A new instance of a background worker is created and added the Preference work.
2. The BW starts running.
3. The BW reports progress to the UI.
4. A new instance of the PreferenceProcess invokes overridden `report()` method.
5. The PreferenceProcess instance gets a list of customized preferences by calling `getCustomizedPreferenceList()`.
6. The PreferenceProcess instance exports found preferences to XML file and returns a list of preference information.
7. The BW adds the Preference list to the HTML report.
8. The BW finishes and reports work progress to the UI.

3.4.4 Collecting Query

The figure below describes how collecting Query function works.

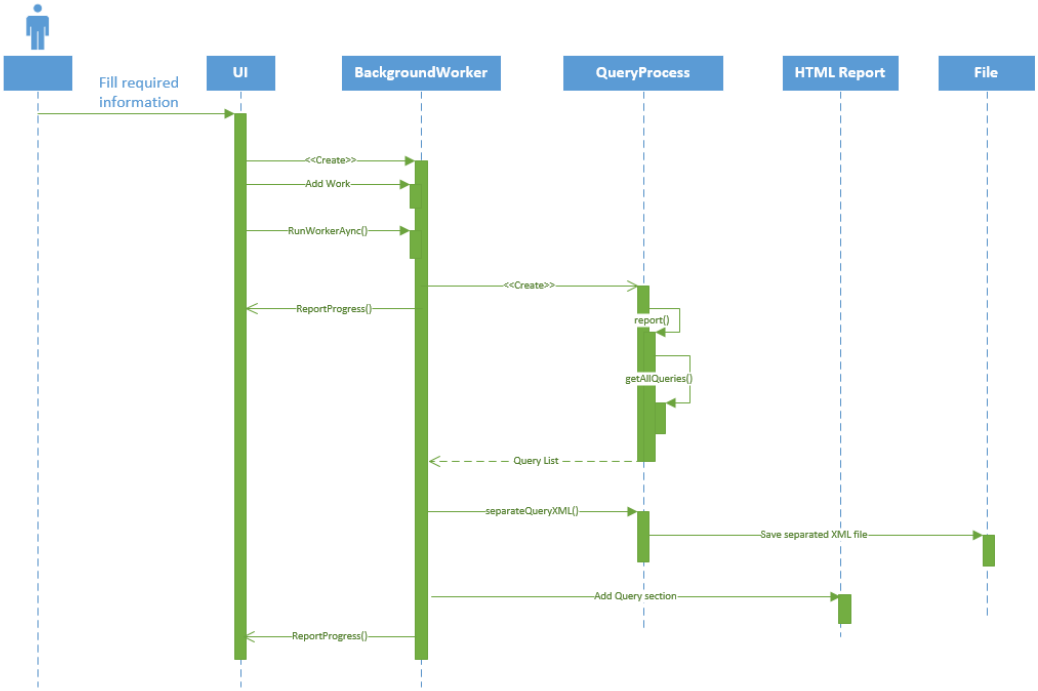


Figure 21: Collecting Query

1. The user selects the Query option, clicks the Start button in the UI: A new instance of a background worker is created and added the Query work.
2. The BW starts running then reporting progress to the UI.
3. New instance of QueryProcess invokes overridden `report()` method.
4. The QueryProcess instance gets a list of all queries by calling `getAllQueries()`.
5. The QueryProcess instance exports filtered query to one XML file and returns a list of query information.
6. The QueryProcess instance invokes `separateQueryXML()` function.
7. The QueryProcess separates queries one by one from an XML file, then saves each to XML files.
8. The BW adds Query list to the HTML report.
9. The BW finishes and reports work progress to the UI.

3.4.5 Collecting Workflow

The figure below describes how collecting Workflow function works.

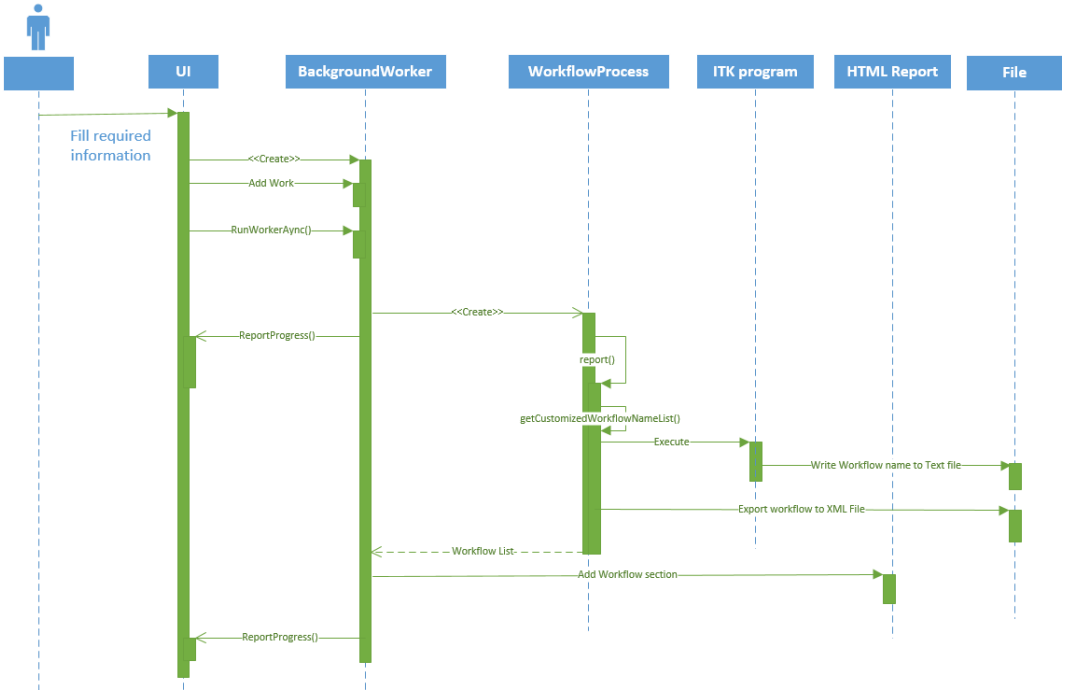


Figure 22: Collecting Workflow digram

1. The user selects the Workflow option, clicks the Start button in the UI: A new instance of a background worker is created and added the new work.
2. The BW starts running.
3. The BW reports progress to the UI.
4. A new instance of WorkflowProcess invokes overridden method `report()` method function.
5. The WorkflowProcess instance gets a list of customized Workflow names by calling `getCustomizedWorkflowNameList()`.
6. The WorkflowProcess instance executes an ITK program which writes all workflow names to a text file by name.
7. The WorkflowProcess exports workflows to corresponding XML files.
8. The WorkflowProcess instance returns a list of workflow information.
9. The BW adds the Workflow list to the HTML report.
10. The BW finishes and reports work progress to UI.

3.4.6 Collecting XML Rendering Stylesheet

The figure below describes how collecting XML Rendering Stylesheet function works.

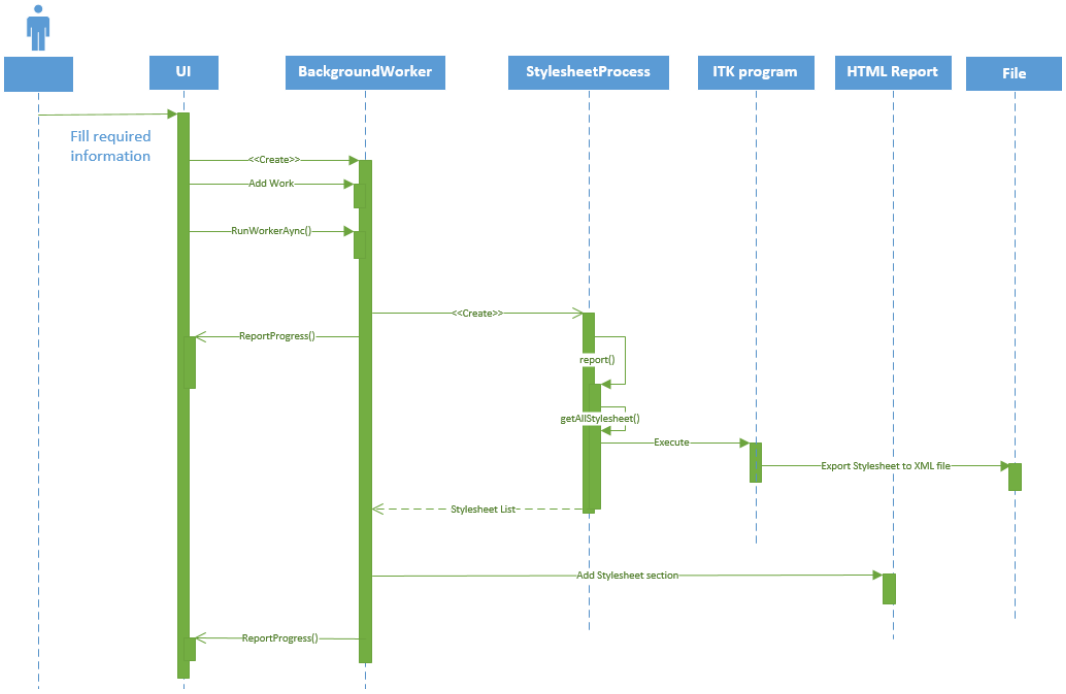


Figure 23: Collecting XML Rendering Stylesheet diagram

1. The user selects XML Rendering Stylesheet option, clicks the Start button: A new instance of a background worker is created and added the new work.
2. The BW starts running.
3. The BW reports progress to the UI.
4. A new instance of StylesheetProcess invokes overridden method `report()`.
5. The StylesheetProcess instance will create a file with a list of all Stylesheet name by calling `getAllStylesheet()`.
6. The StylesheetProcess instance executes an ITK program which exports stylesheets to XML files by name.
7. The StylesheetProcess instance returns a list of stylesheet information.
8. The BW adds XML Rendering Stylesheet list to the HTML report.
9. BW finishes and reports work progress to the UI.

3.4.7 Collecting Data model template name

The figure below describes how collecting Data model template name function works.

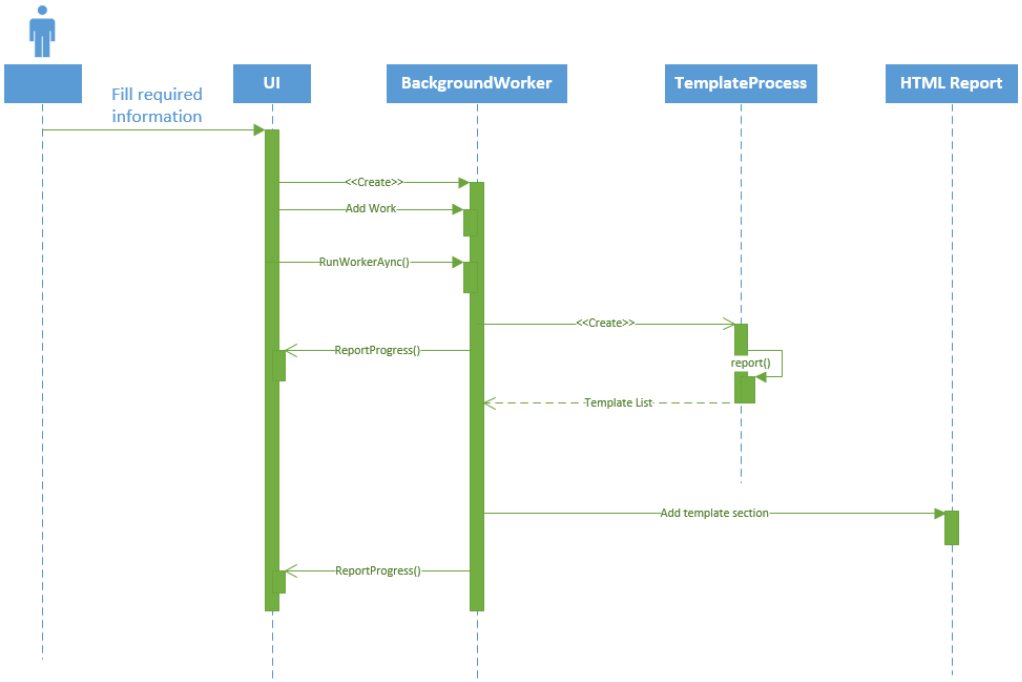


Figure 24: Collecting data model template name

1. The user selects Data model Template option, clicks the Start button in the UI: A new instance of a background worker is created and added the Data model Template work.
2. The BW starts running.
3. The BW reports progress to UI.
4. A new instance of TemplateProcess invokes overridden method `report()` method.
5. The TemplateProcess instance returns a list of template information.
6. The BW adds the template list to HTML report.
7. The BW finishes and reports the work progress to the UI.

4 IMPLEMENTATION

This chapter is divided into 2 main parts, the GUI and the main application are discussed in the following sections.

4.1 Graphical User Interface

Thanks to Windows Forms, now the GUI design for Windows application is much easier than ever before. Windows Forms is a graphical API included in .NET Framework. With the help of Visual Studio, the application's GUI is designed with Windows Forms controls such as textboxes, buttons, progress bars, etc. and automatically generated action events such as mouse clicks, key presses, to name a few, attached to those controls.

The GUI also needs to support resizing windows by separating controls into panels and anchors. Hence it is possible to resize the application window if users get trouble with their screen resolution.

In the application, the GUI's target is to be user-friendly but to concentrate on the report result. So the design is as simple as described with below steps.

1. Provide Teamcenter information (username, password, version…)
2. Provide acknowledged keywords.
3. Select supported options.

Due to different types of Teamcenter customizations, different tab views are required:

4.1.1 Main Tab

The main tab is the most important tab, it consists of all controls that require input from users.

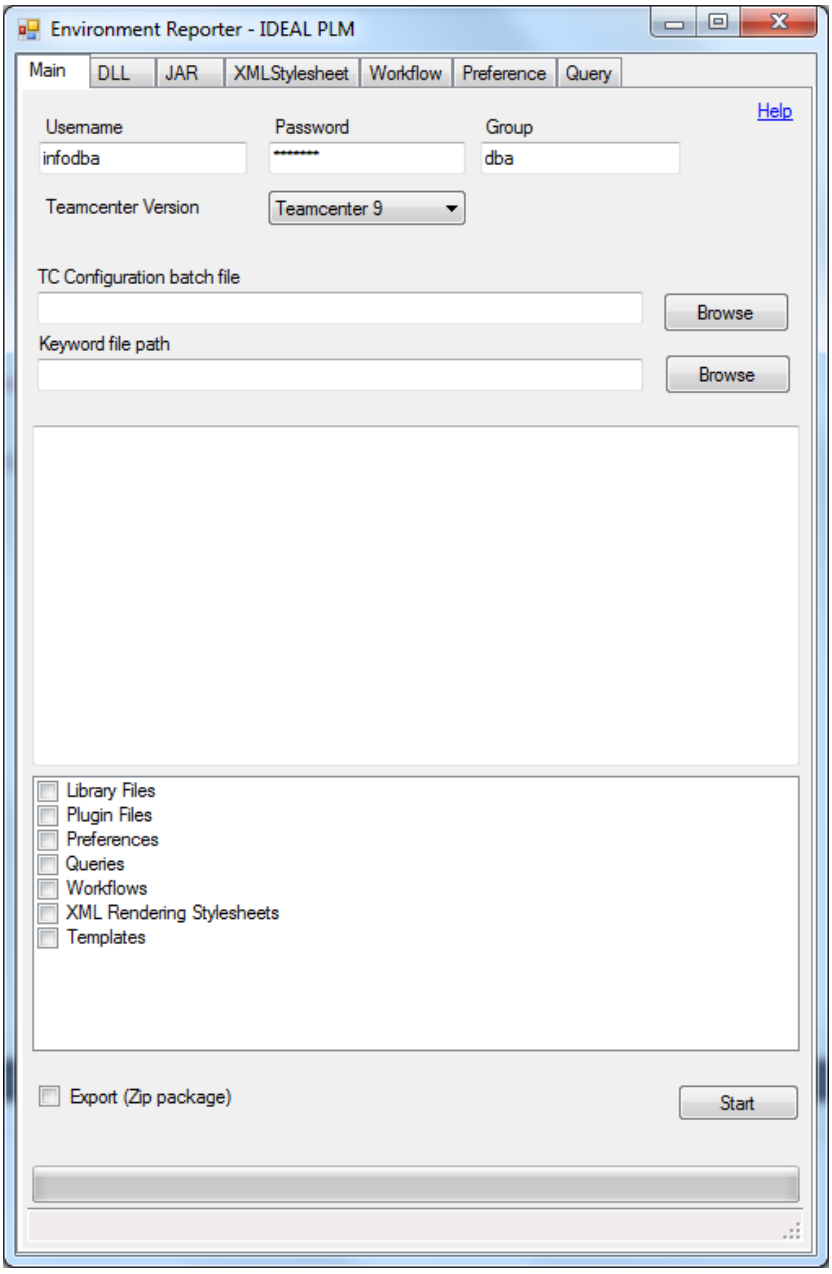


Figure 25: Main Form

Firstly, users need to provide the application their using Teamcenter version. There are 3 supported popular versions of Teamcenter 8.3, 9.1 and 10.1. When users select a version, the corresponding excluded keywords will be automatically filled.

Then, users insert the account that has adequate administrator privilege.

In order to locate Teamcenter root and other profile variables, the application needs Teamcenter configuration batch file (a .bat file contains all needed environment variables of Teamcenter).

Users can choose a text file containing keywords or type keywords to a textbox or both.

There are different checked boxes that allow users to select which type of customizations they want to have. Users can also select Export checkbox and a zip package will be delivered after the report.

At the bottom of the window, a progress bar with status label will indicate status of the running application.

An instruction for new users with clear instructions and notes will be opened if instruction button is clicked.

4.1.2 DLL tab

Below is a screenshot of a DLL tab.

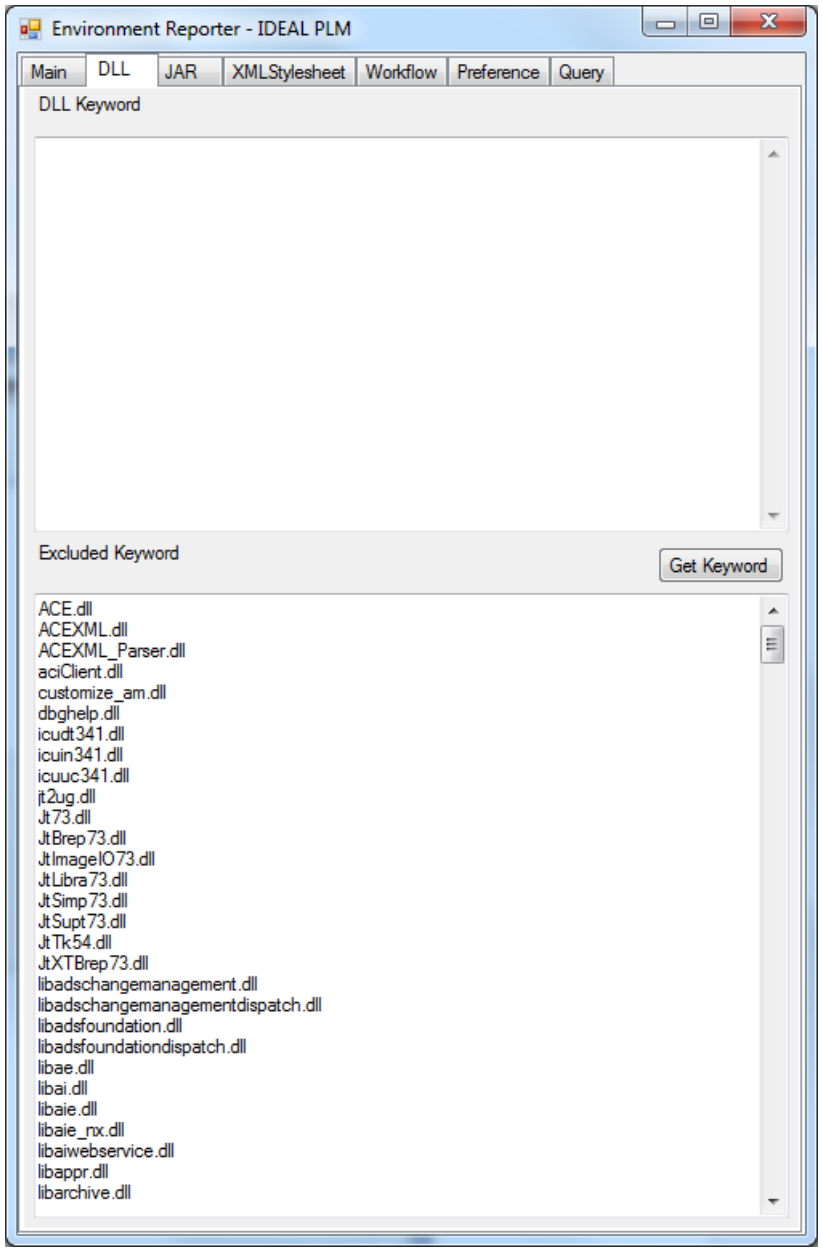


Figure 26: DLL tab

There are two multiple lines textboxes, one is included keyword, the other for excluded DLL names. The excluded keywords box is automatically loaded when users select TC version.

Because the preference `TC_customization_libraries` holds most of the customized DLL, so the user can get these DLL names by pressing Get Keyword button under the keyword textbox. The application will ask this preference's values and copy these values to the textbox.

4.1.3 Other tabs

Similar to the DLL tab, other tabs such as JAR, XML, Workflow, Preference, Query tab, have 2 multiple lines textboxes for included and excluded keywords. Excluded keywords are automatically loaded when selecting the TC version.

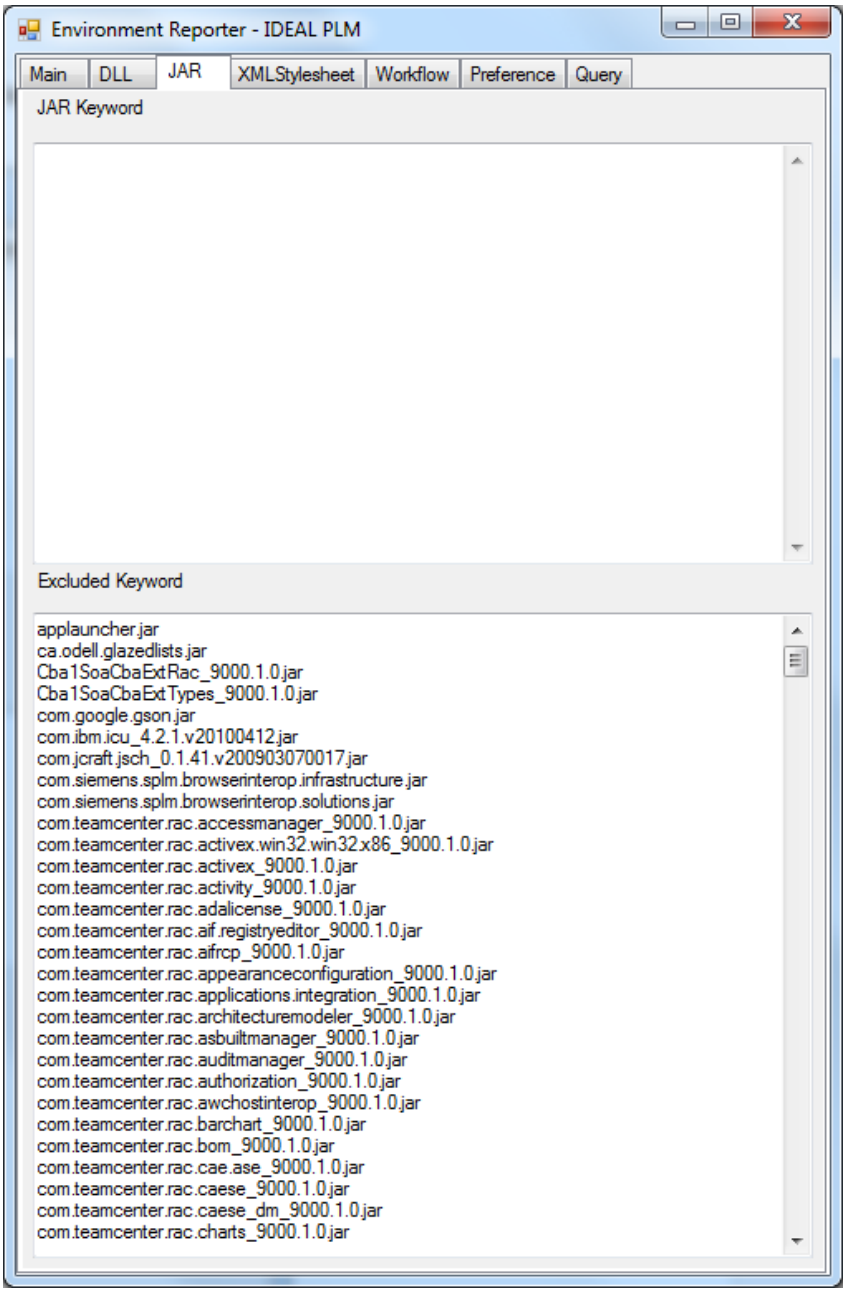


Figure 27: JAR tab

4.2 Main application

Implementation in the main application is divided into parts which are described and discussed here.

4.2.1 Filter

In order to collect customization data accurately, filters are needed to be applied to every Teamcenter customizable components using included and excluded keywords. Both included and excluded keywords are normally identified by users. In addition, Teamcenter already has out of the box (original functionality ready to be used) components so excluded keywords must have these OOTB names. Therefore, included keywords and excluded keywords are displayed as multiple lines text boxes in the GUI.

In the application, filter process algorithm for DLL, JAR, Workflow and Query is simplified in below picture:

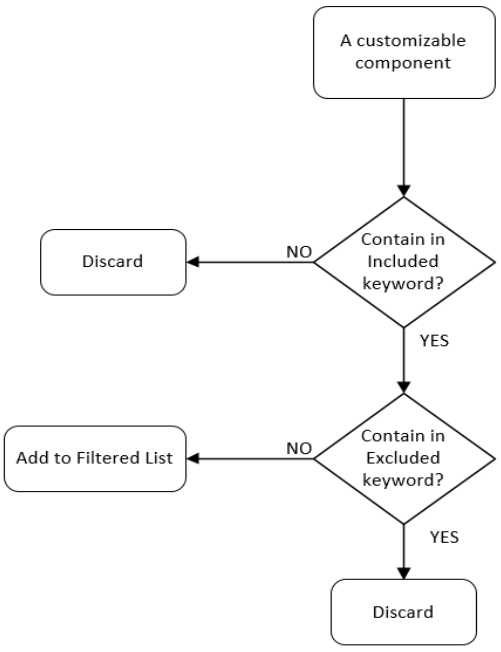


Figure 28: Normal filter logic

The logic for preference is described in the following figure

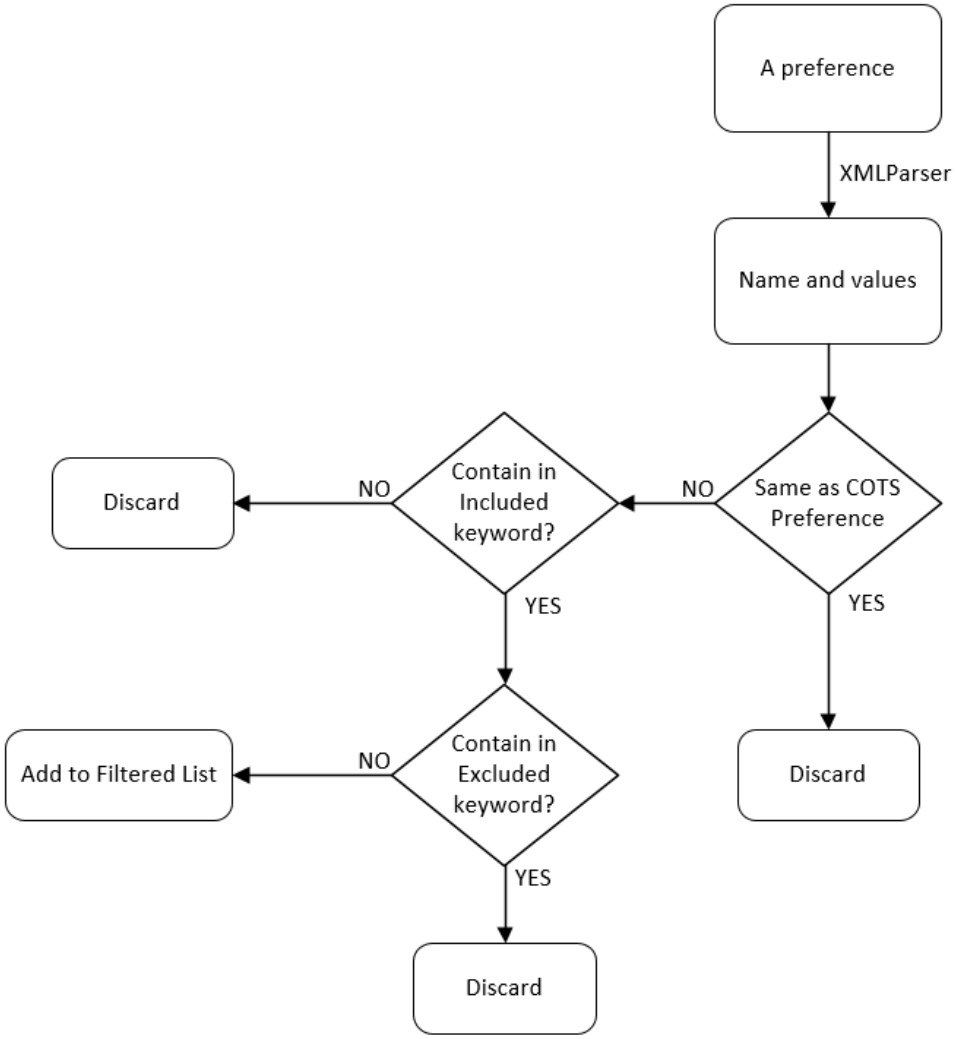


Figure 29: Preference filter logic

This preference case is more complex than the others because Commercial off-the-shelf (the ready-made item but needs to be configured before use) preference are usually modified. So the XMLParser has to work here to read preference content and compare preference name, values with the COTS ones before going through the normal sequence with included and excluded keywords.

4.2.2 Teamcenter Utilities

Teamcenter provides many stand-alone programs that can command directly to Teamcenter without logging into client. It requires administrator privilege and a configured environment command line to run.

To manually set up the Teamcenter environment, use these commands in the command prompt line:

```
set TC_ROOT=Teamcenter_root_directory
set TC_DATA=Teamcenter_data_directory
call %TC_DATA%\tc profilevar
```

Code snippet 1: TC environment configuration batch

Since Teamcenter always has a batch file doing the same, users only have to provide this batch file to the application instead of making their own. The batch file is ready under TC_ROOT/tcmenu/.

A static method named `ExecuteCommand` in the Utilities class is defined to run any Teamcenter command. The first parameter is the path of the batch file, the second one is a command. The following code snippet described `ExecuteCommand()` method.

```
public static string ExecuteCommand(string tcPath, string command)
{
	ProcessStartInfo processInfo = new ProcessStartInfo(@"cmd");

	//display command window
	if (debug)
		processInfo.CreateNoWindow = false;
	else
		processInfo.CreateNoWindow = true;

	//Print all input, output and error to screen
	processInfo.UseShellExecute = false;
	processInfo.RedirectStandardError = true;
	processInfo.RedirectStandardInput = true;
	processInfo.RedirectStandardOutput = true;
	string output = "";

	Process process = Process.Start(processInfo);
	if (process != null) {
		//setting Teamcenter configuration
		process.StandardInput.WriteLine(tcPath);
		//write the command that is executed
		process.StandardInput.WriteLine(command);

		process.StandardInput.Close(); // line added to stop
		process from hanging on ReadToEnd()
		output = process.StandardOutput.ReadToEnd();

		if (debug) {
			string error =
		process.StandardError.ReadToEnd();
		}

		process.Close();
	}
```

Code snippet 2: ExecuteCommand function

With Teamcenter utilities, Workflows, Queries, and XML Rendering Stylesheets can be exported by using command: `plmxml_export`. But this command is limited to tailor-made options. Therefore, stand-alone ITK programs must be coded to fulfill the limitations.

- When exporting Workflows, `plmxml_export` only works if the workflow names are already known. An ITK program can find all Teamcenter Workflows and write them down to a text file. The application reads text file and calls `plmxml_export` for each workflow.

```
//plmxml_export is used to export Workflow
Utilities.ExecuteCommand(config.tcPath, @"plmxml_export -u=" +
profile.username + " -p=" + profile.password + " -g=" +
profile.group + " -xml_file=\"" + outputFilePath + "\"" + " -
template=\"" + workflowName + "\"");
```

Code snippet 3: Execute export workflow command

- When exporting Queries, `plmxml_export` only exports an XML file containing all queries of Teamcenter. To separate into smaller query XML file, XMLDocument will do the job.

```
// plmxml_export is used to export Query
Utilities.ExecuteCommand(config.tcPath, "plmxml_export -u=" +
profile.username + " -p=" + profile.password + " -g=" +
profile.group + " -imantypedef=ImanQuery -xml_file=\"" +
outputFile + "\"");
```

Code snippet 4: Execute export query command

- Concerning XML Rendering Stylesheet, `plmxml_export` only export XML file containing all Stylesheet dataset information under XML format. So an XML parser and ITK program are needed to filter and export them into individual XML files.

```
//plmxml_export is used to export XML Rendering Stylesheet
Utilities.ExecuteCommand(config.tcPath, @"plmxml_export -u=" +
profile.username + " -p=" + profile.password + " -g=" +
profile.group + " -type=XMLRenderingStylesheet -xml_file=\"" +
outputFilePath + "\"");
```

Code snippet 5: Execute export XML Rendering Stylesheet command

- Regarding preference, Teamcenter Utilities provides very useful command `preference_managers` located in TC_BIN directory. In order to execute this command to export preferences in C#, these code snippets are used.

```
//preference_manager is used with parameters in C#
Utilities.ExecuteCommand(config.tcPath, @"preferences_manager -
u=" + profile.username + " -p=" + profile.password + " -g=" +
profile.group + " -mode=export -scope=" + scope + " -file=\"" +
inputPrefFile + "\" -out_file=\"" + outputPrefFile + "\"");
```

Code snippet 6: Execute export preference from a file command

4.2.3 ITK Program

Collecting DLL and JAR files is quite simple because these files are located in specified directories within Teamcenter root directory. But other components need other approach methods.

Workflows, XML Rendering Stylesheets, Queries and Preference can be exported by using Teamcenter utilities in the Teamcenter environment command line but it is not fully supported. Therefore, ITK program will be used to fulfill unsupported parts.

This ITK program is used to find all workflow names in Teamcenter and write them down to a text file. It can be done by the code snippet below:

```
int      workflow_count  = 0;
char     **nameList      = NULL;

//initialize module and find all workflows
ifail = ERROR_CHECK(EPM_init_module());
ifail = ERROR_CHECK(EPM_ask_all_procedure_names(&workflow_count,
&nameList));

//for each found workflow, write it down to a text file
if(workflow_count > 0){
    FILE *pFile;
    pFile = fopen(fileName, "w");
    for(i = 0; i < workflow_count; ++i){
        fprintf(pFile, "%s\n", nameList[i]);
    }
    fclose (pFile);
}
```

Code snippet 7: ITK code for extract and write workflow names to a text file

A text file with all workflow names will be created after running this ITK program. From that we can export each workflow into XML file using Teamcenter utilities.

About XML Rendering Stylesheets, first, Teamcenter utilities exports a XML file containing all Teamcenter stylesheets. Then stylesheet names will be filtered and parsed by the XMLParser. After that, another ITK program is needed to read and export stylesheet names to XML file from the text file. A code snippet below only shows ITK code, the other works are in other parts.

```
//find the dataset based on file name in command parameter
ifail = ERROR_CHECK(AE_find_dataset(xmlFileName[i], &dataset));
//ask real name of this dataset
if(!ifail && dataset != NULLTAG){
      ifail = ERROR_CHECK(AE_ask_dataset_named_refs(dataset,
      &found, &refDataset));
}
//Iterate through all dataset
for(j = 0; j < found && !ifail; j++){
      char    orgFileName[IMF_filename_size_c + 1];
      char    *outputFilePath = NULL;

      ifail = ERROR_CHECK(IMF_ask_original_file_name(refDataset[j],
      orgFileName));
      if(!ifail){
            //append path and file name
            outputFilePath = (char*) MEM_alloc(strlen(outputPath) +
            strlen(orgFileName) + 2);
            *outputFilePath = '\0';

            strcat(outputFilePath, outputPath);
            strcat(outputFilePath, "\\");
            strcat(outputFilePath, orgFileName);
            printf("\nExport file path: %s", outputFilePath);
            //export to XML file
            ifail = ERROR_CHECK(IMF_export_file(refDataset[j],
            outputFilePath));
      }
}
```

Code snippet 8: ITK code for exporting a stylesheet to XML file

Because different Teamcenter versions have different libraries, the ITK programs have to be compiled in different environments. There are three popular Teamcenter versions 8.3, 9.1 and 10.1, thus, each ITK program has three executable files. The application decides which executable file by user selections. The following

command will compile .c file and link ITK with .obj file. Output is an executable standalone .exe file.

```
call compile -DIPLIB=none wf2txt.c
call linkitk -o wf2txt wf2txt.obj
call compile -DIPLIB=none ds2xml.c
call linkitk -o wf2txt ds2xml.obj
```

Code snippet 9: Compile commands for two ITK program

In C#, static method `Utility.ExecuteCommand()` will be called to execute the proper ITK program version for retrieving all workflow names:

```
try{
    //select correct version
    string wf2TxtExecFile = "";
    switch (config.tcVersion){
        case 8:
            wf2TxtExecFile = "wf2txt-tc83.exe";
            break;
        case 9:
            wf2TxtExecFile = "wf2txt-tc9.exe";
            break;
        case 10:
            wf2TxtExecFile = "wf2txt-tc10x64.exe";
            break;
        default:
            wf2TxtExecFile = "wf2txt-tc9.exe";
            break;
        }

//run wf2txt executable file
Utilities.ExecuteCommand(config.tcPath, Utilities.currentDir
+ "\\" + wf2TxtExecFile + " -u=" + profile.username + " -p="
+ profile.password + " -g=" + profile.group + " -f=\"" +
outputFile + "\"");
}
catch (FileNotFoundException ex){
    Console.WriteLine("WF process file not found.");
    Console.WriteLine(ex.ToString());
}
```

Code snippet 10: User selection of Teamcenter versions

4.2.4 XML Parser

Because exporting or importing Teamcenter customizable components such as Preferences, XML Rendering Stylesheets and Queries are formatted as XML files,

XML Parser is needed to parse XML data. .NET Framework supports several built-in XML Parsers, each of which has both advantages and disadvantages. Researches suggest using uses XMLReader which is suitable for reading XML data and XMLDocument which is better for modifying XML data. All of them are included in System.XML library.

Since having efficiency in reading speed, XMLReader can read forward-only whole XML document fast and save memory space. In the application, XMLReader is implemented to compare the preferences and to read Stylesheets data.

The code snippet below shows how Stylesheet XML data is parsed by using XMLReader. Stylesheet names will be added to a `List<string>` before being filtered and exported from Teamcenter datasets.

```
//Setting the XMLReader
XmlReaderSettings settings = new XmlReaderSettings();
settings.IgnoreComments = true;
settings.IgnoreWhitespace = true;
settings.ConformanceLevel = ConformanceLevel.Document;

//Read through all XML elements, find and add XMLRendering
dataset to a list
try{
      using (XmlReader reader = XmlReader.Create(fileStream,
      settings)){
            while (reader.Read()){
                  if (reader.IsStartElement() && reader.NodeType
                  == XmlNodeType.Element){
                        if (reader.Name == "DataSet"){
                        string styleName =
                        reader.GetAttribute("name");
                              if (!styleList.Contains(styleName)
                              && !styleName.Contains(";")){
                                    styleList.Add(styleName);
                              }
                        }
                  }
            }
      }
}
catch (Exception ex){
      throw ex;
}
```

Code snippet 11: Parse Stylesheet XML

This is a part of XML Rendering Stylesheet XML data which upper code can read:

```
<DataSet id="id95" name="MaterialRevisionMasterForm" accessRefs="#id3"
version="2" memberRefs="#id96" type="XMLRenderingStylesheet">
```

Code snippet 12: XML data of a XML Rendering Stylesheet dataset

Because of different structures of XML data of Teamcenter components, different XMLReader configurations must be set. However, the idea is the same.

- The XMLReader reads forward each XML tags by `read()` method.
- XML tag names will be caught by XMLReader.Name property
- `GetAttribute()` method will return attribute value

In the following code snippet, preference XML data will be parsed with preference name, description and values and will be added to a `List<Preference>` before being filtered.

```
if (reader.Name == "preference"){
    //set preference properties by XML value
    Preference pref = new Preference();
    prefList.Add(pref);
    pref.name = reader.GetAttribute("name");
    pref.type = reader.GetAttribute("type");
    pref.disabled = reader.GetAttribute("disabled");
    pref.array = reader.GetAttribute("array");
    List<string> valueList = new List<string>();

    //loop to find all value for preference
    while (!reader.EOF){
      if (reader.Name == ""){
            continue;
}
      //Break the loop if it is the closing tag of preference
      else if (reader.Name == "preference" && reader.NodeType ==
      XmlNodeType.EndElement){
            if (valueList.Count > 0){
                pref.value = valueList;
            }
            break;
      }
      //Add description for preference instance
      else if (read-er.Name.Equals("preference_description") &&
      reader.NodeType != XmlNodeType.EndElement){
            pref.desc = reader.ReadElementContentAsString();
      }
      //Add values for preference instance
      else if (read-er.Name.Equals("value") && reader.NodeType !=
      XmlNodeType.EndElement){
            valueList.Add(reader.ReadElementContentAsString());
      }
      else{
            reader.Read();
      }
    }
}
```

Code snippet 13: Parse Preference XML

And Preference XML data is parsed by the code:

```
<preference name="Duplo_ESaveAs_Drawing Set Revision Master"
type="String" array="true" disabled="false">
    <preference_description></preference_description>
    <context name="Teamcenter">
    <value>object_desc</value>
    <value>wg003_81_remarks</value>
    <value>wg004_01_type</value>
    <value>wg004_63_execution_drawing</value>
    <value>wg006_01_drawingstandard</value>
    <value>wg006_03_quality_instruct</value>
    <value>wg006_05_design_group</value>
    <value>wg006_07_cad_type</value>
    <value>wg006_08_tight_torque</value>
    <value>wg004_61_designed_for</value>
    </context>
</preference>
```

Code snippet 14: XML data of a preference

The XMLDocument is being used for Teamcenter Queries. This case is more complicated because all Queries which is exported by `plmxml_export` command is grouped in only one XML file. By using XMLDocument, each query can be detached from the big XML file and saved into a separated XML file.

```
//Iterate all queryNode in a List
foreach (XmlNode queryNode in queryNodeList){
      XmlDocument doc2 = new XmlDocument();
      doc2.Load(filePath);
      //Add namespace to Namespace Manager
      XmlNamespaceManager nsmgr2 = new
      XmlNamespaceManager(doc2.NameTable);
      nsmgr2.AddNamespace("plmxml_bus",
      "http://www.plmxml.org/Schemas/PLMXMLBusinessSchema");
      XmlNodeList queryNodeList2 =
      doc2.SelectNodes("/plmxml_bus:PLMXMLBusinessTypes/plmxml_bus:Sav
      edQueryDef", nsmgr2);

      //remove all nodes but keeping one node
      foreach (XmlNode queryNode2 in queryNodeList2){
            if (queryNode2.Attributes["name"].Value !=
            queryNode.Attributes["name"].Value){
                  if (queryNode2.ParentNode != null){
                        queryNode2.ParentNode.RemoveChild(queryNode2);
                  }
            }
      }
      //save xml document
      doc2.Save(Path.GetDirectoryName(filePath) + "\\" + queryFileName
      + ".xml");
}
```

Code snippet 15: Parse Query XML

The output of the above code will be XML files corresponding to filtered queries. These exported XML files can be validated and imported back to Teamcenter.

4.2.5 Background Worker

Collecting customizations is a heavy procedure of tasks, such as executing Teamcenter commands, filtering, saving, and creating report. Therefore, this procedure should be taken to a separated thread from the main thread to prevent from freezing UI when executing. .NET provides Background Worker (BW) as a convenient solution for creating a thread job is, which helps implement thread easier and more effective. The BW is a built-in component that belongs to System.ComponentModel namespace in System.dll library.

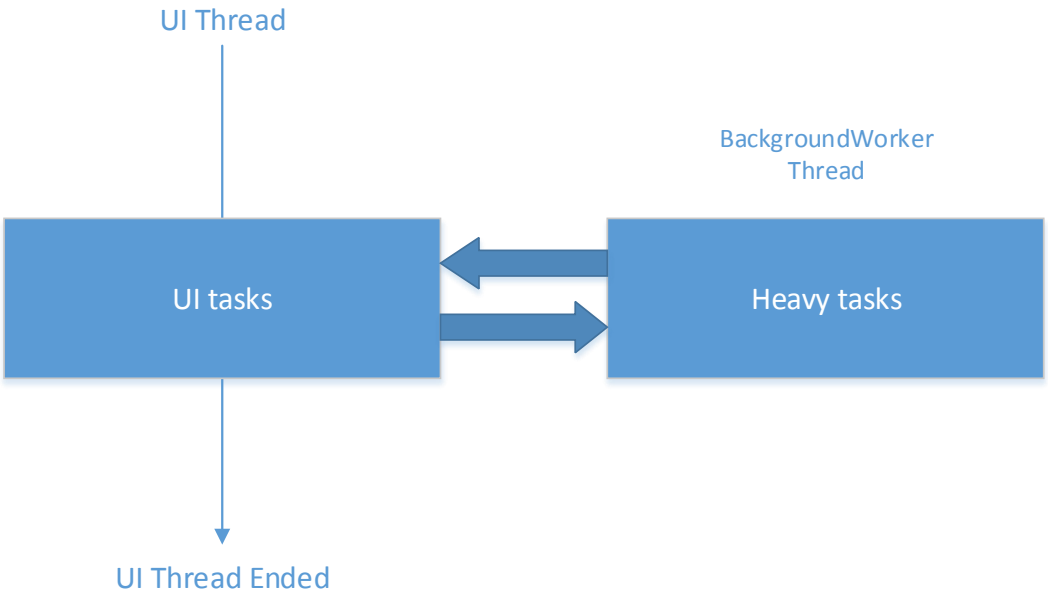


Figure 30: BackgroundWorker concept

A new instance of BW is created and configured to enable a progress report when pressing Start button in GUI:

```
//Configuring new instance of BW
BackgroundWorker mainWork = new BackgroundWorker();
mainWork.WorkerReportsProgress = true;
mainWork.ProgressChanged += new
ProgressChangedEventHandler(mainWork_ProgressChanged);
mainWork.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(mainWork_RunWorkerCompleted);
```

Code snippet 16: BackgroundWorker configuration

After that, according to user option selections, different operations are added to DoWork event handler of this BW instance:

```
foreach (object item in checkedListBox.CheckedItems){
//check which options are selected and add them to BW work
      switch (item.ToString()){
            case dllFilesTxt:
                  mainWork.DoWork += new
                  DoWorkEventHandler(libBW_DoWork);
                  break;

            case JARFilesTxt:
                  mainWork.DoWork += new
                  DoWorkEventHandler(pluginBW_DoWork);
                  break;
            case preferencesSiteTxt:
                  mainWork.DoWork += new
                  DoWorkEventHandler(prefBW_DoWork);
                  break;

            case queriesTxt:
                  mainWork.DoWork += new
                  DoWorkEventHandler(queryBW_DoWork);
                  break;

            case workflowsTxt:
                  mainWork.DoWork += new
                  DoWorkEventHandler(workflowBW_DoWork);
                  break;

            case stylesheetTxt:
                  mainWork.DoWork += new
                  DoWorkEventHandler(stylesheetBW_DoWork);
                  break;

            case templateTxt:
                  mainWork.DoWork += new
                  DoWorkEventHandler(templateBW_DoWork);
                  break;
      }
}
```

Code snippet 17: Add BW event handler

To start running BW, method `RunWorkerAsync()` need to be called.

In each work, BW reports progress and update to UI. There are a progress bar and a status label in the GUI that inform users about progress and status of current work.

The following is a code snippet about reporting work to UI when `ReportProgress()` is invoked.

```
void mainWork_ProgressChanged(object sender,
ProgressChangedEventArgs e)
{
      //update status, update progress bar
      string statusValue = e.UserState as String;
      toolStripStatusLabel.Text = statusValue;
      progressBar.Value += e.ProgressPercentage;
}
```

Code snippet 18: BW ProgressChanged event handler

When the BW completes, event RunWorkerCompleted will be called whether the work is done, cancelled or an error happening. To handle all the cases, a parameter Error is needed to be checked. If the BW finishes successfully, UI will be updated to notice users and do the rest of work.

```
void mainWork_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e){
      //Error handle if BW got error
      if (e.Error != null)
      {
            MessageBox.Show(e.Error.Message, "Error:",
            MessageBoxButtons.OK, MessageBoxIcon.Error,
            MessageBoxDefaultButton.Button1);
            toolStripStatusLabel.Text = e.Error.Message;
      }
      //Work is cancelled
      else if (e.Cancelled)
      {
            toolStripStatusLabel.Text = "Cancelled exporting!";
      }
      //Work is done successfully
      else
      {
            progressBar.Value += 1;
      }
}
```

Code snippet 19: BW RunWorkerCompleted event handler

4.2.6 HTML Report

HTML is chosen for the report because its outlook is simple and convenient to navigate through. Customizations are divided into tables and attached by anchors. After the BW thread completes, a HTML file is created by `StreamWriter` and opened automatically.

```
string htmlFile = "EnvironmentReporter.html";
//open a stream to write a file
using (StreamWriter sw = new StreamWriter(htmlFile))
{
      sw.WriteLine("<html> <head>");
      sw.WriteLine("<title>Environment Reporter</title>");
}
```

Code snippet 20: Write HTML file

Each collected customization is attached to the report and illustrated by a table. A table displays the information of the customization.

- DLL and JAR table display name, path and last modified date.
- Workflow, XML Rendering Stylesheet, Query display name, and file name.
- Preference table display name, description and value.
- Data model template table display only name.

```
private static void createTabletoHtml(string filetype,
StreamWriter sw, string[] header, List<string[]> values)
{
       int counter = values.Count();
       //opening tags of a HTML table
       sw.WriteLine("<table border=\"1\">");
       sw.WriteLine("<tr><h2>" + counter + " <a id= " +
       filetype.Replace(" ", "") + ">" + filetype + "
       </tr></a></h2>");

       //create header of table
       sw.WriteLine("<tr>");
       foreach (string hValue in header)
       {
           sw.WriteLine("<td><p>" + hValue + "</p></td>");
       }
       sw.WriteLine("</tr>");

       //create content of each cell on each row
       foreach (string[] row in values)
       {
           sw.WriteLine("<tr>");
           foreach (string value in row)
           {
               sw.WriteLine("<td>" + value + "</td>");
           }
           sw.WriteLine("</tr>");
       }
       //closing tags
       sw.WriteLine("<table>");
       sw.WriteLine("<br/>");
}
```

Code snippet 21: Write HTML table

Last but not least, in order to have a good looking report, a CSS file is attached to HTML to define and style the entire elements of the report.

4.2.7 Zip Package

One of the most important things is packing a zip package of all extracted customizations. After finishing collecting, customization files are copied or exported to one folder. 7zip is a free software that supports creating zip package from command line.

```
7za.exe a -tzip destination source
```

Code snippet 22: 7zip command

If users not only need a report, but a file package also, a dialog will be opened and ask for the saving location after the BW completes.

```
//configure save dialog with only .zip extension allowed
SaveFileDialog sfd = new SaveFileDialog();
sfd.Filter = "Zip file (*.zip) | *.zip";
//open dialog and ask for a file name
if (sfd.ShowDialog() == DialogResult.OK)
{
      try
      {
          string fileName = sfd.FileName;
          toolStripStatusLabel.Text = "Making zip package...";
          //delete the old file if exists
          if (File.Exists(fileName))
              File.Delete(fileName);
          string zipMaker = Utilities.currentDir + "\\7za.exe";
          //run 7zip command to create zip package
          Utilities.ExecuteCommand(zipMaker + " a -tzip  " +
          fileName + " " + customizedFolder);
      }
      catch (Exception ex)
      {
          MessageBox.Show(ex.Message);
      }
}
```

Code snippet 23: Make zip package

In order to use the command, the 7zip command line version needs to be downloaded from http://www.7-zip.org/download.html and place to same folder as executable file of Environment Reporter. Thanks to great help of 7zip, the zip package will be delivered to the users after command completes.

5 TESTING

From the first developing steps to further, Teamcenter Environment Reporter was tested in various system environments and situations. During testing, bugs and errors are identified and fixed. On the other hand, enhancements are also made to improve the performance. A testing plan is prepared to ensure this software meets actual requirements.

5.1 Testing Environments

Teamcenter Environment Reporter was tested in three below environments:

- Teamcenter 8.3, Windows Server 2003, VMware
- Teamcenter 9.1, Windows Server 2008, Virtual Box
- Teamcenter 10.1, Windows Server 2008, Virtual Box

5.2 Features to be tested

- User Interface must be verified for ease of navigation and use.
- Collecting, filtering DLL, JAR, Workflow, Query, XML Rendering Stylesheet, Preference, and Data model template names must work properly.
- ITK standalone programs for extracting individual Workflow names and exporting XML Rendering Stylesheet dataset must work properly.
- Creating HTML file as final report must be opened and readable.
- Packing all collected customizations into a zip package that is readable.

5.3 Test strategy

This describes all approach methods for testing the application. The application was tested at unit and system level. In addition, the application was also tested through regression and performance tests.

5.3.1 Unit Test

For each individual function, a unit test can verify the code's behavior and its result for specific correct or incorrect situations. Visual Studio provides Test Explorer

with Unit testing framework that supports developers effectively when they test their code. /14/

Each functionality of Environment Reporter is tested and debugged by Unit test to ensure all code's behavior works as expectation.

Below is code snippet for unit testing of `report()` function of JarProcess as an example.

```
[TestMethod()]
public void reportTest()
{
	//Setting Teamcenter configuration
	Profile profile = new Profile("infodba", "infodba", "dba");
	EnvironmentVariable ev = new
	EnvironmentVariable(@"E:\Siemens\TC91x32",
	@"E:\Siemens\TC91x32\TC91x32tcdata",
	@"E:\Siemens\TC91x32\bin", @"E:\Siemens\TC91x32\BNL");
	List<string> keyword = new List<string>();
	keyword.Add("*");
	Configuration config = new Configuration(9, profile, ev,
	@"E:\Siemens\TC91x32\tc_menu\TcEng_WTC91x32.bat", keyword,
	true);
	String currentDir = @"E:\Test";

	//Test JarProcess by exporting a List
	JarProcess target = new JarProcess();
	target.readConfig(config, "", "", currentDir);
	List<string[]> actual = target.report();

	//Test if the List is null or not
	Assert.IsNotNull(actual);
}
```

Code snippet 24: Unit Test code

5.3.2 System Test

The entire application is tested in this level. It verifies all application components are integrated and meets the requirements. Teamcenter Environment Reporter is tested as a whole in different environments. In this test, a problem is found whenever ITK code is not compatible with different versions of Teamcenter. Then the solution is re-compiling ITK code under proper environments and give users a combo box to select in GUI.

5.3.3 Regression Test

From the beginning steps of development, Teamcenter Environment Reporter received comments to have changes in specifications. In order to avoid conflicts between functionalities or the changes affect other areas, regression tests are conducted whenever a change is made.

These are major changes have been done and verified by regression tests:

Table 3: Major changes

Date	**Change**
16.07.2014	Change event handlers of Background Worker
06.08.2014	Change XML Parser for reading preference
07.08.2014	Change table format in HTML report
14.08.2014	Add default excluded keywords
30.09.2014	Add data model template as an option

5.3.4 Performance Test

This performance testing verifies the response times for reaching the requirement or not. The application performance is tested in three conditions:

- Fresh case: new installation of Teamcenter, no customization is made.
- Normal case: few customization on Teamcenter.
- Worse case: heavy customization (on Wärtsilä environment image).

The result of the test is evaluated to be positive even in the worst case.

6 CONCLUSION

The project achieved its main objectives. During the application development period, many changes in filtering logic took place to adjust the application for collecting more accurate customizations.

An HTML report with collected customization information and a zip package containing these customizations are delivered to users in the end of entire process, when the application completes.

The biggest limitation is that the result is restricted by user keywords. This problem becomes even more difficult to be solved in an efficient way due to differences in Teamcenter versions. Therefore, the filtering logic is still being improved for getting better results.

Another limitation is that the application has not been tested in a production environment, which requires permissions when testing. Instead it has been tested in a local copied images of environments which does not require any permission.

One improvement, which can be made in the future could be one optional functionality that categorizes customizations into projects. With this, users can evaluate needed resources for the project work easier and faster when reviewing the final report.

Developing this application provided a chance to improve programming skills and get a better understanding of Teamcenter, which is essential for pursuing career in IDEAL PLM in the future.

7 REFERENCES

/1/ Company Info. Accessed 12.10.2014
http://www.ideal.fi/en/info/company-info/

/2/ Overview of Teamcenter. Accessed 18.12.2014
https://support.industrysoftware.automation.siemens.com/docs/teamcenter/9.1/help/index.html#uid:index_plm00002:id254816:c01a0001

/3/ Overview of customization. Accessed 18.12.2014
https://support.industrysoftware.automation.siemens.com/docs/teamcenter/9.1/help/index.html#uid:index_plm00003:id239801:why_customize

/4/ ITK Function Reference. Accessed 12.10.2014
https://support.industrysoftware.automation.siemens.com/docs/teamcenter/10.1/help/en_US/custom/ITKFunction/index.html

/5/ Client Customization Programmer's Guide. Access 15.10.2014
https://support.industrysoftware.automation.siemens.com/docs/teamcenter/9.1/PDF/pdf/client_customization_programmers_guide.pdf

/6/ Workflow Designer Guide. Accessed 17.10.2014
https://support.industrysoftware.automation.siemens.com/docs/teamcenter/9.1/PDF/pdf/workflow_designer.pdf

/7/ Query Builder Guide. Access 17.10.2014
https://support.industrysoftware.automation.siemens.com/docs/teamcenter/9.1/PDF/pdf/query_builder.pdf

/8/ Utilities Preference. Accessed 17.10.2014
https://support.industrysoftware.automation.siemens.com/docs/teamcenter/9.1/PDF/pdf/utilities_reference.pdf

/9/ .NET Framework Conceptual Overview. Accessed 21.10.2014
http://msdn.microsoft.com/en-us/library/vstudio/zw4w595w%28v=vs.100%29.aspx

/10/ C# Language and the .NET Framework. Accessed 21.10.2014
http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx

/11/ Windows Forms Overview. Accessed 21.10.2014
http://msdn.microsoft.com/en-us/library/8bxxy49h%28v=vs.110%29.aspx

/12/ HTML & CSS. Accessed 26.10.2014
http://www.w3.org/standards/webdesign/htmlcss

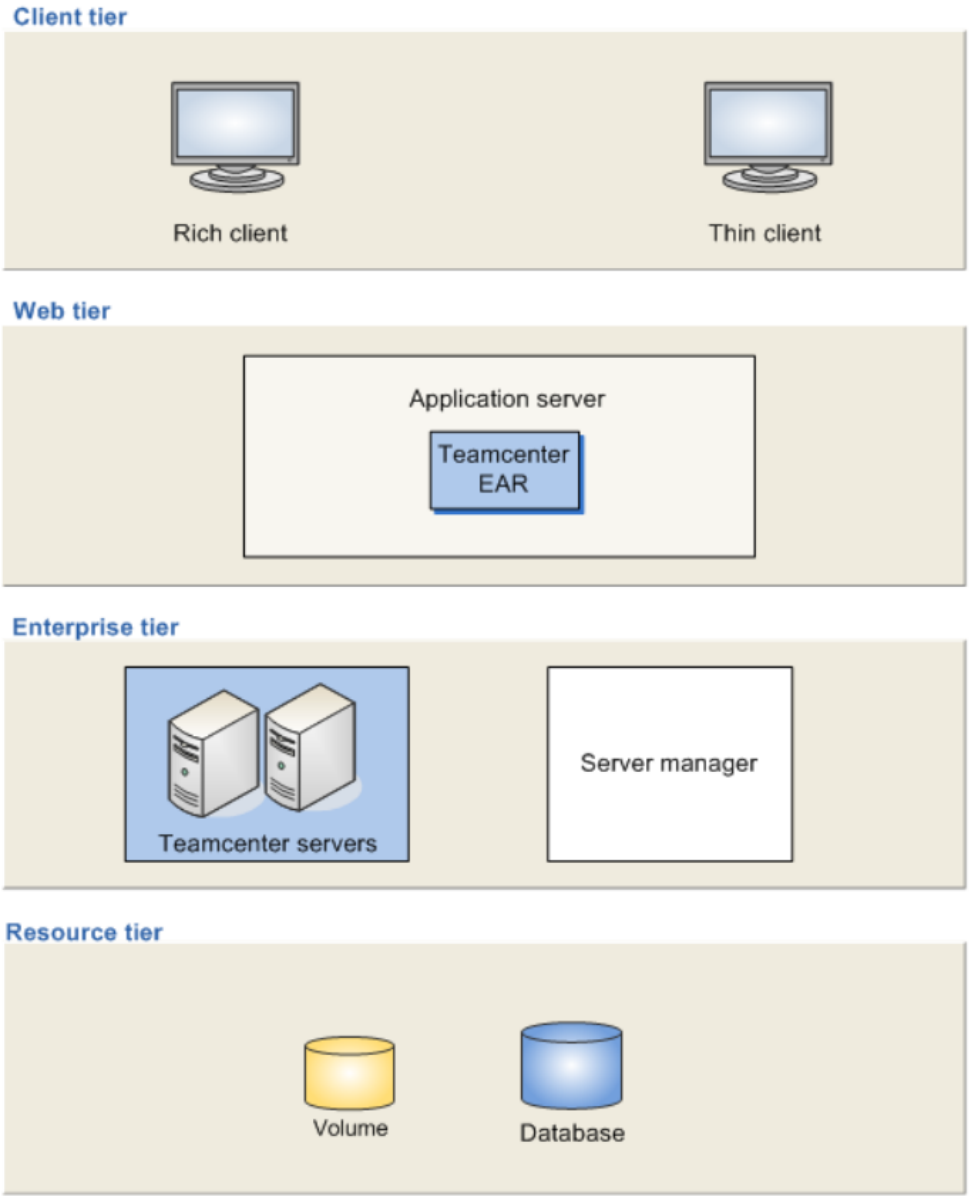
/13/ XML Essentials. Accessed 27.10.2014
http://www.w3.org/standards/xml/core

/14/ Unit Test Framework. Accessed 10.11.2014
http://msdn.microsoft.com/en-us/library/ms243147%28v=vs.80%29

APPENDICES

1. Teamcenter Architecture

Teamcenter 4-tier layers can be describes in the following figure



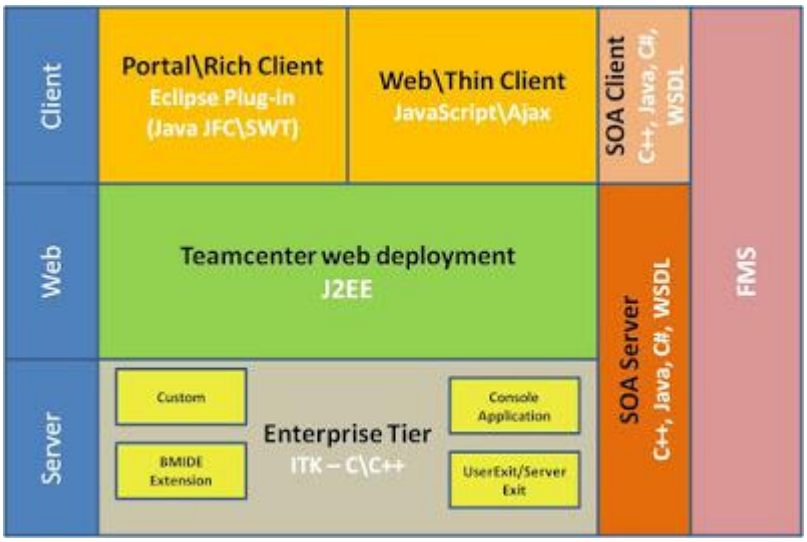
Teamcenter 4-tier architecture is divided into 4 layers:

- Client layer includes a thin client and a rich client.
- Web layer is a Java application that runs in Java 2 Enterprise Edition application server.

- Enterprise (server) comprises a server manager. This layer retrieves and stores data in the database.
- Resource layer comprises database, volumes and file server.

2. Customization Teamcenter Architecture

The below figure shows how the customizations are designed in Teamcenter



Customization Architect of Teamcenter can be divided into 3 layers:

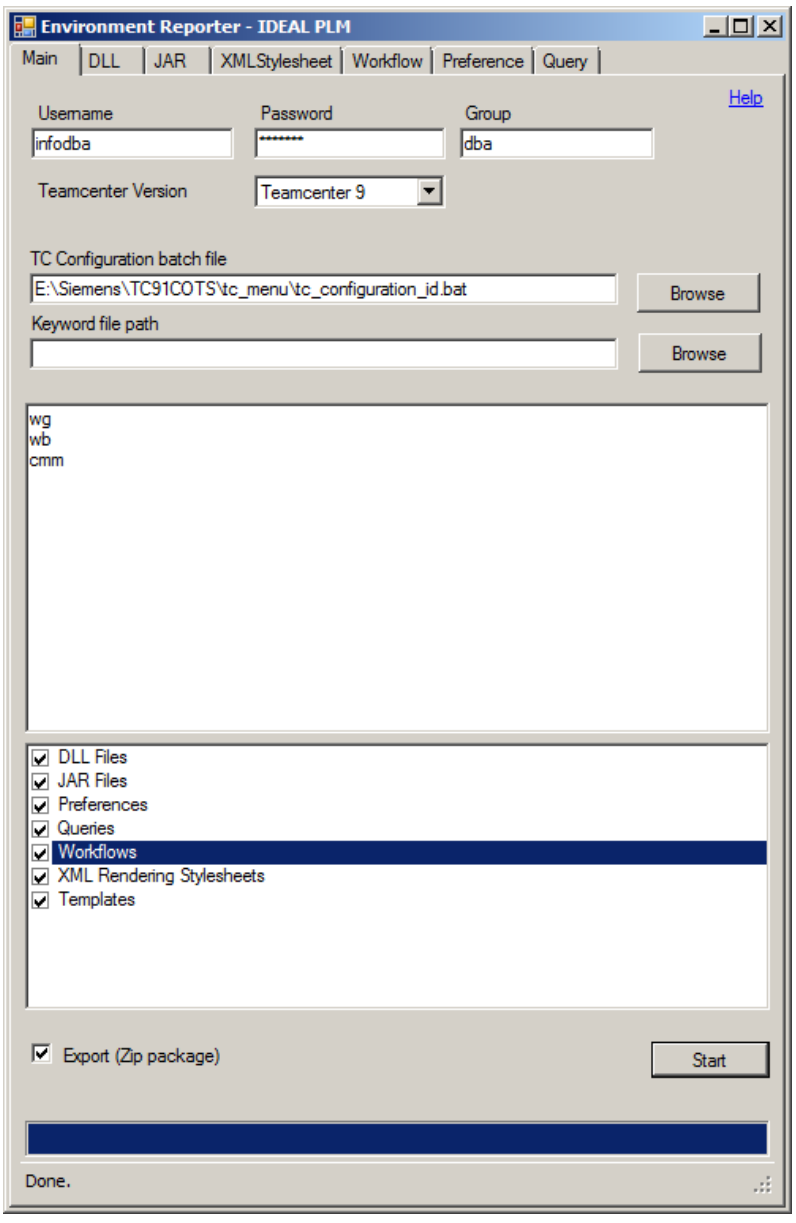
- Client layer: Rich client (UI and data handling) can be customized by Eclipse plugin (written in Java).
- Web layer runs in a Java 2 Enterprise Edition. This layer is used to communicate between Server and Client.
- Server layer: Business logic is defined in this layer. Server customization can be done in this layer with ITK, a Teamcenter API.

3. Running the application on the environment:

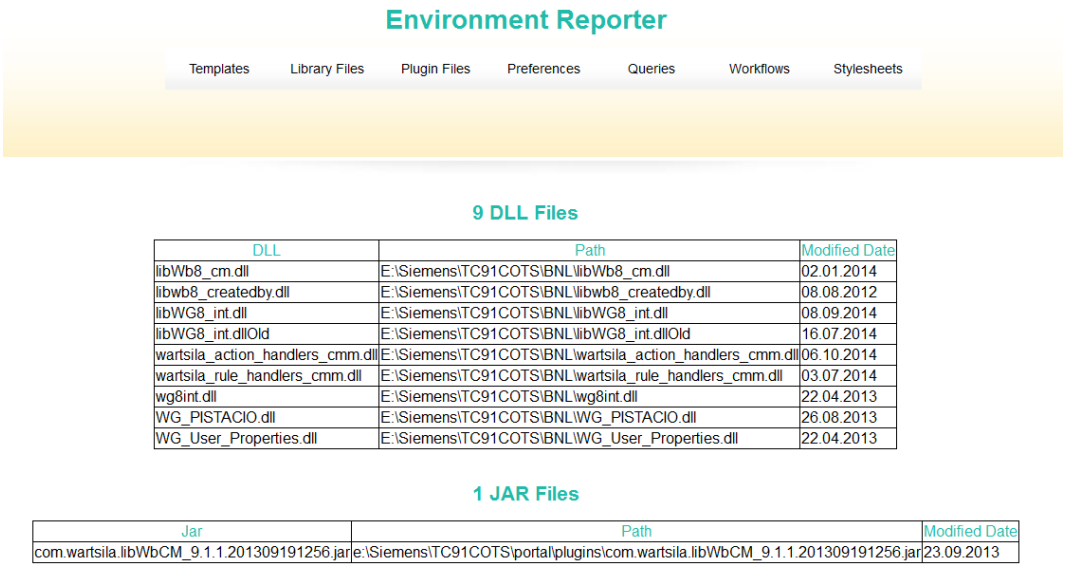
Environment settings:

- Teamcenter 9.1.2.5
- Windows Server 2008
- .NET Framework 4.0
- Firefox 34.0.5

Start the application.



After the work is done, HTML Report is automatically opened.



Environment Reporter

Templates | Library Files | Plugin Files | Preferences | Queries | Workflows | Stylesheets

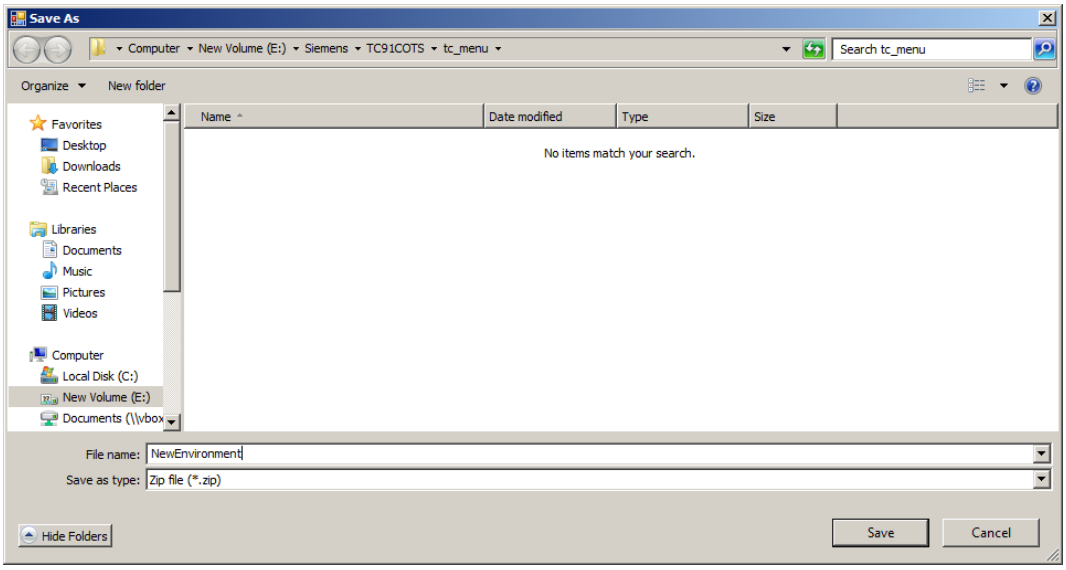
9 DLL Files

DLL	Path	Modified Date
libWb8_cm.dll	E:\Siemens\TC91COTS\BNL\libWb8_cm.dll	02.01.2014
libwb8_createdby.dll	E:\Siemens\TC91COTS\BNL\libwb8_createdby.dll	08.08.2012
libWG8_int.dll	E:\Siemens\TC91COTS\BNL\libWG8_int.dll	08.09.2014
libWG8_int.dllOld	E:\Siemens\TC91COTS\BNL\libWG8_int.dllOld	16.07.2014
wartsila_action_handlers_cmm.dll	E:\Siemens\TC91COTS\BNL\wartsila_action_handlers_cmm.dll	06.10.2014
wartsila_rule_handlers_cmm.dll	E:\Siemens\TC91COTS\BNL\wartsila_rule_handlers_cmm.dll	03.07.2014
wg8int.dll	E:\Siemens\TC91COTS\BNL\wg8int.dll	22.04.2013
WG_PISTACIO.dll	E:\Siemens\TC91COTS\BNL\WG_PISTACIO.dll	26.08.2013
WG_User_Properties.dll	E:\Siemens\TC91COTS\BNL\WG_User_Properties.dll	22.04.2013

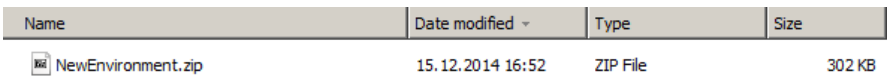
1 JAR Files

Jar	Path	Modified Date
com.wartsila.libWbCM_9.1.1.201309191256.jar	e:\Siemens\TC91COTS\portal\plugins\com.wartsila.libWbCM_9.1.1.201309191256.jar	23.09.2013

Saving zip package dialog appear to ask users for name of the package.



Open and extract the zip package.



Name	Date modified	Type	Size
NewEnvironment.zip	15.12.2014 16:52	ZIP File	302 KB

Name	Date modified	Type
Libraries	15.12.2014 16:55	File folder
Plugins	15.12.2014 16:55	File folder
Preferences	15.12.2014 16:55	File folder
Queries	15.12.2014 16:55	File folder
Stylesheets	15.12.2014 16:46	File folder
WorkFlows	15.12.2014 16:55	File folder

Verify files inside Libraries folder with the report

Name	Date modified	Type	Size
libWb8_cm.dll	2.1.2014 12:12	Application extension	21 KB
libwb8_createdby.dll	8.8.2012 12:20	Application extension	21 KB
libWG8_int.dll	8.9.2014 10:46	Application extension	143 KB
libWG8_int.dllOld	16.7.2014 9:02	DLLOLD File	140 KB
wartsila_action_handlers_cmm.dll	6.10.2014 15:43	Application extension	122 KB
wartsila_rule_handlers_cmm.dll	3.7.2014 13:29	Application extension	118 KB
WG_PISTACIO.dll	26.8.2013 5:28	Application extension	46 KB
WG_User_Properties.dll	22.4.2013 6:06	Application extension	22 KB
wg8int.dll	22.4.2013 6:06	Application extension	53 KB

Verify files inside Plugins folder with the report

Name	Date modified	Type	Size
com.wartsila.libWbCM_9.1.1.201309191256...	23.9.2013 9:21	JAR File	80 KB

Verify files inside Workflows folder with the report

Name	Date modified	Type	Size
CMM Delete Status.xml	15.12.2014 16:45	XML Document	7 KB
CMM Errors.xml	15.12.2014 16:45	XML Document	12 KB
CMM Gather Objects for Migration.xml	15.12.2014 16:45	XML Document	12 KB
CMM Migration Approved.xml	15.12.2014 16:45	XML Document	49 KB
CMM Migration Initial.xml	15.12.2014 16:45	XML Document	6 KB
CMM Migration.xml	15.12.2014 16:45	XML Document	27 KB
CMM ReMigration.xml	15.12.2014 16:45	XML Document	20 KB
CMM Simple Migration.xml	15.12.2014 16:45	XML Document	21 KB
CMMSortSub.xml	15.12.2014 16:45	XML Document	32 KB
QuickCMM_Done.xml	15.12.2014 16:46	XML Document	3 KB