VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Burcu Ser

NX 2D DRAWING FIX TOOL

Information Technology
2016

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Burcu Ser
Title	NX Drawing Fix Tool
Year	2016
Language	English
Pages	74
Name of Supervisor	Ghodrat Moghadampour

NX Software is an integrated product design, engineering and manufacturing solution provided by Siemens. It offers a full range of advanced manufacturing applications, integration of multi-discipline simulation and it is considered as the best tightly integrated software with Teamcenter, Product Lifecycle Management platform. NX is the follower of I-deas software which is also used for similar purposes.

The target of this project was to investigate the problems, which occur while migrating designs from I-deas to NX and develop a software, which will modify and adapt the designs NX automatically.

In the project an NX Open API called Drawing Fix tool was developed and integrated to NX solution in order to modify 2D drawings on the NX Drafting application which is a module in NX system. The Drawing Fix tool contains the following functions: functions to change style setting of view, dimension and notes, functions to fix Surface Finish Symbols, Weld Symbols and Identification Symbols, functions to remove I-deas title block, custom symbols and to save data from the title block and a function to hide the points.

The Drawing Fix tool was implemented by using NX Open libraries with C# programming language while using NX Block UI Styler and Microsoft Windows Form library for user interfaces. The tool runs on NX Drafting module and while the tool is running in the background the designer is informed on the fixing process through pop up information window.

All the objectives of the project were achieved and the Drawing Fix Tool was presented to two different customers of IDEAL PLM and it is currently in the production phase.

Keywords I-deas, NX Open API, Drafting Application, Teamcenter, Drawing Fix Tool, Product Life Cycle

Preface

This project was started in IDEAL PLM based on research on issues that occur during the migration process from I-deas to NX. The target of the project was to find problems appearing on the NX drawings during this process and develop a software tool to fix them automatically. As a final result a software was delivered to the customers resolving the discovered problems successfully. The project was started in July 2015 and it was published as a bachelor's thesis at Vaasa University of Applied Sciences in February 2016.

My colleagues from IDEAL PLM contributed to this project in so many ways. Therefore I would firstly like to thank my supervisor Olli Väätäinen and I would like to extend my thanks to all my colleagues for all his support and motivation.

I also would like to thank my supervisor Ghodrat Moghadampour from VAMK for his academic support and encouragement.

Finally I wish to express my gratitude to my big family and all friends who never withheld their supports.

Vaasa, February 2016
Burcu Ser

TABLE OF CONTENTS

1 INTRODUCTION 10
1.1 Problem statement 10
1.2 Objective of the Project 11
1.3 Business scale of the project 12
1.4 Software Development Model 12
2 RELEVANT TECHNOLOGIES 13
2.1 NX SOFTWARE 13
2.1.1 NX Drafting Application 13
2.1.2 NX Block UI Styler 13
2.2 NX Open API 14
2.3 Teamcenter 14
2.3.1 Product Data Control (Design) 14
2.3.2 Product Document Control 15
2.3.3 Product Structure Control 15
2.3.4 Process Control 15
2.4 .NET Framework and C# Programming language 15
2.5 Oracle VM Virtual Box 16
3 DRAWING FIX TOOL 17
3.1 Requirements 17
3.2 Classes and Their Relationships 19
3.2.1 Current Application Class 20
3.2.2 DrawingFix Class 20
3.2.3 Config Class 21
3.2.4 FixStyle Class 22
3.2.5 FixSymbol Class 22
3.2.6 Attributes Class 23
3.2.7 RemovePartListAndTitleBlock Class 23
3.2.8 AttributeConfirmation Class 24
3.2.9 Hidepoints Class 25
3.3 Detailed Functional Description 26

3.3.1 Fix Style 26
3.3.2 Fix Symbols 31
3.3.3 Remove I-deas TitleBlock and Frames 36
3.3.4 Hide Points 40
4 GUI DESIGN 42
4.1 Change on NX UI 42
4.2 Drawing Fix Main GUI 43
4.3 Confirmation Window 44
5 IMPLEMENTATION AND UNIT, FUNCTIONAL, SYSTEM INTEGRATION TESTING 46
5.1 Class Implementing Logic 46
5.2 Iterations 47
5.2.1 Iteration 0: Reading configuration file 47
5.2.2 Iteration 1: Creating Main Window and Integrating to NX UI 48
5.2.3 Iteration 2: Adding Fix Style Functionality: View Style Fixing 51
5.2.4 Iteration 3: Adding Fix Style Functionality: Dimension Style Fixing 53
5.2.5 Iteration 4: Adding Fix Style Functionality: Notes Fix Style 56
5.2.6 Iteration 5: Fix Symbol: Surface Finish Symbol fixing 58
5.2.7 Iteration 6: Fix Symbol: Weld Symbols Fix 60
5.2.8 Iteration 7: Fix Symbol: Identification Symbols Fix 62
5.2.9 Iteration 8: Gathering Data from Title Block 64
5.2.10 Iteration 9: Removing TitleBlock and Frames 68
5.2.11 Iteration 10: Hiding Points 69
6 TESTING 71
6.1 Acceptance Testing 71
6.2 Alpha Testing 71
6.3 Beta Testing 71
7 CONCLUSION 72
7.1 Future Work 72
REFERENCES 73

LIST OF FIGURES

Figure 1. Agile Methodology /3/ 12
Figure 2. Teamcenter Design View /9/ 14
Figure 3. Simple workflow sample /10/ 15
Figure 4. The Application required functions 19
Figure 5. Application Classes and their relationships 20
Figure 7. CurrentApplication Class 20
Figure 6. DrawingFix Class 21
Figure 8. Config Class 21
Figure 9. FixStyle Class 22
Figure 10. FixSymbol Class 22
Figure 13. Attributes Class 23
Figure 11. RemovePartListAndTitleBlock Class 24
Figure 12. AttributesConfirmation Class 25
Figure 14. HidePoints Class 25
Figure 15. Design of Fix Style functionality 26
Figure 16. View from NX, setting window 27
Figure 17. Information Window while fixing the views 27
Figure 18. Shows Tolerance Types and Values 28
Figure 19. Tolerance Text Values 28
Figure 20. Dimesion Text Values 29
Figure 21. Appended Text Values 29
Figure 22. Appended Text Values 30
Figure 23. Caption from Information Window, shows dimensions updates 30
Figure 24. Lettering Values 31
Figure 25. Information Window while fixing the notes 31
Figure 26. Implementation Design of Fix Symbols functionality 32
Figure 27. Values for Surface Finish Symbols 32
Figure 28. The Surface Finish Symbol before and after running Drawing Fix Tool. 33
Figure 29. Information Window while fixing the Surface Finish Symbols 33

Figure 30. Values for Weld Symbols 34
Figure 31. Weld symbols before and after running the Fix Style. 34
Figure 32. Information Window shows Weld Symbols Fixes. 34
Figure 33. Identification Symbol setting Values 35
Figure 34. An Identification Symbols before and after running Drawing Fix Tool. 35
Figure 35. Information Window showing Identification Symbols fixes. 36
Figure 36. Implementation Design of Remove Idea-s Title Block and Frames functionality 36
Figure 38. Snippet from a titleblock shows "Made" row. 38
Figure 39. Snippet from a titleblock shows design info. 38
Figure 40. Attributes Confirmation Window. 39
Figure 41. Shows properties of part. 39
Figure 43. Before and after removing titleblock and frames 40
Figure 44. Information Window display data related the Remove I-deas TitleBlock and Frames. 40
Figure 45. Implementation Design of Hide Points functionality 41
Figure 47. Capture from NX, displays Hide command. 41
Figure 48. Capture from NX, NX tool bar. 43
Figure 49. Capture from NX, Drawing Fix main GUI. 43
Figure 50. Confirmation Window 45
Figure 52. Creation of Main UI 48
Figure 53. Code generation properties. 49
Figure 54. UI testing, button response. 50
Figure 55. Captures from NX to show result Gathering Data from title block for test case 1, MaterialID row is used as a sample 66
Figure 56. Captures from NX to show result Gathering Data from title block for test case 2, MaterialID row is used as a sample 66
Figure 57. Captures from NX to show result Gathering Data from title block for test case 3, MaterialID row is used as a sample 67

LIST OF CODE SNIPPET

Code Snippet 1. Snippet from the Configuration file, gathering data from the titleblock..... 37
Code Snippet 2. Capture from configuration file, symbols to remove. 39
Code Snippet 3. Capture from configuration file, list of custom symbols to hide points. 41
Code Snippet 4. Customizing the toolbar, tbr file 42
Code Snippet 5. Callback function for NX Block UI Design..... 46
Code Snippet 6. Method to read configuration file 47
Code Snippet 7. Main function, activates the application 49
Code Snippet 8. GetUnloadOption function, at termination. 51
Code Snippet 9. GetUnloadOption function, immediately. 51
Code Snippet 10. RunFixStyle method, fixViewStyle activation. 51
Code Snippet 11. fixViewStyle method 52
Code Snippet 12. RunFixStyle method, fixDimensionStyle activation. 53
Code Snippet 13. fixDimensionStyle merhod. 54
Code Snippet 14. Appended Text check on dimensions 56
Code Snippet 15. RunFixStyle method, fixNoteStyle activation. 56
Code Snippet 16. fixNoteStyle method. 57
Code Snippet 17. RunFixSymbol method, fixSurfaceFinishSymbol activation. 58
Code Snippet 18. fixFinishSymbols method. 59
Code Snippet 19. RunFixSymbol method, fixWeldSymbol activation. 60
Code Snippet 20. fixWeldSymbol method. 61
Code Snippet 21. RunFixSymbol method, fixIdebtificationSymbol activation. . 62
Code Snippet 22. fixIdentificationSymbol method. 64
Code Snippet 23. runRemovePartListAndTitleBlock method. 64
Code Snippet 24. Checking TitleBlock. 65
Code Snippet 25. Checking the attributes. 65
Code Snippet 26. Implementation of confirmation window options. 68
Code Snippet 27. removeCustomSymbols method. 69
Code Snippet 28. runHidePoints method..... 70
Code Snippet 29. HidePoints method 70

LIST OF TABLES

Table 1. NX Versions and Compilers Platforms 16

ABBREVIATIONS

API	Application programming Interface
TC	TEAMCENTER a Siemens PLM application
BOMs	Bill of Materials
CAD	Computer-Aided design
CAM	Computer Aided Manufacture
CAE	Computer Aided Engineering
I-DEAS	a Siemens CAD application
NX	a Siemens CAD application
PDM	Product Data Management
PLM	Product Lifecycle Management
DLL	Dynamic Link Library
UI	User Interface
GUI	Graphical User Interface
XML	Extensible Markup Language

1 INTRODUCTION

In 2001, Siemens PLM Solution started implementing a plan in order to move I-deas customers to NX and this plan included ending I-deas development and support in 2015. Therefore many Siemens customer started to migrate their designs from I-deas to NX. IDEAL PLM, as representative of Siemens in Finland and Russia, is working with many big and small-sized companies during the migration process. /1/

During the migration process, companies are facing many different problems regarding designs and drawings. After investigating difference cases, it appears that different customers of IDEAL PLM have similar problems. Thereafter a research has started to observe the most common issues and their solutions. During the research, opinions of many NX designers who work actively, from customer side and the company side, are considered.

The research and result of the first test runs indicated that the tool also can be used for different cases since each company work with many different NX designers, and customers. Drawing Fix tool can set views, dimensions, and notes, custom symbols setting to common setting for each drawing and for each customer.

Drawing Fix tool is used for NX8 and NX9 by end of 2015. In this project only NX 9 implementation and test results will be published with no information of IDEAL PLM customers who currently uses the drawing fix tool.

1.1 Problem statement

As a result of the research, the most common problems which must be fixed on the drawing are listed as below:

- Changes to be fixed related to style on drawings after the migration.
    - Changes on views style
    - Changed on dimensions style
    - Changes on notes style
- Changes to be fixed related to Symbols
    - Changes on Surface Finish Symbols
    - Changes on Identification Symbols
    - Changes on Weld Symbols
- Remove points which do not belong in any element of drawing.
- Removing custom symbols such as titleBlock, frames, partlist tables, revision tables and removing these custom symbols. And required data (speci-

fied by customer) of item revision must be stored from titleBlock before it is removed.

- Configuration file must be created to keep data which can be changed or needs to be modified by the NX designer in order to avoid code changes or any re-compiling processes.

Currently these listed problems are solved manually by the NX designer. This process, may take hours and in some cases days since each drawing may have several views, many dimension, many notes, many symbols and since companies has hundreds of different drawings. It is estimated that by using this software the companies could save long working hours and it will avoid repetitive works for the designer. And it appears to be that this may increase the productivity of employees.

1.2 Objective of the Project

One of the main objectives of this project is to expand the research to expose the problems which occurs while or after the I-deas to NX migration process and to develop a tool integrated to NX for fixing the changes/faults on the drawings. And this process must be done with NX default setting such as “customer defaults”, “preference” or “selected object” in order to provide ability of changes for fixing. In this way the software can also be used for other proposes than for fixing the migration faults and it can be easily extensible. Additionally, a configuration file is added to the tool for necessary data modification.

The project also aims to determine daily problems which NX Designers face and repeated actions are done by NX designers to add these functionalities and solution of the problems to the tool in order to avoid waste of time and money while decreasing the possibility of mistakes that may be done by designers.

After applying NX designer opinions, it is decided that the user interface must be as simple as possible and very similar to NX’s own pop-up windows. Therefore one of important aims is to have very simple and user interactions are kept as small as possible.

1.3 Business scale of the project

This project was started as a solution development project in IDEAL PLM. As IDEAL PLM has successfully delivered projects of all levels of complexity while carefully listening to the customers since 1992, the working methodology of this project follows the same way to provide customers processes to save their time, resources and money with quality of IDEAL PLM awarded ISO 9001: 2008 quality certificate. /2/

1.4 Software Development Model

This project was carried out using the agile software development model. Iterations are planned based on four main functionalities and for each iteration every single function is created, integrated and tested one by one. After each test the requirements were open to be discussed and if it is necessary, to be changed.

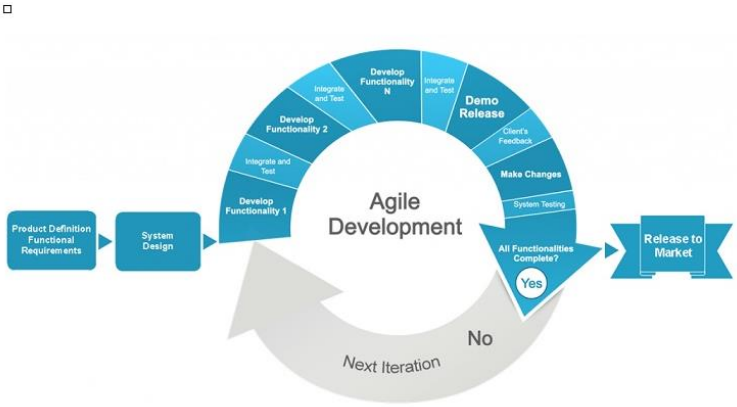


Figure 1. Agile Methodology /3/

2 RELEVANT TECHNOLOGIES

Drawing Fix Tool is designed to be integrated to NX. NX Open API is used with C# programming language for integration and modifying the drawings on NX drafting application. Necessary dialog box, blocks and main interface are generated by using NX Block UI Styler and Windows Forms Applications. In this part all technologies used in this project are explained.

2.1 NX SOFTWARE

NX Software is an integrated product design, engineering and manufacturing solution provided by Siemens. NX offers such a full range of advanced manufacturing applications, integration of multi-discipline simulation and it is considered as the best tightly integrated software with Teamcenter, Product Lifecycle Management platform. /4/

2.1.1 NX Drafting Application

NX Drafting application is used to create engineering drawings from 3D models using drafting tools. NX Drafting application can automatically create drawings view from 3D part and assembly models and designer cab align, scale views and arrange the drawing's sheets. /4/

Drawing Fix tool is only available on NX Drafting application, it fixes errors of these engineering drawings, remove custom symbols off of them and add new custom symbols if it is required.

2.1.2 NX Block UI Styler

Block UI Styler is an application created for NX Users and third-party developers in order to build dialog boxes that are consistent with NX block-based user interface. The dialog boxes can be launched from menu bars and it enables rapid prototyping because of the visual builder and automatic file generation.

Main user interface of our software is generated by using this application in order to work smoothly with NX and have similar an interface to NX. /5/

2.2 NX Open API

NX Open APIs provide an open architecture which can be utilized by third parties, customers and users for creating and integrating custom software. By using this API, the programmers are able to create, access and edit the NX objects. Since this project aims to fix/change object on drawings such as views, custom symbols, notes, dimensions this API allows us to perform this while selecting preferred programming language. /6/

2.3 Teamcenter

Teamcenter is a suite of product lifecycle management software that enables companies digitally manage their product and manufacturing information in context of the product life cycle. Teamcenter providers managing and sharing product designs, documents, Bill of Materials and any data in Teamcenter. And also it provides workflows in order to change and control the process. /7/

2.3.1 Product Data Control (Design)

Users can locate, open, interrogate and mark up engineering created by multiple CAD, CAM, and CAE systems in order to keep data always current and up-to-date when deployed in a Teamcenter environment. /8/

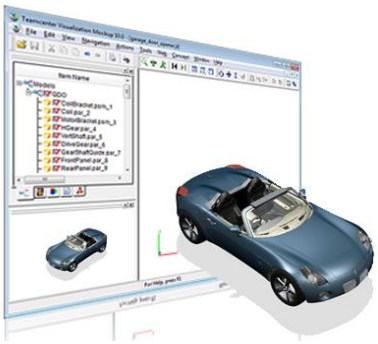


Figure 2. Teamcenter Design View /9/

2.3.2 Product Document Control

Product document control provides management of all product documentation, product planning and testing result in real time through Microsoft Office Word, Excel, PowerPoint, and Outlook. /8/

2.3.3 Product Structure Control

Product Structure Control enables users to define single generic structure from entire range of product variants. Teamcenter has an application called Structure Manager to create, view and modify product structures. Structure managers allows to manage structures that were created in multiple CAD programs such as NX. /8/

2.3.4 Process Control

Teamcenter uses workflows to accomplish an objective. Workflows processes are initiated by users by submitting item to workflow for task such as approvals, reviews and assignments complete work. /8/

Figure 3. Simple workflow sample /10/

2.4 .NET Framework and C# Programming language

NX Open API is built on the .NET framework, therefore in the beginning of this project .NET Framework is chosen in order to take full advantage of all benefits provided by the frameworks. Any .NET compliant language, including Visual Basic .NET and C#, could be chosen for NX Open API part of this project but for the future additions, for instance if the software needs to communicate with Teamcenter, C# is chosen since SOA libraries supports C# programming language.

. /11/

The following table specifies the compiler to use for the applicable platform by Siemens in order to run integrated tool on NX. Since or target versions for NX are NX10, NX9, NX8.5 and NX8 for each version software must be complied with required compiler.

Table 1. NX Versions and Compilers Platforms /12/

NX version	Windows
10	.NET Framework 4.5
9	Visual Studio 2012 Update 1
8.5	Visual Studio 2010 SP1 Build 16.00.40219.01
8	Visual Studio 2010

2.5 Oracle VM Virtual Box

Oracle VM Virtual Box is used in order to create a different environment for the testing and implementation process. As mentioned above, a different version of NX required different compiler platforms and a different Teamcenter version. For this reason, for each customer different virtual boxes are created and used actively. /13/

3 DRAWING FIX TOOL

Since this project has started as result of experiences from different users, all specifications have been collected and as a final result the most common issues are selected and combined. In the very beginning, the problems were known but methods for solutions were based on guesses. Therefore it seemed the best way to use the agile methodology and implement a small part of the software and execute the testing process. In this way the capacity of NX Open API could be seen and all capacity could be used for the project.

3.1 Requirements

The requirements divided into the four part based on the customer's needs. Minimal functions refer to normal requirements, important implicit requirements refer to expected requirements and existing requirements refer to nice to have functionalities.

Normal requirements that the application must fulfill:

- The software must be integrated to NX.9
- The designer must be able to start to software with a button which is added to NX tool bar.
- The user interface must be as simple as possible and user action must be mostly only to activate the functionalities, in rare cases to confirm the data to be removed, saved, searched and giving login information.
- Fixes/Changes must be separated on the user interface, the user must be able to choose only one or all of actions at ones.
    - Fix Style is one of the actions which includes fixing style of views, notes and dimensions.
    - Remove Title Blocks, frames and tables is one of the actions which includes the title block, the frames and the tables.
    - Fix Symbols is one of the actions which includes fixing Weld Symbol fixing, Identification Symbols fixing and Surface Finish Symbols fixing.

    - Hide Points is created to find all points on drawing and removes them. This must have functionality to check if custom symbols have points to remove.
- The software must find all views, dimensions, notes and make the changes on all of them. In case of any issues of changing an object, the user must be informed that changes could not be done for a specific object.
- Name of the custom symbols, title blocks, frames, tables, must be provided by customer then the software must find all and remove from drawing. In case of any issue during the removal process the user must be informed.
- All symbols, Surface Finish Symbols, Weld Symbols, and Identifications Symbols must be found and fixed. In case of any issue during the process the user must be informed.

Expected requirements:

- The designer should be able zoom in and zoom out in order to check the drawing while Drawing Fix Tool user interface is activated.
- Before removing TitleBlock item information should be saved as properties of displayed part.
- Information of process should be printed on the information window with error messages and number of objects.
- The designer should be able to undo all changes made by the Drawing Fix Tool by pressing CTRL-Z only ones, either the designer choose only one action or choose all of them.

Exciting Requirements:

- When TitleBlock is removed, the software needs to get a confirmation from the user while a window is displaying data to be saved and the user should be able to modify the data before it is saved.
- The Configuration file should be used for the data which may need to be modified by NX designer or which can be change for each environment.

Based on requirements, as it is mentioned above, functionalities are collected under four main titles: Fix Style, Fix Symbols, Remove I-deas TitleBlock and Frames and Hide Points. A designer will be able to only activate one or all functionalities, all changes and fixes are executed by the system and the designer will be informed about the process simultaneously from the information window.

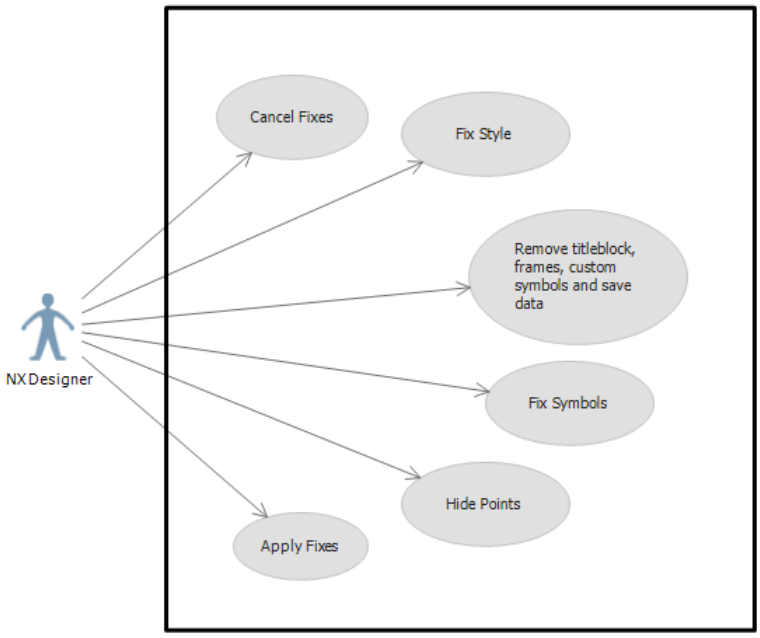


Figure 4. The Application required functions

3.2 Classes and Their Relationships

Drawing Fix Software is designed based on the single responsibility principle in object-oriented programming. Therefore each class has responsibility over a single part of the functionality provided by Drawing Fix. Since the software has 4 main functionalities, each functionality has its relevant class: FixStyle class, FixSymbol class, RemovePartListandTitleblock class and HidePoint class. Additionally the software has Config class responsible for reading data from configuration file, Attributes class to read and write attributes of the part, Current application class to gather all necessary information of current application of NX, Draw-

ingFix class for main user interface, AttributeConfirmation class to create UI in order to get confirmation from the user to remove the frames and TitleBlocks.

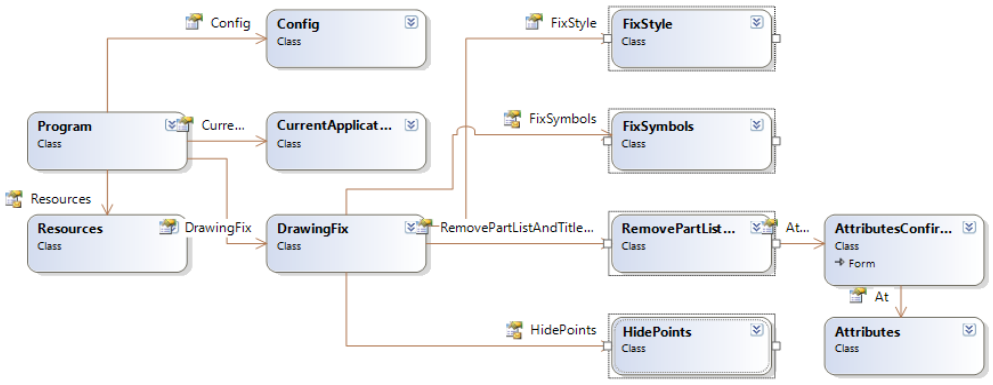


Figure 5. Application Classes and their relationships

3.2.1 Current Application Class

This class is used to get information of the current application. This class is used in main class in order to make application information for every necessary case.

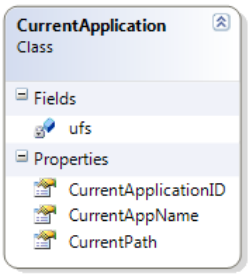


Figure 6. CurrentApplication Class

3.2.2 DrawingFix Class

Drawing Fix class is responsible for activating Drawing Fix user interface and present options to the designer to be able to select between the functionalities.

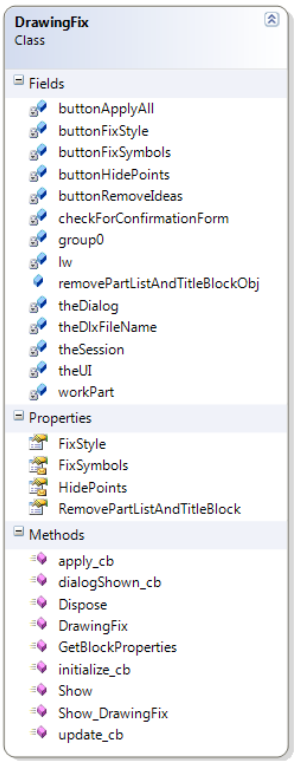


Figure 7. DrawingFix Class

3.2.3 Config Class

Config class is used to read data from configuration file. The data read from by this class are accessible as public methods in every other class.

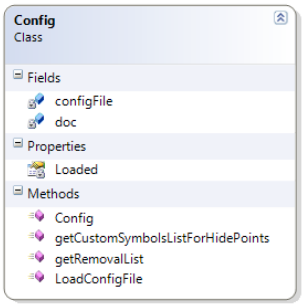


Figure 8. Config Class

3.2.4 FixStyle Class

This class is responsible for managing the Fix Style process. All methods and variable related to Fix Style functionality are in this class.

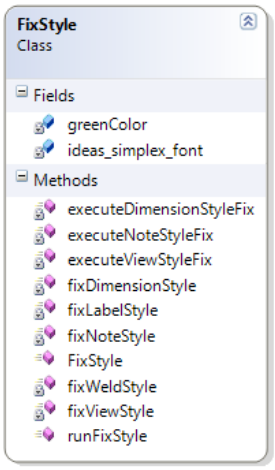


Figure 9. FixStyle Class

3.2.5 FixSymbol Class

This class is responsible for managing the Fix Symbols process. All methods and variables related to Fix Symbols functionality are in this class.

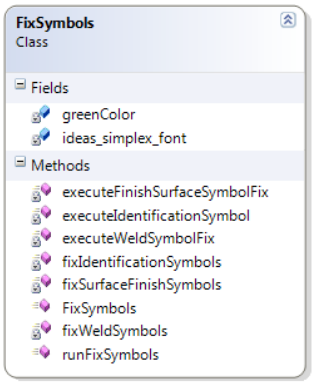


Figure 10. FixSymbol Class

3.2.6 Attributes Class

Currently this class has only a method to save the attributes which are gathered by AttributeConfirmation object. It is kept separated in case in the future, any functionality is added the tool which involve reading attributes or other issues related to current drawing attributes.

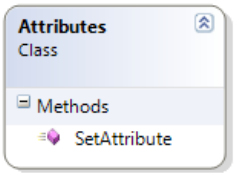


Figure 11. Attributes Class

3.2.7 RemovePartListAndTitleBlock Class

This class is responsible for managing the Remove Part List and Title Blocks process. All methods and variable related to removing the custom symbols are in this class.

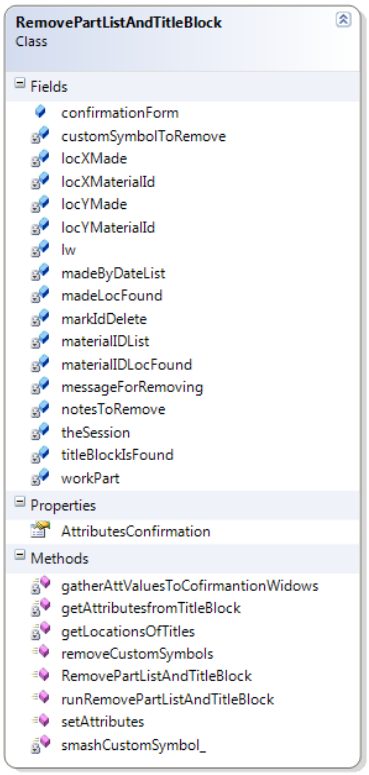


Figure 12. RemovePartListAndTitleBlock Class

3.2.8 AttributeConfirmation Class

This class is used to create dialog which holds the entry of attributes which are going to be saved as the current part's attributes to the drawings. But as mentioned earlier, this action is executed if the designers confirm it. Therefore this class includes a windows form with the options and methods to gather the attributes values.

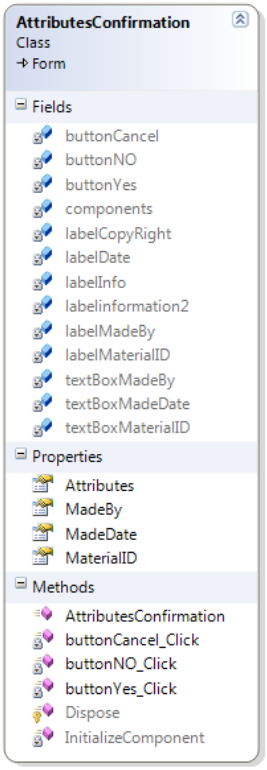


Figure 13. AttributesConfirmation Class

3.2.9 Hidepoints Class

This class is responsible for managing the Hiding Points process. All methods and variables related to Hide Points functionality are in this class.

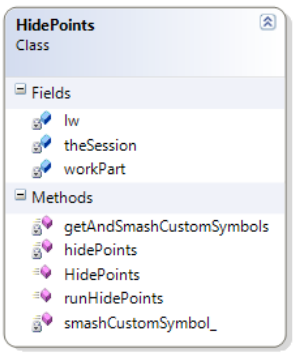


Figure 14. HidePoints Class

3.3 Detailed Functional Description

In this part all functions are explained with their sub-functionalities.

3.3.1 Fix Style

For Fix Style actions, general approach of style fixing is to inherit customer defaults to style setting of views, dimensions and notes. In this way whatever the customer defaults are for different customers will be applied on the drawings. But there are several properties of setting that preferred to be keep in same way as how it is exactly in I-deas or in original drawing therefor before inheriting the customer setting, the software saves these specific values and set back these values after inheritance of the customer defaults.

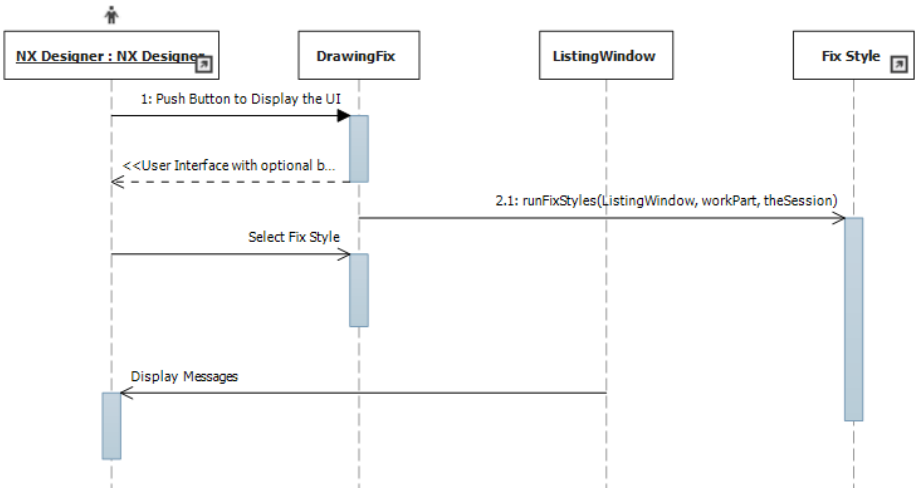


Figure 15. Design of Fix Style functionality

3.3.1.1 View Style

On NX Drafting Application, a part may have multiple sheet and each sheet may have multiple views such as front, back, right and left views or detailed views of a part of the drawing. To be able to keep a view in preferred way, customer defaults are inherited to every view on the drawing.

The figure below shows how a designer is able to do this manually from setting window.

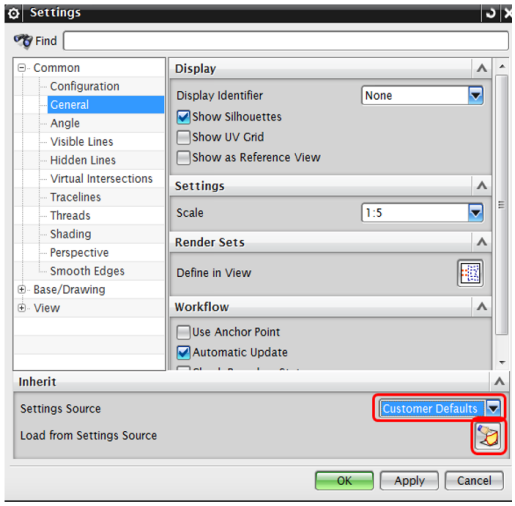


Figure 16. View from NX, setting window

The figure below shows a displayed part tree sample with its all sheets and views and information messages after the program is run.

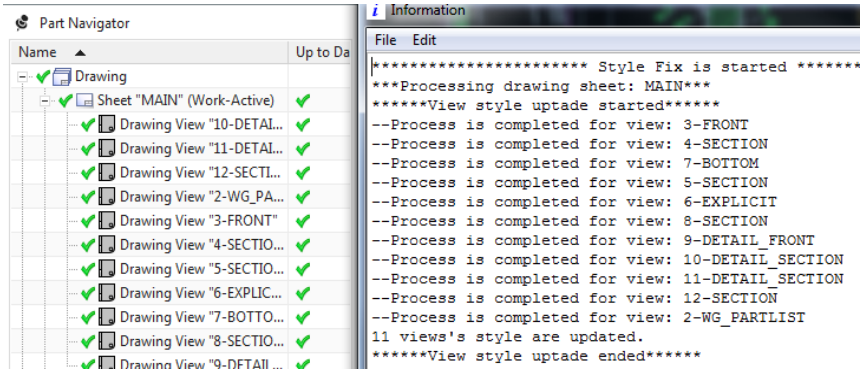


Figure 17. Information Window while fixing the views

3.3.1.2 Dimension Style

Drawing Fix tool will find all dimension on the drawing and it will save values that are listed below and it inherits customer default values. After inheritance it sets back the original values for listed (below) settings.

And also it sets font color and font to values given in the configuration file for dimension text, tolerance text and appended text.

- Tolerance type and Value: Drawing Fix Tool stores the tolerance type's values and after inheriting the customer defaults, the tool sets back the values to drawing.

  - Type
  - Decimal Place
  - Upper Tolerance
  - Lower Tolerance
  - Text Position
  - Zero Display
  - Show Leading Zeros
  - Convert Tolerance when Changing the Units

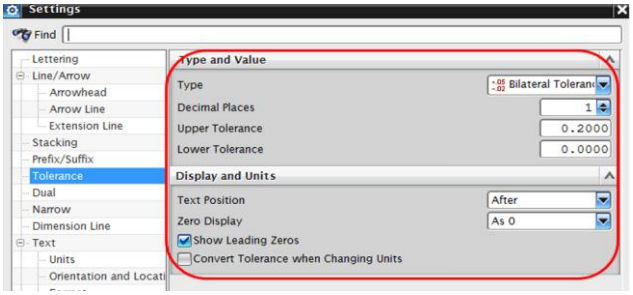


Figure 18. Shows Tolerance Types and Values

- Tolerance Text: The Drawing Tool stores the values listed below and set them back to the same values after inheriting the custom preferences.

  - Height
  - Font Gap Factor
  - Aspect Ratio
  - Line Gap Factor
  - Text GAP Factor

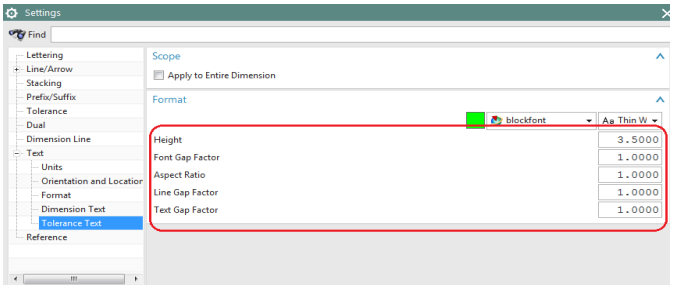


Figure 19. Tolerance Text Values

Note: If the tolerance type is “No Tolerance”, the tolerance text changes will not be done as it works with NX.

- Dimension Text: Drawing Fix Tool stores the Dimension Text values and sets them back after inheriting the customer defaults.

  - Height
  - Font Gap Factor
  - Aspect Ratio
  - Line Gap Factor
  - Dimension Line GAP Factor

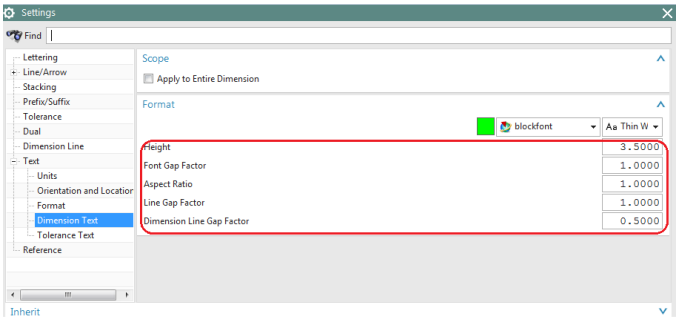


Figure 20. Dimesion Text Values

- Appended Text: Drawing Fix Tool stores the Appended Text Values and set them back after inheriting the customer defaults.

  - Height
  - Font Gap Factor
  - Aspect Ratio
  - Line Gap Factor
  - Text Line GAP Factor

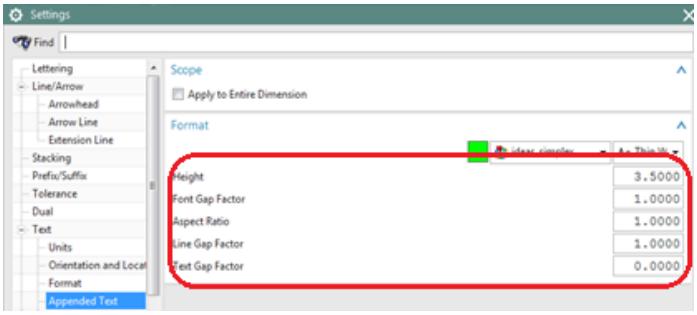


Figure 21. Appended Text Values

- Line/Arrow, Arrowhead: The tool stores the ArrowHead values and set them back after inheriting the customer defaults.

- Dimension Sides /Orientation
- Format /Length
- Format / Angle

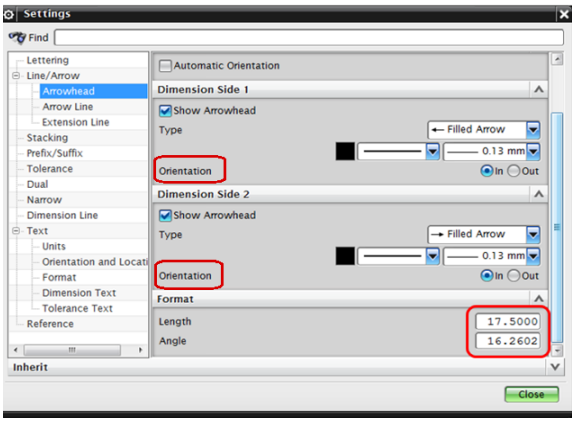


Figure 22. Appended Text Values

And finally the tool inform users for number of dimension which are fixed, and failures.

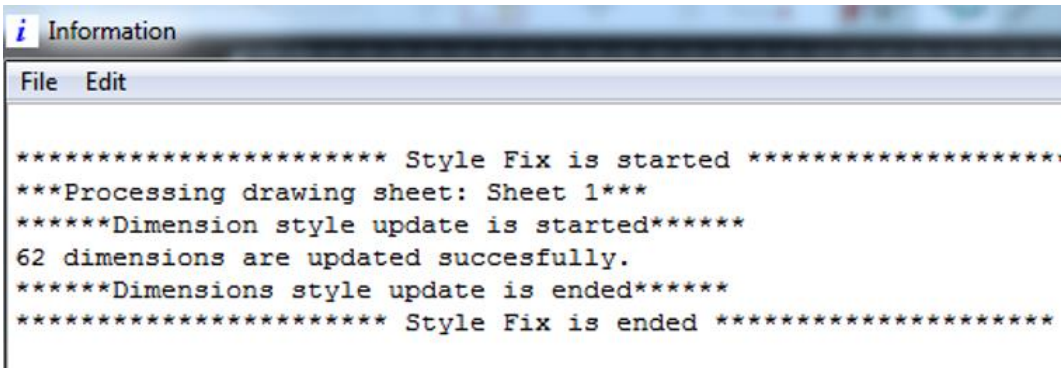


Figure 23. Caption from Information Window, shows dimensions updates

3.3.1.3 Notes Style

Drawing Fix tool will find all notes on the drawing and it will save values which are listed below and it inherits customer default values. After inheritance it sets back the original values for listed (below) settings.

And also it sets font color and font to values given in configuration.

- Height
- Font Gap factor
- Aspect Ratio
- Line Gap Factor
- Lettering Angle

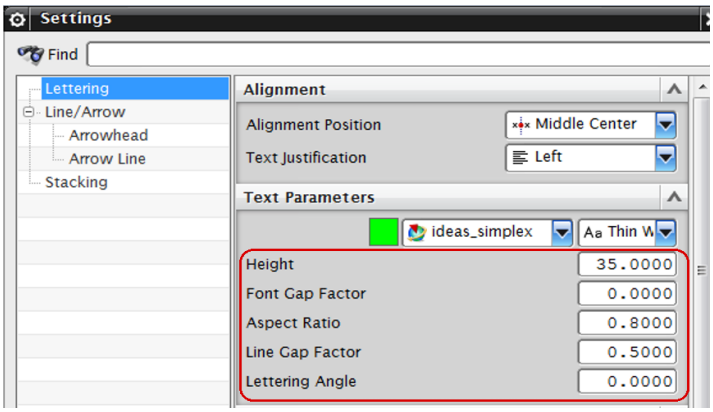


Figure 24. Lettering Values

And finally the program informs users about the number of notes which are fixed, and about the failures.

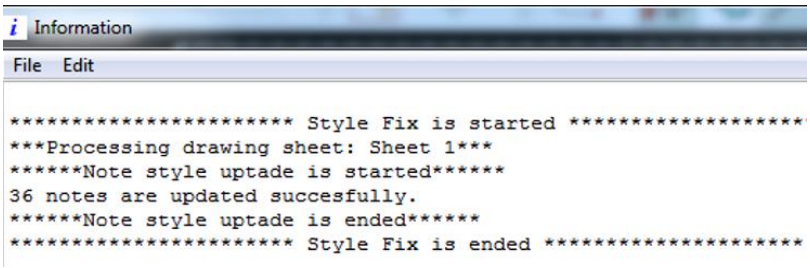


Figure 25. Information Window while fixing the notes

3.3.2 Fix Symbols

Fix symbols are designed to fix specific errors on Surface Finish Symbols, Weld Symbols and Identification Symbols. For this fix also user only needs to select and activate the process.

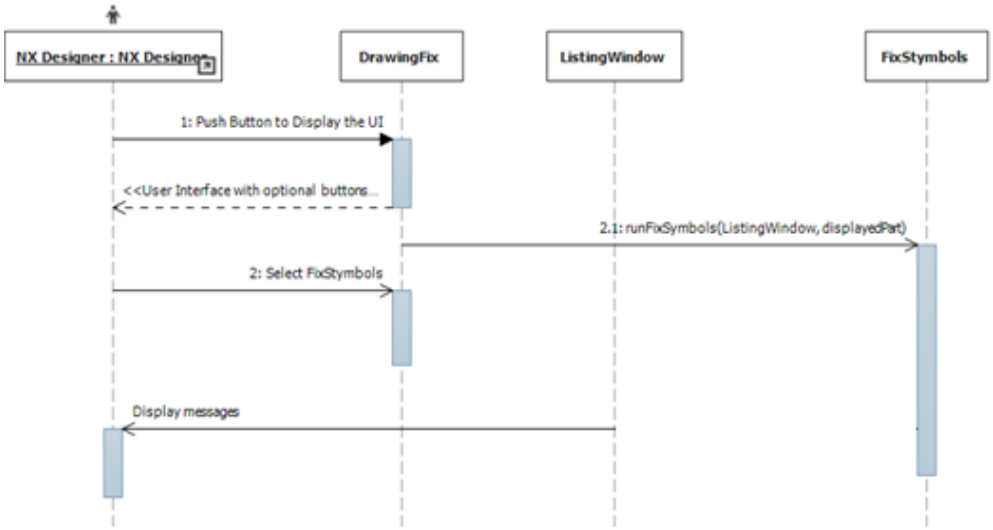


Figure 26. Implementation Design of Fix Symbols functionality

3.3.2.1 Surface Finish Symbols

This functionality is added to remove extra zeros created during the integration process. Drawing Fix tool finds all Surface Finish symbols on drawing and remove the zeros.

Attributes to be checked for zeros are listed below:

- Upper Text (a1)
- Lower Text (a2)
- Production Process (b)
- Waviness (c)
- Lay Symbol (d)
- Machining (e)
- Cutoff (f1)
- Secondary Roughness (f2)

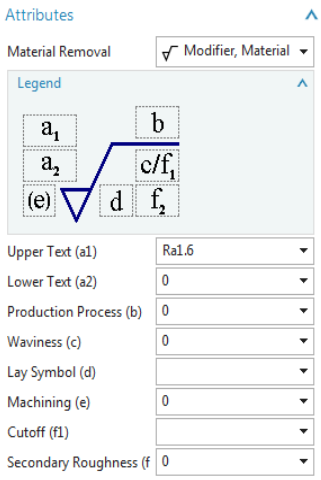


Figure 27. Values for Surface Finish Symbols

And if values are zeros, the tool removes them as it is shown below.

Before: After:

Figure 28. The Surface Finish Symbol before and after running Drawing Fix Tool.

Finally the tool informs the users about the process from information window.

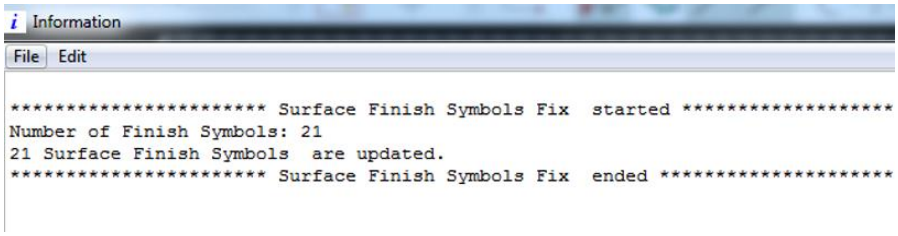


Figure 29. Information Window while fixing the Surface Finish Symbols

3.3.2.2 Weld Symbols

This functionality was added to change the symbol line thickness to thin. Drawing Fix tool finds all weld symbols on drawing and changes the line thickness value.

Attributes to be checked for zeros are listed below:

- Line Thickness

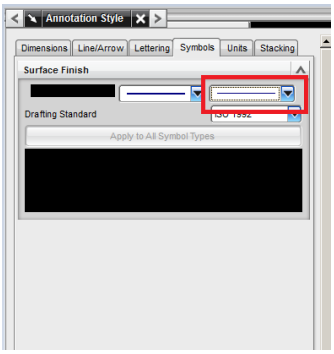
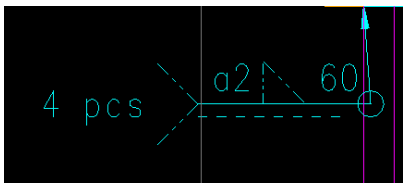


Figure 30. Values for Weld Symbols

The figure shows visual changes on Weld Symbols after running the Drawing Fix Tool.

Before:



After:

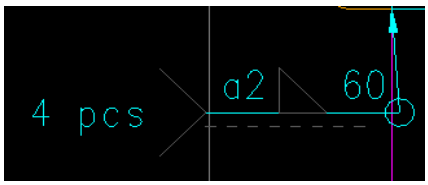


Figure 31. Weld symbols before and after running the Fix Style.

Finally the tool informs the users about the process from information window.

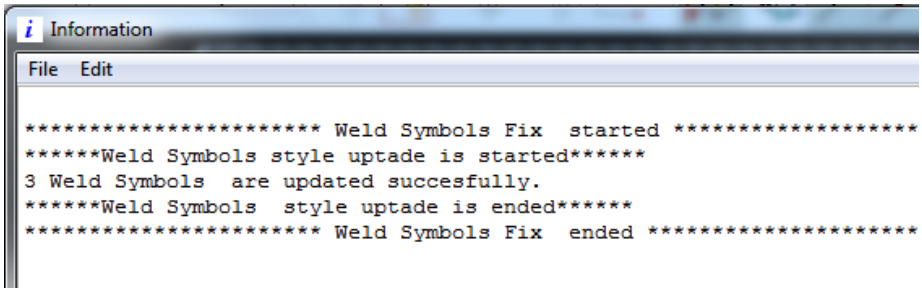


Figure 32. Information Window shows Weld Symbols Fixes.

3.3.2.3 Identification Symbols

This functionality was added to change symbol style properties. The tool finds all identification symbols on the drawing and change the style properties.

Attributes to be checked for zeros are listed below:

- Type to Without Stub
- Arrow Head to Dot

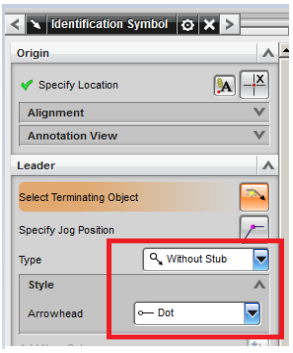
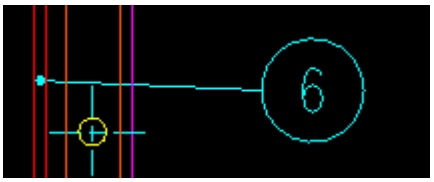


Figure 33. Identification Symbol setting Values

The figure below shows an Identification Symbol before and after running Drawing Fix Tool.

Before:



After:

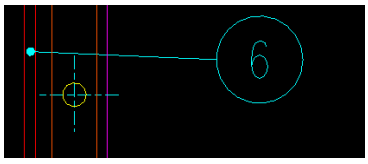


Figure 34. An Identification Symbols before and after running Drawing Fix Tool.

Finally the tool informs the users about the process from information window.

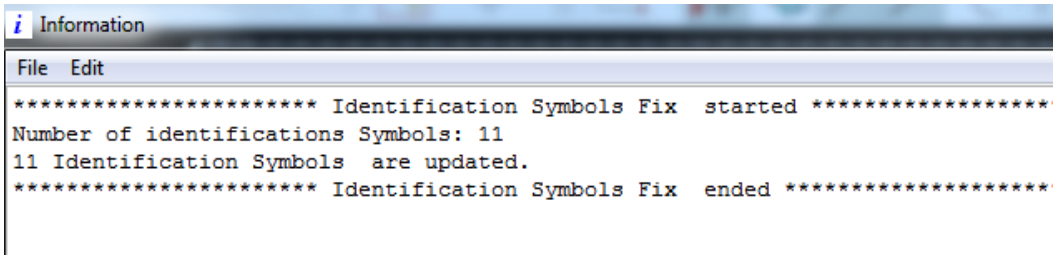


Figure 35. Information Window showing Identification Symbols fixes.

3.3.3 Remove I-deas TitleBlock and Frames

Drawings on NX has mostly have TitleBlock, tables and other custom symbols in order to display some data. And each company/customer have their own symbols. Since in the very beginning Drawing Fix Tool was created for migrated drawings, all these symbols need to be replaced with the customer's symbols in NX. Drawing Fix tool is responsible only for removing the symbols since there are already other ways to re-create them. But before removing them some data must be saved from the symbols for re-created symbols therefore Drawing Fix Tool gathers this data from the drawing and save it as properties.

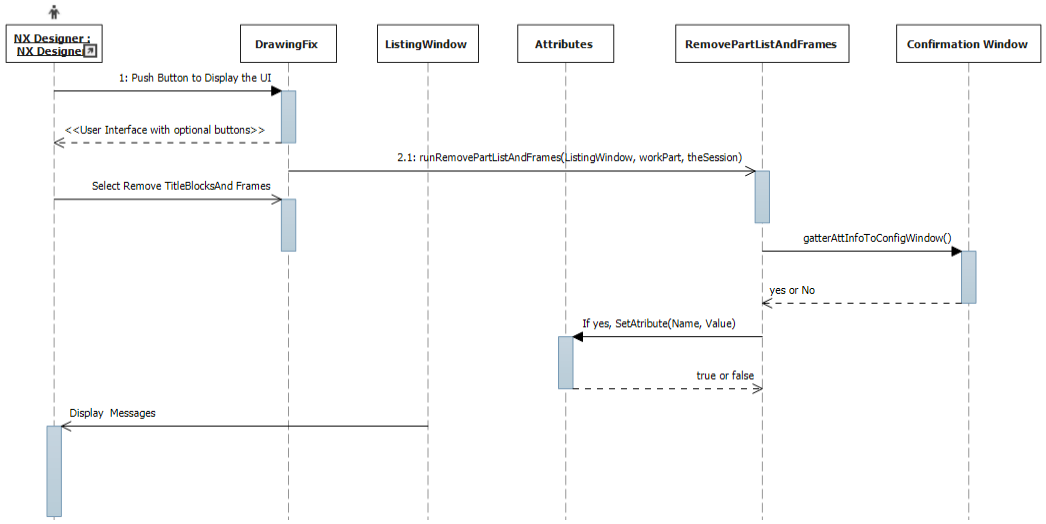


Figure 36. Implementation Design of Remove Idea-s Title Block and Frames functionality

3.3.3.1 Gathering Data from TitleBlock

Gathering data was one of the problems which could not be in the NX Open capability, since the titleblock is created as custom symbols and data is added as notes while defining the custom symbol. Normally there is no way to filter these data created as notes in custom symbols definition with NX Open capability. Therefore, to get these data, custom symbols are smashed and from smashed symbols, location of notes are found and from these locations values are gathered.

Drawing Fix Tool is implemented to gather data of Material name, Designer information and designing date therefor the tool tries to reach these data from titleblock. The steps below show the logic behind of capturing these data.

1. The configuration file contains name of the titleblock, name of the custom symbol which hold data and the notes which are used as property names.

```
<!--Name of the custom symbol used to create titleblock-->
<titleblockTemplate>TITLE_TEMPLATE</titleblockTemplate>
<!--Name of the custom symbol used to fill the titleblock-->
<titleblockData>TITLE_DATA</titleblockData>
<!--Notes defines designer information and material id -->
<propertyToRead>
       <made>Made</made>
       <materialID>MaterialID</materialID>
</propertyToRead>
```

Code Snippet 1. Snippet from the Configuration file, gathering data from the titleblock

2. Drawing Fix Tool finds the custom symbol which is used as title block template name given in configuration file in order read location of the note that reads “Made”. As can be seen in the figure below “Made” is part of custom symbol TITLE_TEMPLATE which is given in the configuration file.

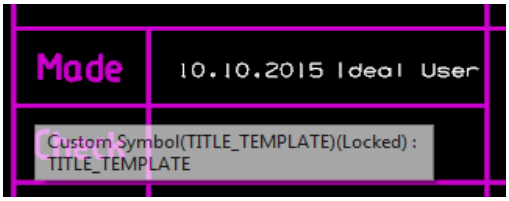


Figure 37. Snippet from a titleblock shows “Made” row.

In order to get location of the note “Made”, the tool smashes the custom symbol and find the location of “Made” and save location information.

3. Drawing Fix Tool finds custom symbols which keeps the data. As can be seen below, sample data is part of the custom symbol TITLE_DATA given in configuration file.

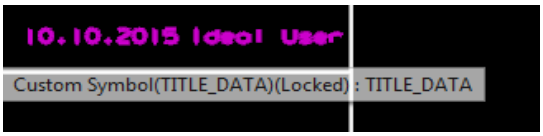


Figure 38. Snippet from a titleblock shows design info.

In order to read this data as notes, the tool smashes the custom symbol.

4. Since now all data are notes on the drawing, the tool check locations found in second step for the notes. It read all notes around this location range. It work in the same way for “MaterialId”. When it completes, a window pops up in order to get confirmation from the designer. The designer is able to modify these data from the window.

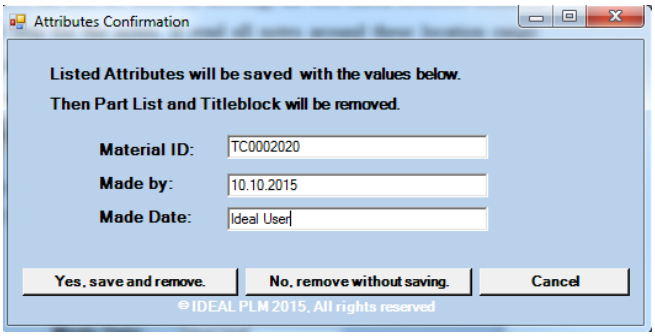


Figure 39. Attributes Confirmation Window.

5. Finally if the designer chooses “Yes, save and remove”. The data is saved as properties MADE_BY, MADE_DATE and MATERIAL_ID as it is shown below.

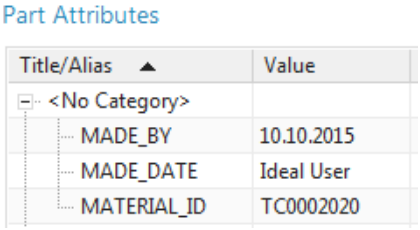


Figure 40. Shows properties of part.

3.3.3.2 Removing the Title Block and Frames

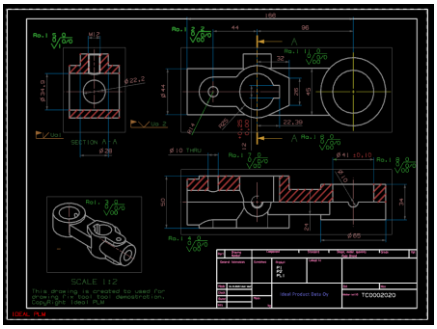
After gathering data, the Drawing Fix tool removes the titleblock and frames defined in configuration file.

```
<!-- List of custom symbols(titleblock, tables) that will be removed.-->
<listOfCustomSymbolToRemove>TITLE_TEMPLATE, BOM_TEXT,
BOM_ROW,TITLE_DATA</listOfCustomSymbolToRemove>
<!-- List of custom frames that will be removed.-->
<ListOfFramesToRemove>A0, A1, A2, A3, A4</ListOfFramesToRemove>
```

Code Snippet 2. Capture from configuration file, symbols to remove.

The figure below shows changes on the drawing after removing the titleblock and the frames.

Before:



After:

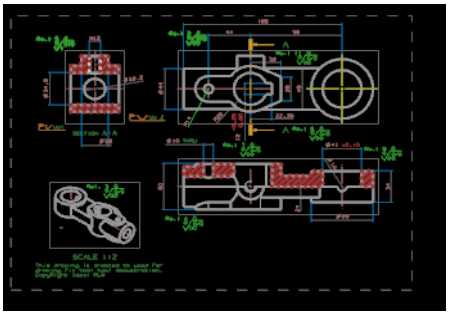


Figure 41. Before and after removing titleblock and frames

All information related to gathering data and the process of removing the title-block and the frames will be displayed in the information below as can be seen in the sample below.

```
********************** Process for removing PartList and Title Block is started *********************
A1 frame is removed.
Title Block and PartList are removed.
MATERIAL_ID is set to 'TC0002020'.
MADE_BY is set to '10.10.2015'.
MADE_DATE is set to 'Ideal User'.
********************** Process for removing PartList and Title Block is ended. **********************
```

Figure 42. Information Window display data related the Remove I-deas Title-Block and Frames.

3.3.4 Hide Points

On a migrated 2D drawing, during the migration process some custom symbols created in different system come with extra points. And also it appears to be that some drawings have points created by mistake or they have been forgotten to be removed. Since points are very small elements of drawing it is very big possibility that the designer may miss to clean or in case that they are part of the custom symbols to remove them requires smashing the custom symbols first.

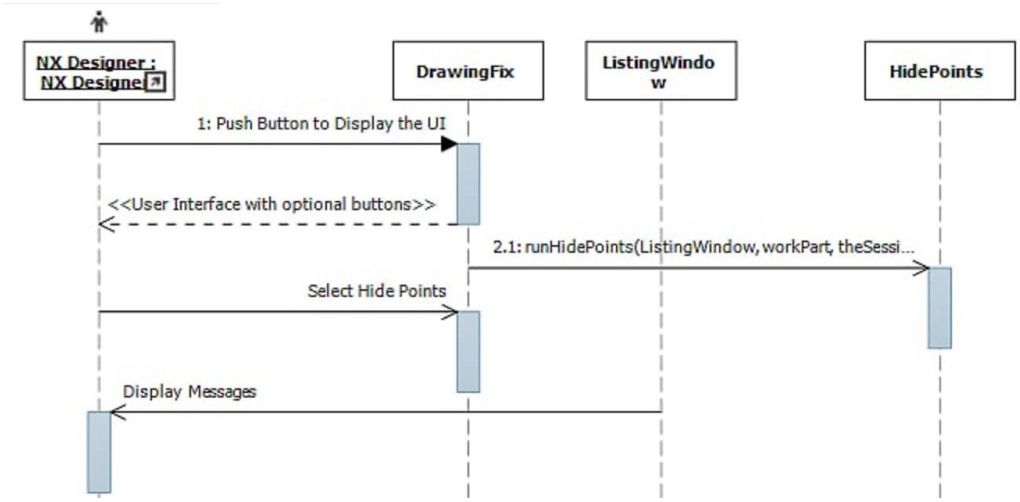


Figure 43. Implementation Design of Hide Points functionality

In order to get all points, The Drawing Fix Tool smashed all custom symbols given in the configuration file.

```
<listOfCustomSymbolsToHidePoints>
      REVISIO_ALAS, REVISIO_OIKEA,REVISIO_VASEN,REVISIO_YLOS
</listOfCustomSymbolsToHidePoints>
```

Code Snippet 3. Capture from configuration file, list of custom symbols to hide points.

After smashing, it finds all points in the drawing and hides these all by using NX own “Hide” command.

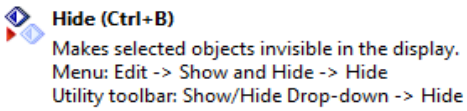


Figure 44. Capture from NX, displays Hide command.

4 GUI DESIGN

As mentioned before, one of the requirements to keep is to keep the user interface as simple as possible and to minimize the user intersection to the process. The Drawing tool has 3 different GUI, the first is the button added on the NX interface, the second is the main interface activated by the button on NX, and the third one is the confirmation window which is created to get user confirmation while removing the titleblock and saving gathered data from it.

4.1 Change on NX UI

The Drawing Fix tool is created for Drafting Application of NX, therefore modification is done on the drafting application toolbar. In order to create the button the TBR file is created to customize the toolbar.

The figure below shows the content of the DrawingFix.tbr file. It includes name of the button, label of the button, icon image directory and dll file directory.

```
TITLE     Drawing Fix
VERSION    170

BUTTON    Drawing Fix
LABEL     Drawing Fix
MESSAGE    Starts Drawing Fix Tool
BITMAP     $SITE_BASE_DIR\DrawingFix\startup\Drawing_Fix.bmp
ACTION    $SITE_BASE_DIR\DrawingFix\application\Drawing_Fix.dll
```

Code Snippet 4. Customizing the toolbar, tbr file

In order to activate this button on the interface a custom folder (DrawingFix) is defined in NX environment file and the custom folder is created. The structure of the custom folder is set in different ways for NX9 and NX8 since NX8 does not have all the features NX9 has.

- In order to set the visibility of the Drawing Fix button only on the drafting application mode for NX9, the TBR file is located under the application/profiles/ UG_APP_DRAFTING folder. And ribbon file is created and added to same location to be able to see the same button on ribbon mode.

- NX8 does not have the future to limit the visibility of buttons according to NX applications by using custom pre-defined folders. This problem is handled by coding. Before starting to the main UI, after pushing the Drawing Fix button on the NX interface, the tool checks the current mode of NX and if it is the drafting application it starts, if it is not it just informs the designers.

The figure below shows a sample of the NX interface with the Drawing Fix Button.

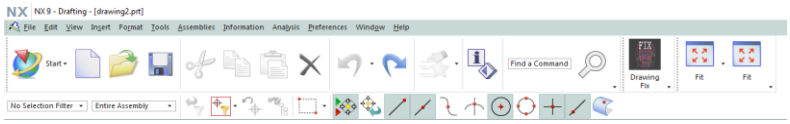


Figure 45. Capture from NX, NX tool bar.

4.2 Drawing Fix Main GUI

Drawing Fix main interface is created by NX Block UI Styler and it is designed as simple as possible. It has only buttons to activate the functionalities and applying cancelling and confirming the functionalities.

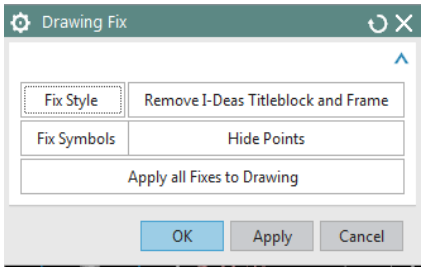


Figure 46. Capture from NX, Drawing Fix main GUI.

Buttons and their functionalities are listed below.

- Fix Style is created to start Fix Style functionalities (Check section 3.3.1).

- Fix Symbols is created to start Fix Symbol **(**Check section 3.3.2**)** functionalities.
- Remove I-deas Titleblock and Frame **(**Check section 3.3.3**)** is created to remove given custom symbols such as frames and titleblock and to save data gathered from titleblock.
- Hide Points **(**Check section 3.3.4**)** is created to start Hide Point functionalities.
- Apply all Fixes to Drawing is created to call all functionalities.
- OK is created to confirm the fixes and close the UI.
- Apply is created to confirm the fixes but it does not close the UI.
- Cancel is created to cancel all changed made by the tool.

4.3 Confirmation Window

Confirmation Windows is created for the functionality. Remove I-deas TitleBlock and Frames. As it is explained in section 3.3.3, the tool saves the data from the titleblock before removing it. Since this data is gathered based on location there might not be any notes or multiple notes or unrelated notes on the location. Therefore before saving the data the designer must confirm if they are the right values. This windows gives the possibility of changing or modifying the data gathered from titleblock in case it is needed.

The figure below shows the confirmation windows which is created as windows form on Microsoft Visual studio. As can be seen the designers have multiple options: cancelling the process, removing titleblock without saving the data and saving and removing the titleblock.

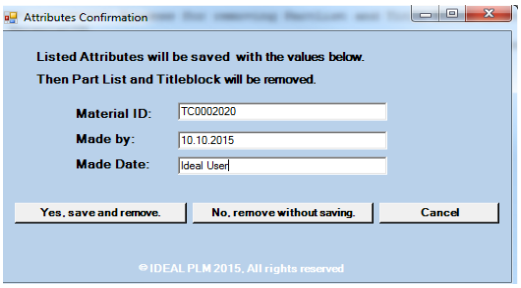


Figure 47. Confirmation Window

5 IMPLEMENTATION AND UNIT, FUNCTIONAL, SYSTEM INTEGRATION TESTING

Instead of treating testing as a quality gate at the end of the implementation process as it is in traditional projects, in this project the testing goes on continuously with implementation as it is in agile methodology. Therefore during the implementation phase unit testing, functional testing and system testing are executed by the developer and NX designers, other level tests are started with the customer.

5.1 Class Implementing Logic

In order to keep the software easy to extend and easy to understand for other developers, relations and structure of classes are kept in the same way. As it will be described in coming chapters, in order to allow the designer start to functionalities relevant methods must be called in update callbacks. And since the software has 4 main functionalities and relevant classes in each class only public method "run_*" will be created. In this way, when a new functionality is added, or after any modification, there will be no need to make changes in multiple classes, only relevant method will be added or modified. And in this "run_*" methods all sub methods will be called to execute the functionality.

```
if(block == buttonFixStyle) //  Fix Style button
   fixStyle.runFixStyle(lw, workPart, theSession);
else if (block == buttonRemoveIdeas) //  Remove Ideas button {
   removePartListAndTitle BlockObj.
       runRemovePartListAndTitleBlock(lw, workPart, theSession);
   checkForConfirmationForm = true;
}
else if (block == buttonFixSymbols) // FixSymbols button
   fixSymbols.runFixSymbols(lw, workPart, theSession);
else if (block == buttonHidePoints) // Hide Points button {
    hidePoints.
       runHidePoints(lw, workPart, theSession, configurations)
```

Code Snippet 5. Callback function for NX Block UI Design

In consideration of working on a part in a session, the NX session and working part variables are created only once in Drawing Fix class and pass through all the classes if it is needed. Also only one Listing Windows is created to avoid opening multiple information windows.

5.2 Iterations

In order to make the progress through successive refinement and implement the software in pieces, the building process is broken into to 10 iterations. In this section each iteration will be explained in detail.

5.2.1 Iteration 0: Reading configuration file

This iteration is named as zero because reading configuration file is implemented in the very beginning simply and it is improved based on need in every steps. As starting point, methods are implemented to check if the configuration file is added and each element of configuration file is able to be read. For instance below method show implementation of reading custom symbol list to Remove I-deas TitleBlock and Frames functionality.

```
/*!
* It is created to read removal custom symbols
* \param lw(I) Listing windows is used to print the information * \return
List<String>    List of custom symbols to be removed.
*/
public List<String> getRemovalList(ListingWindow lw)
{
      String           errorMessage = "";
      List<String>     list = new List<String>();
      try{
          // reads value of element from xml
          String stringValue =
             doc.Root.Element("listOfCustomSymbolToRemove").Value;
          String[] array = stringValue.Split(delimiters);
          char[] delimiters = new[] { ',' }; // splits with comma
          list.AddRange(array); // add to the list
     }catch (Exception){
          ErrorMessage += "List of the custom symbols, frames and
             title blocks cannot be read from configuration file.
             Please check the file. \n";
       }
     if (!errorMessage.Equals(""))
           lw.WriteLine(errorMessage + "\n"); // prints the message
return list;
}
```

Code Snippet 6. Method to read configuration file

5.2.2 Iteration 1: Creating Main Window and Integrating to NX UI.

As mentioned earlier, for the main user interface NX Block UI Styler is used. In order to create, integrate and test the UI, the process is divided into three steps.

5.2.2.1 Creating UI with NX Block UI Styler

By using NX Block UI Styler, an NX dialog box is created with buttons: Fix Style, Remove I-deas title block and Frame, Fix Symbols, Hide Points, Apply all Fixes is created and OK, Apply, Cancel buttons are activated.

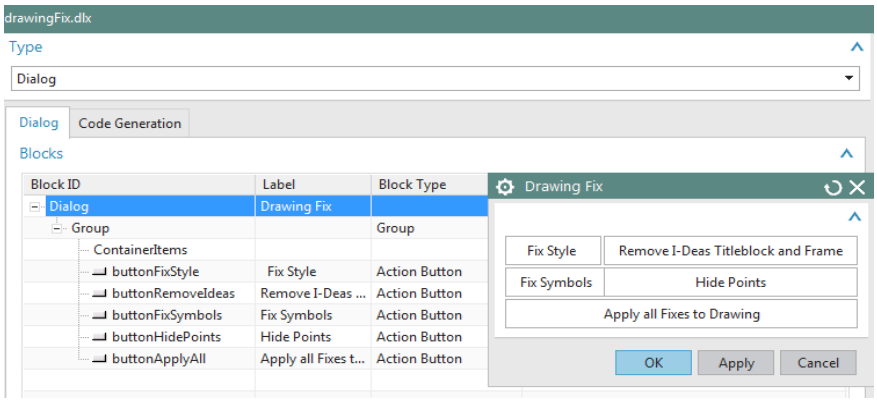


Figure 48. Creation of Main UI

Entry points are set as Callback function and Update is set as true in order to be able to add functionality to each created button. As it can be seen from the figure below programming language is chosen C#.

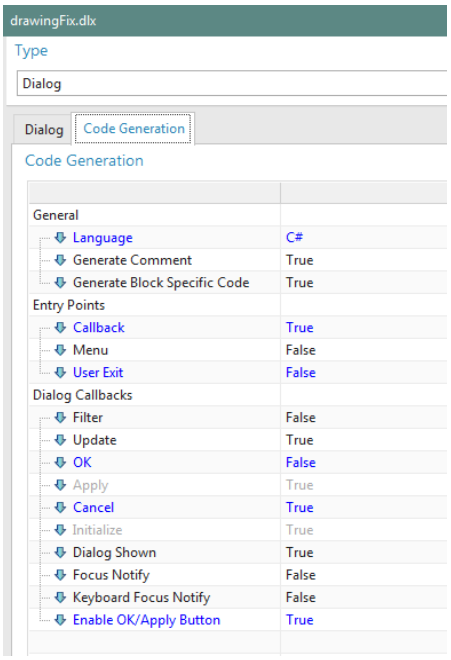


Figure 49. Code generation properties.

5.2.2.2 Modifying generated UI codes for Drawing Fix Tool.

NX Block UI styler creates two file: .dlx file and .cs file based on chosen programming language. In .cs file NX method named Show_DrawingFix () to display the dialog. This method is called in the Main method of the project.

```
//  Explicit Activation. This entry point is used to activate the //  application explicitly
public static int Main(string[] args)
{
   int retValue = 0;
   try{
       theProgram = new Program();
       // Call methods to lunch DrawingFix dialog.
       DrawingFix.Show_DrawingFix();
       TheProgram. Dispose();
   }catch (NXOpen.NXException ex)
   {
      // message box pops up if any error occurs.
      theUI.NXMessageBox.Show("Block Styler", NXMessage
           Box.DialogType.Error, ex.ToString ());
       retValue = 1;
   }
   return retValue;
}
```

Code Snippet 7. Main function, activates the application

Since entry points are added as call backs and update is set to true (check figure 40), NX created update call back method. In this phase message boxes are added under to each button to test if the UI works properly after integrating to NX.

5.2.2.3 Compiling and Integrating to NX to test

After adding the dll path on the button created on the NX toolbar (chapter 4.1). We tested the following matters in different environments.

- The integration process is succesfull, the dialog box response quickly.
- If NX allows the designer zoom in/out of drawing.
  Since NX Block UI Styler is in use, the designer is able to zoom in and zoom out while user interface is visible.
- If all buttons works properly.
  Each button is tested separately and they return the message as it is planned.

Figure 50. UI testing, button response.

- The program is unloaded when the designer closes the dialog box.
  After running test in different environment, it appears to be that library is not unloaded when the designer closes to the dialog box.

5.2.2.4 Returning to implementation, modifying the codes re-run the testing.

It is found that in the method getUnloadOption() generated by NX, the unload option is set to unload library at termination.

```
//Describes when an automation library should be unloaded.
public static int GetUnloadOption(string dummy)
{
    return (int)Session.LibraryUnloadOption.AtTermination;
}
```

Code Snippet 8. GetUnloadOption function, at termination.

In order to unload the Drawing Fix tool right after the designer closes the dialog box, the unload option is changed to “Immediately”.

```
//Describes when an automation library should be unloaded.
public static int GetUnloadOption(string dummy)
{
      return (int)Session.LibraryUnloadOption.Immediately;
}
```

Code Snippet 9. GetUnloadOption function, immediately.

After these changes, the testing steps are run again and the software is unloaded immediately after the program is completed.

5.2.3 Iteration 2: Adding Fix Style Functionality: View Style Fixing

View Style fixing is added as sub-functionality to style fixing therefore it is added to runFixStyle method.

```
/*!
* It is created to call all sub fixes.
* \param  lw       (I) Listing windows is used to print the information
* \param workPart  (I) Working part
* \param theSession(I) Current Session
*/
public void runFixStyle(ListingWindow lw, Part workPart, Session the-
Session)
{
    if (!lw.IsOpen)
          lw.Open();
    lw.WriteLine("***Style Fix is started ***");
     foreach (Drawing Sheet sheet in workPart.
          DrawingSheets.ToArray())
        fixViewStyle(lw, sheet, workPart); // start view Fix
   lw.WriteLine("***Style Fix is ended *** ");
}
```

Code Snippet 10. RunFixStyle method, fixViewStyle activation.

5.2.3.1 Implementing Fix View Style functionality

In order to fix all view style, first the program must find all views belonging the working part. For this purpose NX Open libraries provides a method named GetDraftingViews. After getting every view, for each of them executeViewStyleFix() method is used to apply the all view style fixes while all information is printed to the information window.

```
/*!
 * It created to fix views styles. It loads the customer defaults
setting to views.
 * It sets the view scale and view level values to false
 * \param lw        (I) Listing windows is used to print
 * \param sheet     (I) Drawing Sheet
 * \param workPart  (I) Working part
 */
private void fixViewStyle(ListingWindow lw, DrawingSheet sheet, Part
workPart)
{
    int i = 0;
    int ii = 0;
    String errorMessage = "";
    lw.WriteLine("Processing drawing sheet: " + sheet.Name)
    lw.WriteLine("******View style update is started******");
    DraftingView[] draftingViews = sheet.GetDraftingViews();
    if (draftingViews.Length > 0){
       // looping all views in a sheet
        foreach (DraftingView dw in draftingViews){
          try{
             String result = executeViewStyleFix(dw);
             if(result.Equals(""))
             {
                 lw.WriteFullline("--Process is completed for view:
                           " + dw.Name);
                i++; //counting successfully fixed views.
             }
             lw.WriteLine("--Process is completed for view: " +
                           dw.Name);
                 ii++; //counting failures
            }
          catch (Exception e){
             errorMessage += dw.Name + "'s view style could not be
                    updated. Error Message: "  +  e.Message + "\n";
             ii++; //counting failures
         }
       }
   }
    lw.WriteLine(i + ”views’s style are updated.");
    if (ii > 0) {
        lw.WriteLine(ii + " views's style could not be updated.");
        lw.WriteLine("     " + errorMessage);
    }
    lw.WriteLine("******View style update is ended******");
}
```

Code Snippet 11. fixViewStyle method

The executeViewStyleFix() method is the method that does actual work on the views. But In this study, the content of method will not be published.

5.2.3.2 Testing the fix view functionality

After adding the functionality tests are run for 4 different view types:

- Testing Base views
- Testing Projected view
- Testing Detail view
- Testing Section view

These all type of views could fixed successfully without any issue.

5.2.4 Iteration 3: Adding Fix Style Functionality: Dimension Style Fixing

Dimension Style fixing is one of the sub-functionality of Fix Style. Therefore this fix is added to runFixStyle method.

```
/*!
*
* It is created to call all sub fixes.
* \param  lw       (I) Listing windows is used to print information
* \param workPart  (I) Working part
* \param theSession(I) Current Session
*
 */

public void runFixStyle(ListingWindow lw, Part workPart, Session the-
Session)
{
   if (!lw.IsOpen)
      lw.Open();
   lw.WriteLine("***Style Fix is started ***");
   foreach (Drawing Sheet sheet in workPart.
     DrawingSheets.ToArray())
   fixViewStyle(lw, sheet, workPart); // start view Fix
   fixDimensionStyle(lw, workPart);// starts dimension style fix
   lw.WriteLine("***Style Fix is ended ***");
}
```

Code Snippet 12. RunFixStyle method, fixDimensionStyle activation.

5.2.4.1 Implementing Fix Dimension Style Functionality

Dimension Fix style functionality targets to fix all dimensions belong to current part, for this purpose it get all dimensions from the working part. And it call methods executeDimensionStyle for every single dimension, in this way it gather all process information for each dimension.

```
/*!
 *
 * It created to fix dimension styles.
 * It loads the customer defaults setting to dimensions and reload
 * some of setting which are descripted in the TDS documentation
 * of the tool
 * \param lw      (I) Listing windows is used to print the information
 * \param workPart(I) Working part
 *
 */
private void fixDimensionStyle(ListingWindow lw, Part workPart)
{
    int ii    = 0; // number of successfully fixed dimension
    int i     = 0; // number of failures
     String  errorMessage = ""; // error messages
     Dimension[]     allDimensions; //array to get all dimensions
    lw.WriteLine("***Dimension style update is started***");
     try
     {
          // getting all dimensions belong the part.
         AllDimensions = workPart.Dimensions.ToArray();
         //looping all dimensions
         foreach (Dimension dimension in allDimensions)
         {
            String result = executeDimensionStyleFix(dimension);
             if(result.Equals("")) //if success, no error message
               ii++; // counting successfully fixed dimension
             else
             {
                i++; // counting failures
                errorMessage +=result; // gathering error message
             }
          }
       }
       catch (Exception e)
       {
            // in case if dimension cannot be gotten from part.
            lw.WriteLine("Error: " + e.Message);
            return;
       }

       lw.WriteLine (ii + ”dimensions are updated successfully.");
      if (i > 0)
        lw.WriteLine(i + " dimension update are failed.");

      lw.WriteLine("***Dimensions style update is ended***");
}
```

Code Snippet 13. fixDimensionStyle merhod.

Above method shows how dimensions are gathered from part and how information and error messages are collected. All fixes are done on a dimension by calling method executeDimensionStyle, the content of this method is not published in this study.

5.2.4.2 Testing the fixes on dimension

Since the dimension fix is the secondly added fix after view fix test are run in two phases.

- Phase 1: View Fix Style is commented in the source codes, in this way only dimension fixes are tested based on the below listed cases.
    - Based on dimension type listed below:
        - Inferred Dimension
        - Horizontal/Vertical
        - Parallel
        - Perpendicular
        - Angular
        - Diameter/Radius
        - Perimeter
    - Based on the existence of properties which are set back after inheriting customer defaults: Tolerance Text, Dimension Text, Appended Text, dimension orientation and location, arrow line properties, tolerance values, decimal places, stubLength.
- Phase 2: In second phase fix view functionality is activated back and the tool is tested in case of any collapse between these two functionalities. Each listed cases in phase 1 are checked.

5.2.4.3 Returning to implementation, modifying the codes re-run the testing.

After running the test, it is noted by the tester, if a dimension does not has appended text, the tool does not complete the fix on the dimension and return as a failure. Therefore a small check is added to implementation in order to continue fixing even if a dimension has no appended text.

```
//gathering original appended text values
if (!appendedText.Equals("") || !appendedText.Equals(null))
{
     //implementation is not published in this level
}
```

Code Snippet 14. Appended Text check on dimensions

5.2.5 Iteration 4: Adding Fix Style Functionality: Notes Fix Style

Note Fix Style is added as a part of Style Fix functionality and it is added to run-FixStyle method as a sub-functionality.

```
/*!
* It is created to call all sub fixes.
*
* \param  lw        (I) Listing windows is used to print the information
* \param workPart   (I) Working part
* \param theSession (I) Current Session
 */
public void runFixStyle(ListingWindow lw, Part workPart, Session the-
Session)
{
  if(!lw.IsOpen)
    lw.Open();
  lw.WriteLine("***Style Fix is started *** ");
  foreach (Drawing Sheet sheet in workPart.DrawingSheets.
             ToArray())
     fixViewStyle(lw, sheet, workPart); // start view Fix
  fixDimensionStyle(lw, workPart);// starts dimension style fix
  fixNoteStyle(lw, workPart); // start note style fix.
  lw.WriteLine("***Style Fix is ended *** ");
}
```

Code Snippet 15. RunFixStyle method, fixNoteStyle activation.

5.2.5.1 Implementing Fix Notes Style Functionality

It is mentioned in earlier chapter of this documentation that this functionality is created to gather all notes belongs to parts and fix them based on the specification. For this purpose the tool finds all notes of work part by using NX Open library to get notes as collection.

```
/*!
 * It created to fix notes styles.
 * It loads the customer defaults setting to notes.
 * \param lw      (I) Listing windows is used to print the information
 * \param workPark (I) Working part
 */
private void fixNoteStyle(ListingWindow lw, Part workPart)
```

```
{
   int  ii    = 0; // number of successfully fixed notes
   int  I   = 0; // number of failures
   String  errorMessage    = ""; // error messages for each note.
   lw.WriteLine("***Note style update is started***");
   try
   {
     //getting all notes belong the part
     NXOpen.Annotations.BaseNote[] allNotes
              = workPart.Notes.ToArray();
     if (allNotes.Length > 0) {
        foreach (NXOpen.Annotations.BaseNote note in allNotes)
        {
           String result = executeNoteStyleFix(note);
           if (result.Equals("")) //if success, no error message
             i++; // counting the notes, successfully fixed
           else{
              errorMessage += result;
              ii++; // counting the failures
           }
         }
     }
    }
  catch (Exception e)// catch the errors         {
      lw.WriteLine("Error: " + e.Message);
  return;
  }
  lw.WriteLine(i + " notes are updated successfully.");
  if (ii > 0)
    lw.WriteLine(ii + " notes could not be updated.");
    lw.WriteLine("***Note style update is ended***");
}
```

Code Snippet 16. fixNoteStyle method.

Above method shows how notes are collected from working part and how information and error messages are gathered. There is a method called executeNotesStyle which executes all fixes on each note on the drawing.

5.2.5.2 Testing Notes on drawing

Since there is one type of notes on drawings, testing process for notes included only check on matters below:

- Checking the customer defaults if they are inherited.
- Checking general text size, if the original value is stored.
- Checking general text aspect ratio, if the original value is stored.
- Checking line space factor, if the original value is stored.
- Checking char space factor, if the original value is stored.

- Checking angle, if the original value is stored.
- Checking arrow head length, if the original value is stored.
- Checking arrow head included angle, if the original value is stored.

Based on the matters above, tests are done in two phases.

- Phase 1 includes tests, only testing notes on drawings. In order to do that all other foxes are de-activated in the code level.
- Phase 2 included tests with other implementation to check if fix notes functionality collapse with fix dimension style functionality or with fix view style functionality.

5.2.6 Iteration 5: Fix Symbol: Surface Finish Symbol fixing

Surface Finish Symbol is added as a sub-functionality to Fix Style functions. Therefore fixSurfaceFinishSymbol method is called in the method runFixSymbols().

```
/*!
* It is created to run all sub fixes.
* \param lw         (I) Listing windows is used to print
* \param workPart   (I) Working part
* \param theSession (I) Current Session
*/
public void runFixSymbols(ListingWindow lw, Part workPart, Session the-
Session)
{
   lw.WriteLine("***Surface Finish Symbols Fix  started *** ");
   fixSurfaceFinishSymbols(lw, workPart, theSession);
   lw.WriteLine("***Surface Finish Symbols Fix  ended *** ");
}
```

Code Snippet 17. RunFixSymbol method, fixSurfaceFinishSymbol activation.

5.2.6.1 Implementing Surface Finish Symbols Fix

In order to execute all fixes on Surface Finish Symbols, property DraftingSurfaceFinishSymbol of annotation class (from NX Open libraries) is used. In this way all Finish Surface symbols belong the working part is collected into the array.

Then looping this array executeFinishSymbolFix method is called for each symbol to be able to fix the symbols.

```
/*!
* It is created to find all Surface Finish Symbols and remove the zeros.
*
*  \param lw        (I) Listing windows is used to print the information
*  \param workPart  (I) Working part
*  \param theSession(I) Session
*/
private void fixSurfaceFinishSymbols(ListingWindow lw, Part workPart,
Session theSession)
{
   int i = 0;
   int ii = 0;
   String errorMessage = "";
   NXOpen.Session.UndoMarkId markId2;
   markId2 = theSession.
      SetUndoMark(NXOpen.Session.MarkVisibility.Invisible,
      "SurfaceFinishSymbols fix");

   NXOpen.Annotations.DraftingSurfaceFinish[] surfaceFinnishSymbols
   = work  Part.Annotations.DraftingSurfaceFinishSymbols.ToArray();
   lw.WriteLine("Number of Finish Symbols: "
      + surfaceFinnishSymbols.Length);
  foreach (DraftingSurfaceFinish draftingSurfaceFinish in
      surfaceFinnishSymbols)
  {
    try
    {
       String result
        = executeFinishSurfaceSymbolFix(draftingSurfaceFinish);
       if (result.Equals("")) //if success, no error message
          i++; // counting the notes which are successfully fixed
       else
       {
             errorMessage += result;
             ii++; // counting the failures
       }
    }
    catch (Exception e)
    {
       errorMessage += draftingSurfaceFinish.Name.ToString()
       + "Surface Finish could not be updated. Error Message: "
       + e.Message + "\n";
       ii++;
    }
  }
    lw.WriteLine(i + " Surface Finish Symbols are updated.");
    if (ii > 0)
    {
      lw.WriteLine(ii
       + " Surface Finish Symbols could not be updated.");
      lw.WriteLine(errorMessage);
     }
     theSession.UpdateManager.DoUpdate(markId2);
}
```

Code Snippet 18. fixFinishSymbols method.

5.2.6.2 Testing Surface Finish Symbol Fix

Since there is only one type of Surface Finish Symbol tests are done on following matters listed below:

- Checking Upper Text, if it is zero it is removed.
- Checking Lower Text, if it is zero it is removed.
- Checking Production Process, if it is zero it is removed.
- Checking Waviness, if it is zero it is removed.
- Checking Lay Symbol, if it is zero it is removed.
- Checking Machining, if it is zero it is removed.
- Checking Cutoff, if it is zero it is removed.
- Checking Secondary Roughness, if it is zero it is removed.

After adding this functionality test are done in 2 phases:

- Phase 1 is executed to run only Fix Surface Finish Symbols.
- Phase 2 is executed to check if there is any collapse between this functionality and Fix Style functionalities.

5.2.7 Iteration 6: Fix Symbol: Weld Symbols Fix

Weld symbols fixes is added as sub-functionality to symbols fixes and fixWeldSymbol method is created and added to runFixSymbol method.

```
/*!
* It is created to run all sub fixes.
* \param lw         (I) Listing windows is used to print the information
* \param workPart   (I) Working part
* \param theSession (I) Current Session
*/
public void runFixSymbols(ListingWindow lw, Part workPart, Session the-
Session)
{
   if(!lw.IsOpen)
     lw.Open();
   lw.WriteLine("***Surface Finish Symbols Fix  started *** ");
   fixSurfaceFinishSymbols(lw, workPart, theSession);
   lw.WriteLine("***Surface Finish Symbols Fix  ended *** ");
   lw.WriteLine("***Weld Symbols Fix  started **** ");
   fixWeldSymbols(lw, workPart);
          lw.WriteLine("***Weld Symbols Fix  ended *** ");
}
```

Code Snippet 19. RunFixSymbol method, fixWeldSymbol activation.

5.2.7.1 Implementing Weld Symbols Fix

In order to fix all weld symbols which belong to the working part, the property Weld of the Annotation class is used. With this property all weld symbols are collected and added to array.

```
/*!
* It created to fix weld symbols styles. It loads the customer defaults
* setting to notes reload some of setting which are descripted in the TDS
* documentation of the tool.
*
*\param lw       (I) Listing windows is used to print the information
*\param workPart (I) Working part
*/

private void fixWeldSymbols(ListingWindow lw, Part workPart)
{
   int i = 0;
   int ii = 0;
   lw.WriteLine("***Weld Symbols style update is started***");
   NXOpen.Annotations.Weld[] welds
   = workPart.Annotations.Welds.ToArray();
    if (welds.Length > 0)
    {
      foreach (NXOpen.Annotations.Weld weld in welds)
      {
        try
        {
           String result = executeWeldSymbolFix(weld);
           if (result.Equals("")) //if success, no error message
              i++; // counting the notes which are successfully fixed
            else
            {
               errorMessage += result;
               ii++; // counting the failures
             }
        }
        catch (Exception)
          ii++
    }
  }
    lw.WriteLine(i + " Weld Symbols  are updated successfully.");
    if (ii > 0)
       lw.WriteLine(ii + " Weld Symbols could not be updated.");
    lw.WriteLine("******Weld Symbols  style update is ended******");
}
```

Code Snippet 20. fixWeldSymbol method.

Above method shows how weld symbols are gathered, information and error messages are collected. Actually fixes are done in executeWeldSymbolFix which is not published in this study.

5.2.7.2 Testing Weld Symbol Fix

Weld symbols Fix functionality is tested for only line thickness of the symbol in this version of Drawing Fix Tool. This test is executed in 3 phases:

- Phase 1 is run to see only weld symbol functionality for this purpose other fix symbol functionalities are commented in implementation.
- Phase 2 is run to see weld symbol fix and other symbol fixes together in order to see if there is any collapse between them.
- Phase 3 is run to see Fix symbol and Fix style fixes result together at this level. Since symbols, dimensions and notes are annotation types, this was a mandatory test case to see if any change affects each other.

5.2.8 Iteration 7: Fix Symbol: Identification Symbols Fix

Identification symbol fix is also defined as one of sub-functionality of fix Symbols therefore it is added to FixSymbol class as a method and called in runFixSymbolm method in order to activate it.

```
/*!
* It is created to run all sub fixes.
* \param lw          (I) Listing windows is used to print the information
* \param workPart    (I) Working part
* \param theSession (I) Current Session
*/
public void runFixSymbols(ListingWindow lw, Part workPart, Session the-
Session)
{
  lw.WriteLine("***Surface Finish Symbols Fix  started ***");
  fixSurfaceFinishSymbols(lw, workPart, theSession);
  lw.WriteLine("***Surface Finish Symbols Fix  ended *** ");
  lw.WriteLine("***Weld Symbols Fix  started ***");
  fixWeldSymbols(lw, workPart);
  lw.WriteLine("***Weld Symbols Fix  ended ***");
  lw.WriteLine("***Identification Symbols Fix  started ***");
  fixIdentificationSymbols(lw, workPart, theSession);
  lw.WriteLine("***Identification Symbols Fix  ended *** ");

}
```

Code Snippet 21. RunFixSymbol method, fixIdebtificationSymbol activation.

5.2.8.1 Implementing Identification Symbols Fix

Identification symbols are one of the Annotations in NX, therefore in order to get all identification symbols belonging to the working part a property name IdSymbols from Annotation class provided in NX Open libraries in used. After gathering all identification symbols to a collection, every single symbol is fixed and relevant messages are collected. The actual work is done in method named executeIdentificationSymbol which is not published in this documentation.

```
/*!
* It is created to find all identification symbols and it edits to
* identification symbols without stub as type and dot as arrowhead.
* \param lw         (I) Listing windows is used to print the information
*  \param workPart  (I) Working part
*  \param theSession(I) Session
*/
private void fixIdentificationSymbols(ListingWindow lw, Part workPart,
Session theSession)
{
    int      i    = 0;
    int      ii   = 0;
    String   errorMessage = "";

    NXOpen.Session.UndoMarkId markId3;
    markId3 =
      theSession.SetUndoMark(NXOpen.Session.MarkVisibility.Invisible,
      "IdentificationSymbols fix");
    NXOpen.Annotations.IdSymbol[] identificationSymbols =
      work   Part.Annotations.IdSymbols.ToArray();
    lw.WriteLine("Number of identifications Symbols: "
      + identificationSymbols.Length);
    foreach (NXOpen.Annotations.IdSymbol IdSymbol in identificationSym-
bols)
    {
       try
       {
           String result = executeIdentificationSymbol(IdSymbol);
             if (result.Equals("")) //if success, no error message
               i++; // counting the notes which are successfully fixed
             else
             {
                errorMessage += result;
                ii++; // counting the failures
             }
           }
            catch (Exception e)
           {
              errorMessage += IdSymbol.Name.ToString()
              + " Id Symbol could not be updated. Error Message: "
              + e.Message + "\n";
              ii++;
           }
         }
         lw.WriteLine(i + " Identification Symbols are updated.");
         if (ii > 0)
         {
            lw.WriteLine(ii +
             " Identification Symbols could not be updated.");
```

```
            lw.WriteLine(errorMessage);
          }
        theSession.UpdateManager.DoUpdate(markId3);
}
```

Code Snippet 22. fixIdentificationSymbol method.

5.2.8.2 Testing Identification Symbol Fix

Identification symbols Fix functionality is tested for Terminator type and leader head arrow head setting in this version of Drawing Fix Tool. This test is executed in 3 phases:

- Phase 1 is run to see only identification symbol functionality for this purpose other fix symbol functionalities are commented in implementation.
- Phase 2 is run to see identification symbol fix and other symbol fixes together in order to see if there is any collapse between them.
- Phase 3 is run to see Fix symbol and Fix style fixes result together at this level. Since symbols, dimensions and notes are annotation types it was a must case as it was for weld symbols fix functionality.

5.2.9 Iteration 8: Gathering Data from Title Block

Gathering data from title block is defined as sub-functionality of RemovePartListAndTitleBlock functionalities. And it is invoked in run RemovePartListAndTitleBlock.

```
/*!
* It is created to run all sub fixes.
* \param lw         (I) Listing windows is used to print the information
* \param workPart   (I) Working part
*/
public void runRemovePartListAndTitleBlock(ListingWindow lw, Part work-
Part, Session theSession)
{
  lw.WriteLine("****Process for removing Title Block is started **** );
  gatherAttValuesToCofirmantionWidows();
  lw.WriteLine("***Process for removing Title Block is ended. ***");
}
```

Code Snippet 23. runRemovePartListAndTitleBlock method.

5.2.9.1 Implementing Gathering data from Title Block

As mentioned earlier in this documentation the Drawing Fix Tool reads custom symbols and attributes names from configuration file. After reading these data it collects all custom symbols and filter them by their names. The below code capture shows how this part is implemented.

```
BaseCustomSymbol[] customSymbols
= workPart.Annotations.CustomSymbols.ToArray();
//sub method calls to get location of relevant attributes title getLoca-
tionsOfTitles(customSymbols);
// check if there is TitleBlock
if (!titleBlockIsFound)
{
  lw.WriteLine("No TitleBlock is found.");
  return;
}
```

Code Snippet 24. Checking TitleBlock.

In order to get the location of Titles, a method getLocationOfTitles are created and this methods is implemented to get location of given attribute titles belongs to given title block name in the configuration file.

After getting the location, the tool use these locations to read value of the attributes. And it checks if there is only one value. Because it appears to be that in some cases the designer might add some extra notes or labels in the attribute block. In these cases it simply inform the user to leave only correct values. Below it is shown how material id attribute is checks works as a sample.

```
// If TitleBlock is found it gathers the value of the rows.
getAttributesfromTitleBlock(customSymbols);
// If there are more than 1 value in the Material id row, it does not get
//any and inform user from information window.
if (materialIDList.Count > 1)
   lw.WriteLine("There are too many value for Material id therefore title
block will not be removed." + "Please check the title block.");
// If there is no value on material is row.
else if (materialIDList.Count == 0)
{
   lw.WriteLine("No Material id information is found.");
   lw.WriteLine("Please check the TitleBlock.");
}
```

Code Snippet 25. Checking the attributes.

Actual values are collected in a method named getAttributesFromTitleBlock, after values are gathered it pops-up the confirmation window for the designers (See figure 29).

5.2.9.2 Testing Gathering Data from title block

On the assumption that there are three possible error cases, five different test cases are suggested:

- Test Case 1: One entry into the rows.

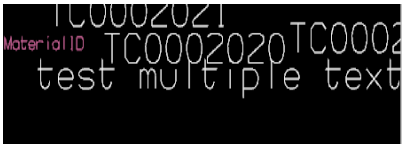
Capture from TitleBlock

Capture From Confirmation Window

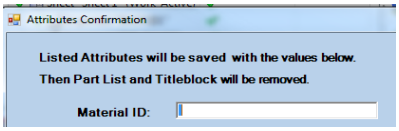
Figure 51. Captures from NX to show result Gathering Data from title block for test case 1, MaterialID row is used as a sample

As can be seen from the above figure, if there is only one entry in the row, the tool gathers the data to the confirmation window.

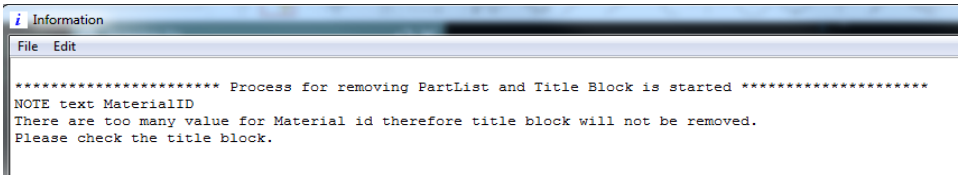
- Test Case 2: Multiple entries into the rows.



Capture from Title Block



Capture From Confirmation Window



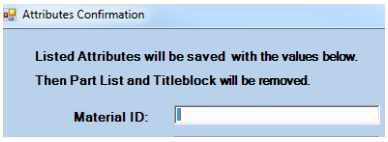
Capture from Information Window

Figure 52. Captures from NX to show result Gathering Data from title block for test case 2, MaterialID row is used as a sample

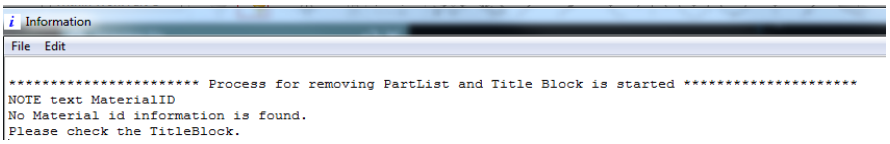
In the case of multiple entries to the title block the tool returns an empty row to the confirmation window in case a designer would entry manually and continue. But in a situation that the designer does not enter anything and chooses to save the properties, the tool ends the process and informs the designer from information window with the text above.

- Test Case 3: No entry in the rows

Capture from TitleBlock



Capture From Confirmation Window



Capture from Information Window

Figure 53. Captures from NX to show result Gathering Data from title block for test case 3, MaterialID row is used as a sample

As can be seen above, if the tool cannot find any information, it provides an empty text filed for the designer to enter data manually in case the designer still would save the attributes. But at the same time send the message to information window related to case.

- Test Case 4 is proceeded with the configuration file, if there is no given name in the file, and if given names for the custom symbols and properties do not exist on drawing.
- Test Case 5 is proceeded to check if the attributes are saved properly. The method provided by NX accepts argument as String values, since these values are read from the text fields as string values there was no need to test for different types such as integer or Boolean. But the tool is tested for

different length of data and if there is already existing attributes on property list. The both cases gave accepted results.

In these test cases, there was need for modification which were done during testing process.

5.2.10 Iteration 9: Removing TitleBlock and Frames

The Drawing Fix tool provides different "remove" options to remove the titleblock and the frames. These options are added to the confirmation window.

```
// if yes, Remove and save button is pushed
private void buttonYes_Click(object sender, EventArgs e)
{
   // call methods to remove the symbols
   DrawingFix.removePartListAndTitleBlockObj.removeCustomSymbols();
DrawingFix.
   removePartListAndTitleBlockObj.setAttributes(textBoxMaterialID.Text,
   textBoxMadeBy.Text, textBoxMadeDate.Text); // save attributes
   this.Close(); // close the confirmation window
}
// If No, Remove without saving the attributes button is pushed.
private void buttonNO_Click(object sender, EventArgs e)
{
  // call methods to remove the symbols
  DrawingFix.removePartListAndTitleBlockObj.removeCustomSymbols
  this.Close();// close the confirmation window
}
```

Code Snippet 26. Implementation of confirmation window options.

5.2.10.1 Implementing Removing Title Block and Frames

After the gathering data process is ended, the tool checks if the custom symbols to be removed from the configuration files are on the drawing. If they are, it adds them to delete list and execute the method to remove the custom symbols. It also removes the notes belonging to these custom symbols.

```
/*!
* It is created to remove custom symbols, Title Block, rows, and Notes
* written as MaterialID and made by/date.
*/
public void removeCustomSymbols()
{
   // loop all custom symbols in removal
  foreach (BaseCustomSymbol symbol in customSymbolToRemove)
     // adds them to Delete List
     theSession.UpdateManager.AddToDeleteList(symbol);
```

```
  // loop all notes in removal list
  foreach (NXOpen.Annotations.BaseNote note in notesToRemove)
     // add them to delete list
     theSession.UpdateManager.AddToDeleteList(note);
  // tell session to do the update
  theSession.UpdateManager.DoUpdate(markIdDelete);
  // Print information
  lw.WriteLine(messageForRemoving);
  // Print information
  lw.WriteLine("Title Block and PartList are removed.");
}
```

Code Snippet 27. removeCustomSymbols method.

5.2.10.2 Testing Removing Title Block and Frames.

Testing this functionality is executed based on two assumption; existence of the custom symbols given in configuration files and non-existence of the custom symbols. Both cases are tested separately and the tool is provided the expected results as it is explained in section 3.2.3.2.

5.2.11 Iteration 10: Hiding Points

The hide point functionality is added as one of main function to the tool, and it is activated by the button on the main UI.

5.2.11.1 Implementing Hide Point functionality

As a first step of implementation the runHidePoints method is implemented to activate the functionality.

```
/*!
* It is created to run all sub fixes.
* \param lw         (I) Listing windows is used to print the information
* \param workPart   (I) Working part
* \param theSession (I) Current Session
* \param configuration (I) Data from configuration file
 */
public void runHidePoints(ListingWindow lw, Part workPart, Session the-
Session, Config configuration)
{
   lw.WriteLine("***Hide Points Fix is started ***");
   List<String> customSymbolListFromConfig
     = configuration.getCustomSymbolsListForHidePoints(lw);
   // if it cannot get the data from config file
   if(customSymbolListFromConfig.Count > 0)
       hidePoints(lw, workPart, theSession, customSymbolListFromConfig);


   lw.WriteLine("***Hide Points Fix is ended *** ");
```

```
}
```

Code Snippet 28. runHidePoints method

In runHidePoints, the tool first gathers a list of custom symbols to be smashed in order to get all points and the list is given as a parameter to hidePoints method. After smashing the symbols it gets all points and repeat activate “Hide” command of NX.

```
/*!
* It is created to hides points. It smashed given custom symbols in
* config file and removes its points too.
* \param lw         (I) Listing windows is used to print information
* \param workPart  (I) Working part
* \param theSession(I) Session
* \param customSymbolListFromConfig(I) list from configuration file
*/
private void hidePoints(ListingWindow lw, Part workPart, Session the-
Session, List<String> customSymbolListFromConfig)
{
   int   numberHidden   = 0;
   String  errorMessage = "";
   // smashed the custom symbols to separate the points
   getAndSmashCustomSymbols(workPart, customSymbolListFromConfig, lw);
   try
   {
     String type = NXOpen.DisplayManager.ShowHideType.Points.ToString();
     numberHidden = theSession.DisplayManager.HideByType(type,
     NXOpen.DisplayManager.ShowHideScope.AnyInAssembly);
     lw.WriteLine(numberHidden + " point(s) is/are hidden.");
   }
   catch (Exception e)
     errorMessage += "Error occurred while hiding the points: "
     + e.Message+"\n";
   if (!errorMessage.Equals(""))
     lw.WriteLine(errorMessage);
}
```

Code Snippet 29. HidePoints method

5.2.11.2 Testing Hide Points

Test execution of hide point functionality did not have any assumption, it is simply done based on result of hiding the points from the drawing. Each time when the test is executed the result simple checked if there is any visible point left.

6 TESTING

Before finalizing the implementation and deploying the Drawing Fix tool for all NX designers of customers, three levels of testing are executed.

6.1 Acceptance Testing

This level of testing is executed by the NX designers from customer's side and IDEAL PLM in order to validate if the result meets the needs of customers. For this purpose the tool is tested by the project team members in custom virtual machines.

6.2 Alpha Testing

After acceptance testing the tool is deployed the development environment for NX designers to execute the test, this test is done totally by the customers and accepted to be deployed production environment for beta testing.

6.3 Beta Testing

At this level the tool is deployed to the production environment for a specific number of employees to ascertain the tool has no discernible bugs before being published for all. By time the numbers of user for testing increased and currently the tool is in use with the agreement which provide the customers fixes in case any bugs existence.

7 CONCLUSION

The main objectives of this work was to develop a software for fixing problems on drawings and adapting them to the NX system. As a result the Drawing Fix Tool was developed. The software meets all the requirements set by the customers and it can be used in order to fix the problems and help migrating drawings successfully. The Drawing Fix Tool is currently used by two big companies in Finland.

In the beginning the project required so much effort in order to determine if the solution is possible to implement with the NX Open library capacity. The project also required NX knowledge which was lacking in the beginning of the project but this problem was solved by the spirit of team work built in IDEAL PLM. Every necessary answer and support were provided by the other employees for the customer satisfaction.

7.1 Future Work

The Drawing Fix tool can be extended by adding new functions based on customer needs. New functions have already been added to the tool by IDEAL PLM specialists and tested by the customers and the specialists. More custom functions have also been requested by current users of the tool.

REFERENCES

/1/ Siemens Product Information
http://www.plm.automation.siemens.com/en_us/products/nx/ideas/

/2/ IDEAL PLM
http://www.ideal.fi/en/info/company-info/

/3/ Agile Development Methodology
http://www.strategybeach.com/our-agile-development-methodology/

/4/ Siemens NX
http://www.plm.automation.siemens.com/en_us/products/nx/about-nx-software.shtml

/5/ Siemens NX Block UI Styler
https://docs.plm.automation.siemens.com/tdoc/nx/10/nx_api/#uid:index_blockstyler:intro_int_ov_v1

/6/ NX Open API

https://docs.plm.automation.siemens.com/tdoc/nx/10/nx_api/#uid:index_nxopen_prog_guide:id1142146:what_is_nxopen

/7/ Teamcenter

http://www.plm.automation.siemens.com/en_us/plm/

/8/ Teamcenter

https://docs.plm.automation.siemens.com/tdoc/tc/10.1.4/help/#uid:index_plm00002:xid1014176:c01a0001

/9/ Teamcenter design View

https://docs.plm.automation.siemens.com/tdoc/tc/10.1.4/help/#uid:index_plm00002:xid1014176:xid1014480:xid1014494

/10/ Simple workflow sample

https://docs.plm.automation.siemens.com/tdoc/tc/10.1.4/help/#uid:index_plm00002:xid1014176:xid1014489:xid1015409

/11/ .NET Framework

http://msdn.microsoft.com/enus/library/vstudio/zw4w595w%28v=vs.100%29.aspx

/12/ NX System Information

https://docs.plm.automation.siemens.com/tdoc/nx/10/release_notes/#uid:index_xid849803:id1264341

/13/ Oracle Virtual Box

https://www.virtualbox.org/