Final thesis

Tools and Versioning for GUI text in SDP3

By

Rakesh Dronamraj

LIU-IDA/LITH-EX-A--10/036--SE

2010-09-30
Summary

Scania, one of the heavy engines manufacturers, produces Scania Diagnos Programmer 3 (SDP3) to facilitate repair process in their workshops. SDP3 is localizable software which challenges to separate User Interface strings (UI strings) during development process and later combine with the localized strings for local user access. The objective of this report is to provide knowledgeable solution for Graphical User Interface (GUI) development, especially with respect to synchronization of UI strings in SDP3.

The migration of SDP3 from .NET 3.0v framework to .NET 3.5v framework satisfies modern standards and needs. With regards to migration of SDP3’s localization process, I have attempted to summarize major .NET 3.5v framework methods that can be used for localization of GUI text in SDP3. Experiments show that tools used to facilitate the localization process also lack important features. Although pre-build process and post-build process provide promising solutions for localization, using them along with some proprietary localization tool should result in more features, better and faster production cycle. However, proprietary localization tool have to be used with anyone of the localization methods.

Foreword

I whole heartedly thank Gunnar Robertsson (Head of YSII Department, Scania) for providing me an opportunity to display my ability of analysis and comprehension on vast platform of resources. I thank Ove Bergmark (System Architect, YSII Department, Scania) for keeping me, well informed and always focused on the aim of this thesis. Thank you for guiding me in spite of your busy schedule.

I thank Jalal Maleki (Department of Computer & Information Science, Linköping University) and Henrik Eriksson (Dept. of Computer and Information Science, Linköping University) for giving my first break in Sweden. I thank my parents and Saritha, for supporting and believing in me. Above all, thank you, almighty, for giving me an opportunity.
# Table of Contents

Summary .................................................................................................................................. 3

Foreword .................................................................................................................................. 3

1. Introduction ....................................................................................................................... 6
   1.1 Problem Statement ..................................................................................................... 6
   1.2 Goal............................................................................................................................. 7
   1.3 Personal motivation ..................................................................................................... 7
   1.4 Structure of the report ................................................................................................. 7

2. Background ....................................................................................................................... 9
   2.1 Important Terms and their Best Practices ................................................................... 9
   2.2 Process at Scania ..................................................................................................... 13
   2.3 Method ...................................................................................................................... 19

3. Alternative processes in Localization .............................................................................. 21
   3.1 Overview of existing processes ................................................................................ 21

4. Experiments .................................................................................................................... 24
   4.1 Purpose of Testing .................................................................................................... 24
   4.2 Prerequisites for Testing ........................................................................................... 24
   4.3 Plan ........................................................................................................................... 24
   4.4 Tests ......................................................................................................................... 25
   4.5 Differences ................................................................................................................ 26
   4.6 Result ........................................................................................................................ 27
   4.7 Discussion ................................................................................................................. 28

5. Conclusion and Future Work ........................................................................................... 30

Appendix A ............................................................................................................................. 31
   1: Localization without Using LocBaml.exe ................................................................. 31
   2: Localization with LocBaml tool ................................................................................ 33
   3: Localization with a XAML ResourceDictionary and LocBaml ............................. 34
1. Introduction

Internet has given modern business wings, with which it can reach target audience beyond boundaries. Although market today assumes that everyone knows English, majority of the users would prefer the product to support their regional language. Internationalization of software application means one can work with the application in their local language. In other words internationalizing an application allows support for native input, processing, storage, display and printing. Internationalization is the best way to communicate with a variety of global audience and tell them “we understand where you come from!” Though this is the strategy employed by most of large businesses around the world, some still fail to understand its importance. [2c] Decision for internationalization is often taken by developers at early stage of software development life cycle. This eases the process of localization because no developer would like to write same code for multiple locales. Localization here refers to the process of adding new languages to the software.

The change in modern development paradigm has blessed developers with .Net technology. Building and deploying applications has just got better. The integration of many open standards like XML and SOAP has provided .Net framework with the right base and made it seamless developing environment. Although .Net localization process seems to be complex, many modern applications exist that provide right set of tools for localizing .Net applications. However, it all depends on developers to decide for the right localization process.

1.1 Problem Statement

Scania, one of the world’s leading manufacturers of heavy trucks, buses, industrial and marine engines also produces Scania Diagnos & Programmer 3 commonly known as SDP3. SDP3 is a computer-based diagnostic tool that simplifies and enhances repair process, maintenance work and service operations in garage or workshops. Service technician can sit in the cabin and implement troubleshooting using computer or like devices, which plugs into the instrument panel. SDP3 has been developed to minimize the disruption and reduce the downtime when the vehicle is taken to the workshop. [1]

This tool is developed to meet modern requirements of vehicle, industrial and marine systems especially with respect to their electrical systems. With the increase in complexity of each system there is an increase in the time consumed for troubleshooting it. This places greater demand on both tools and technicians. Thus, SDP3 plays a vital role in reducing downtime.

Global market trends have imposed demands on SDP3 to be localized for different locales of the world. This demand requires separation of localizable content from development content in Graphical User Interface (GUI) of the application. SDP3 has numerous functional parts and is built on various business components. Many professionals work simultaneously on each of these components in the application. Developers develop code for GUI and Technical Writers give UI strings to GUI. This process is however, unsynchronized and involves personal interaction. The reason for this is technical writer may
or may not understand the context of GUI because they do not have similar GUI view as
developers.

Some of the constraints posed by Scania on this thesis are: In present development process, language used for implementing application is en-US (Stands for English in United States) and UI strings written by Technical Writers use Swedish as standard. Swedish standard for Technical writers should not be altered, since a development platform is already chosen, there should be no change in either platform (.NET 3.5) or programming language (WPF and C#) and the ID’s assigned to UI strings are generated based on GUI Specifications designed at Scania. These specifications are imported from older version of SDP3, even before choice of platform and programming language were made. These specifications no longer fit into present development process, so, further replacement of these specifications is required. This replacement would not only serve the purpose but also give rise to a new process.

1.2 Goal

The primary goal of this report is to study existing development process for UI strings and analyze all possible alternatives so as to replace the same. Secondary goal is to study availability of different tools which facilitates alternative development process incorporating new flow structure for UI strings. In other words, a tool for synchronization between Developer and Technical writers for UI strings.

1.3 Personal motivation

I was completely new to localization process until I began my work at Scania. I have always been interested to learn and explore new opportunities and possibilities. After I started my research, the topic became more interesting and changed the way I looked at developing an application.

1.4 Structure of the report

Throughout the report I have maintained three-level hierarchal convention. The words mapped within glossary are italicized. The codes corresponding to Appendix B are made bold. This is the sample code that was used during experimenting.

Chapter 1: Introduction presents the most important terms used in this report, Internationalization and Localization. Problem statement and goal of this work is also documented in detail.

Chapter 2: Background consists of theoretical framework. In this chapter, all the important definitions and terms that are used in rest of the report like Internationalization, Localization, Translation, WPF, XAML, etc. are defined along with their best practices and recommended practices by Microsoft. Present GUI text development process along with disadvantages is also described in this section. Method consists of all the techniques that were employed while gathering relevant information associated with thesis work.

Chapter 3: Alternative processes in Localization, describes all the possible approaches available in .Net for localization. Theoretical description of all the possible approaches is documented in this section. Technical details have been documented in Appendices for future reference.
Chapter 4: Experiments section describes the tests that were conducted using a sample code. Four of five approaches mentioned in chapter 3 were tested during testing phase. Given sample code was modified so as to fit in approaches’ procedure. The tests are then documented as results in following sub-section where outcome of these tests are discussed in terms of merits and demerits. This section also has a result and relevant discussion about possible alternative approach.

Chapter 5: In this chapter I conclude my work along with future suggestions.
2. Background

2.1 Important Terms and their Best Practices
Internationalization is the process of separation of text from source code to allow translation of the text into other languages without the need for re-design or re-compilation. It is the ability to represent the character set of a particular language. [2c][4][9]

Internationalization ensures that the product is: functional, adhering to international market norms and localizable. International market norms refer to character sets, keyboard layouts, date format, time format and currency format. Information here is clear; free from local slangs, jargons and culture-specific references. This also helps in overall reduction of localization expenses. Externalizing localizable objects from the code provides smooth and efficient process. Internationalizing a product during development phase is upheld by Microsoft and Bert Esselink. [2d][4] The term ‘Enablement’ is often used in the context of internationalization. “Enablement is the process of adjusting software to make it functional in certain countries or geographical locations.” [4] For example a double byte enabled application can be used to display Asian characters.

The most important parameters for software internationalization are character encoding, location of translatable objects, GUI design and cultural standards. Character encoding challenges not only the characters that are being displayed but also input characters. Unicode is 16 bit character set that encompasses scripts and general purpose symbols for almost all languages today. Location of translatable objects poses specific problems. When software development was in adolescent stage, the application was developed several times to support different languages because translatable objects were embedded in source code. As technology evolved centralizing all translatable objects into a resource file became common practice. Externalizing these objects provides efficiency, security and quality during localization. Often translators and localization vendors are employed to complete translation process. If translatable objects are included in source code then, firstly, important and under production code is given to translators; security risk, and secondly, challenges translator’s ability to find these objects in source code; efficiency and quality risks. GUI design is the most important aspect as this is the actual user interface. Extra space should be provided in GUI to allow expansion of text. Bert Esselink [4] recommends approximately 30% extra space should be provided. Cultural Standards include date format, time format, address format, phone number format, currency, measurement parameter etc.

“Localization is the process of adapting and translating a software application into another language in order to make it linguistically and culturally appropriate for a particular local market. Modern software developers consider localization as a part of the development process of a software product.” [4]

Most of the software development is done in US and so, English has been adopted as the base language. But this would mostly change in future. Localization is no
longer just translation of English text into other languages and it is not just about translation. It is the process of translation while preserving linguistic correctness. For example, a sentence in English may mean something else when translated into other language. However, localization ensures that the underlying meaning of the sentence is preserved in all languages. Localization also addresses cultural and technical issues: cultural issues like color, graphics regionally accepted norms, etc, technical issues like handling bi-directional texts, spacing between characters for some eastern languages, etc. According to Madan Puraskar Pustakalaya [12] there are few factors to consider for localization of software. They are: nature and scope of software product, size of target market and audience, length of production cycle and anticipated update frequency, competitor behavior, market acceptance and national or international legislation. As they are self explanatory more emphasis is laid on key concepts of localization.

In order for any process to be accepted world-wide, setting standards would be first step. Important organizations for standardization such as Unicode Consortium, ISO, IEEE, etc, help lay the right foundation for localization process. Character set encoding, single byte encoding, multi byte encoding and different encoding systems in localization are discussed in detail within references [3][4][8][9][12]. Fonts, glyphs, bitmap, vector fonts and output methods are requirements for rendering text on standard output devices. One example for glyphs is the copy write symbol which is sequence of characters in itself. Bitmap is used to represent glyph shapes on pixels in two dimension and vector fonts describe glyphs with lines and curves. Locales were first introduced along with internationalization concept where generic frameworks included cultural specific information like conventions and character encodings. [12] “When a localized application executes, its appearance is determined by two culture values. (A culture is a set of user preference information related to the user's language, environment, and cultural conventions.)” [2d][9] For example sv-SE stands for Swedish in Sweden and de-AT stands for German in Austria.

During localization process we need to consider answers to following questions. Are source files available for each component that contains user interface text? Do the files contain country specific information that needs to be changed, such as default page sizes, currency symbols, numbers, etc? Should locale information be changed in the resource file?

Localization Best Practices within .NET framework [2d] [4] [9]

- Having localizable content such as strings, error messages, dialog boxes, menus, embedded objects, etc, in separate resource-only DLL’s facilitates localization.
- Hard-coding strings in UI resources and addition of non-localizable content into resource-only DLL’s are to be avoided. This simplifies translator’s job.
- Avoiding usage of composite strings at run time from concatenated phrases will reduce grammatical errors for any locale.
- Avoiding usage of text in images and icons can reduce cost of localization.
- Well spaced margins and sufficient padding will yield good translatable application.
Globalization combines both internationalization and localization. It is often used in the context of sales and marketing when software developers, go global, during development, translating and distributing the product for foreign markets. [4]

“Globalization is the first step in the process. A globalized application supports localized user interfaces and regional data for all users. Truly global applications should be culture-neutral and language-neutral. A globalized application can correctly accept, process, and display a worldwide assortment of scripts, data formats, and languages.” [2d]

Globalization Best Practices within .NET framework [2a][2d][4][9]

- It’s better for application to follow Unicode standards.
- There are many cultural aware classes available in System.Globalization namespace which are recommended to format data. For example; sorting, comparisons, date-time, numbers, calendar specific literals and more.
- System.Text namespace contains many encoding classes to enable applications to read and write data to and from a variety of encodings.
- Usage of error detection features is highly recommended. UTF8Encoding class offers security and error detection mechanisms.
- To prevent problems concerning parsing and combined characters, handling strings as whole is recommended over assuming them as individual characters. This eases sorting and searching for substrings.
- Testing of application on international operating system will prove handy.

Some of the best practices for Localization and Globalization in Windows Presentation Foundation (WPF) based UI as prescribed by Microsoft: [2a][3][9]

- To use full ability of built-in localization APIs creation of UI in XAML is advised.
- Relative or automatic sizing schemes are preferred over fixed or absolute positioning.
  - Use SizeToContent; and keep widths and heights set to Auto.
  - Avoid using Canvas to lay out UIs.
  - Use Grid for size-sharing feature.
- Usage of additional space in margins allows possible character overhanging.
- Enabling TextWrapping on TextBlock avoids clipping.
- Usage of xml:lang attribute provides advantages like spell checking, font fallback, change in hyphenation as per locale, number substitution, complex script shaping and more.
- Setting FlowDirection explicitly will help right-to-left and left-to-right representation of text.
- In order to provide extra context to localizers, usage of localization comments are recommended.
- There are many localization attributes so as to control localization of an application rather than selectively omitting Uid’s.
- Usage of MSBuild /t:updateuid and /t:checkuid are recommended to append Uid’s and check Uid’s as they track of changes between development and localization. Any
addition of $Uid$ properties after localization and usage of duplicate $Uid$ properties are not advisable.

- The use of UltimateResourceFallback in AssemblyInfo.* is highly recommended. This means that the applications falls back to the nearest resource match, in case of no resource key definition or if it is missing.

“Translation is the process of converting written or displayed text or spoken words to another language.” [4]

This is not same as word-to-word translation. Translation in the context of localization means that the underlying meaning of the article or text or strings is restored in locale specific language. Translation over time has evolved and became a key factor in industry competitiveness. The number of steps between content creation and translation has reduced drastically over time. The growth in technology has made separate teams working on content generation and translation from different locations come closer and work simultaneously. Many translation tools and .NET framework have this ability. Many multilingual databases, translation memories, glossaries, central repositories and technical dictionaries are setup to facilitate translation. These tools not only provide basic translation functions but also provide translators the context for generating linguistically correct translatable objects. This is rendered by WYSIWYG (What You See Is What You Get) editors embedded in these tools.

During this process answers to following questions should be considered: Which software resource files need to be translated? How should the resource files be translated? Are there any images or icons that need to be translated? Are there space restrictions that translators should keep in mind?

Testing plays a vital role in deciding product’s quality. Most vendors perform cosmetic GUI testing on localized applications. Cosmetic testing is an integral part of software testing process. The testing agreement between product owner and localization vendor will yield better and effective product. Often this agreement outlines the amount of testing done on localized application. To check the quality of localizable applications following questions are to be answered: Should the localized software also be tested? What type of testing should be done, i.e., only linguistic testing, or functionality, compatibility or regression testing? On which platforms should the software be tested? Are the test scripts available?

While testing a localizable application, the first step after translation is the linguistic test. Here the translator validates the translated software in the context, with or without the help of an engineer. The second test is the functionality test where the localization engineers check if the localization of the application did not damage the functionality. Usually, third and final test in localization of a software is Quality Acceptance (QA) or delivery test where the application is installed the way it will be by the end user to check final deliverables against original instructions.[4]
2.2 Process at Scania

"Scania Diagnos & Programmer 3 (SDP3) communicates with Scania vehicles and Scania industrial and marine engines. The program has been developed to support the electrical system with CAN communication. The program is used for troubleshooting, adjusting customer parameters, calibrations, conversions affecting the electrical system and during campaigns to update the control unit software." [1]

In the following Figure 2:1, description of the GUI text development process is broken down into component level. Each step is explained as follows:

1. User interface along with logic is developed. Implementation of product is done here.
2. These files are then parsed using LocBaml tool provided by Microsoft. Parsing generates .CSV file which contains temporary UI strings or text.
3. Same UI strings or text in different languages are sent via WINGS into Production tool.
4. Production tool stores all data into database files (*.mdb).
5. Localization Tool developed at Scania makes .CSV files for different languages.
6. LocBaml is used to generate resource files (*.resources.dll).
7. & 8. both produce a versioned application into sub-versioning system (ClearCase).

Files and Tools

Figure 2:1
In this section I will describe above mentioned steps closely and compare them with Best Practices discussed in earlier section. SDP3 Application has long history; during every cycle (or iteration) application is not built newly, from start. Today, there are many changes made to this existing application so as to suit the needs of modern technology. Business components are added to this application on regular basis. This often depends on release of new equipment, parts, machines and/or tools. Hence new text is written only for the newly added parts of the application and is subjected to localization.

In step one, default locale is specified and user interface of the application is implemented. User interface here contains temporary text. Each component is implemented by a developer, in agreement to requirement specifications. The temporary strings or text are mostly relevant to the component and developer has some knowledge as to what the text should represent. For example, it may be “OK” button or “CANCEL” button on a typical pop-up window. One such example is also included in the Appendix B as sample code. **Window1.xaml** contains all the implementation for GUI of one such component. The corresponding code to manipulate this GUI is implemented in **Window1.xaml.cs**, also included in the Appendix B. Once the component is implemented along with functionality, the project is built; this generates binary file for the overall component.

In step two, localization of application is done by parsing the localizable resources out of the main assembly. This is the first step in localization. Here UI elements and properties are extracted from **BAML** into key-value pairs. The keys of the key-value pairs are `x:Uid` values that are placed by the developer in the original **XAML**. (Full code available in Appendix B, **Window1.xaml**)

```xml
...<TextBlock Grid.Column="0" TextWrapping="Wrap" HorizontalAlignment="Left"
x:Uid="GuiSpecification[Name=&quot;APPLICATION/AboutSDP3Tool&quot;]/Workspace[Name=&quot;AboutDialog&quot;]/Layout[Name=&quot;Layout1&quot;]/Workspace[Name=&quot;IconAndText&quot;]/Layout[Name=&quot;VersionLayout&quot;]/Workspace[Name=&quot;VersionText&quot;]/Label"
   Text="ba.Version"
   Margin="15,2,0,2">
...```

These `Uid`’s map to a specific GUI-Specification file. In other words, they are mapped in such way, that they satisfy all the properties of `Uid`. These ID’s are often of the form `GuiSpecification[Name=&quot;APPLICATION/AboutSDP3Tool&quot;]/Workspace[Name=&quot;AboutDialog&quot;]/Layout[Name=&quot;Layout1&quot;]/Workspace[Name=&quot;IconAndText&quot;]/Layout[Name=&quot;VersionLayout&quot;]/Workspace[Name=&quot;VersionText&quot;]/Label’. This could be interpreted as a specific location in given xml file. (Sample code taken from **GUIspecification_AboutSDP3Tool_original** in Appendix B)

```xml
<Name> AboutSDP3Tool</Name>
<Workspace>
   ...
   <Layout>
   ...
   <Workspace>
```
The process of assigning theseUid’s, today, is done manually. The output here is a .CSV file which contains temporary text; this can be viewed when opened with any .CSV file compliant editor. Figure 2:2 shows sample .CSV file (for French in France) in Microsoft Excel.

In step 3, Technical Writer receives a mail with new GUI-Specifications and zipped xml file containing renditions for GUI-Specifications. It is then his/her responsibility to write relevant text and this text would be shipped in the final version of the application. However, there is more to this process before product is finalized. The temporary version of text is embedded in zipped xml file which is imported from the versioning system into
In this context, WINGS is a tool that interprets these Specification files and converts them into WINGS adaptable form. (WINGS also perform other functionalities that are not important with respect to my work.) Here the text is not only represented but also edited in xml format. So, technical writer has no idea whether the text belongs to Button or Label or Menu Item or which UI Element. At Scania, usually Technical Writers are non-software developers and are unfamiliar about the tags being used. The xml file is exported from WINGS to an external translator for translation and received back into WINGS. After translation WINGS passes all translated files (.xml) to Production Tool along with their GUI Specifications.

In step 4, Production Tool, custom tool at Scania, creates database files (.mdb). No change is done to xml files and their specifications at this step.

In step 5, Localization Tool, another custom tool at Scania, is used for merging the translated values into locale specific files. Locale specific files are often referred to as language files (Sample Swedish lang_sv-SE.xml is included in Appendix B). Here the ID tag corresponds to actual value of the text by mapping to respective contents.

```
<Id>GuiSpecification[Name="APPLICATION/AboutSDP3Tool"]/Workspace[Name="AboutDialog" ]/Layout[Name="Layout1"]/Workspace[Name="IconAndText"]/Layout[Name="VersionLayout"]/Workspace[Name="VersionText"]/Label</Id>
```

The tool edits key-value pairs, so as to have same key with locale language value or translated version of text. The output of this step is number of .CSV files.

In step 6, LocBaml is again used to generate new localized satellite assemblies. For example, LocBaml.exe /generate filename.resources.dll /trans: filename.resources.dll.CSV /out: /cul:de-DE generates German in Germany resource files. This is last step in localization. During final steps all satellite assemblies are committed to the versioning system.

Some disadvantages of existing workflow are:

1. Each newly formed text has GUI-Specifications file attached. Complexity increases with the increase in number of files and their maintenance.
2. Although GUI-Specifications are used internally (within Scania) during the process, they form an important part of the application. More number of resources are added to main assembly, thus to every satellite assembly.
3. Although this process is scalable, scalability depends on the adaptability of developers and technical writers to the system. In other words an inexperienced person cannot do this job. Certainly imposing more requirements on new trainees.
4. Error correction and detection becomes difficult. For example there were few occasions where some text written in WINGS system never appeared in SDP3 application. The text however, appears in WINGS but is missing in the original application’s UI.
5. Testing requirements are not satisfied completely. Questions such as following arise during this process: Is localization testing done for every language? If yes, which type of testing is done, i.e., linguistic, functionality, compatibility and/or regression testing? Is the software tested on all platforms’ language compatibility, intended for support?
6. Resource Fallback hierarchy used during this process is single level. Once a particular resource assembly is loaded resources are cached in a ResourceSet in the application domain, where each ResourceSet represents a single set of resources for a specific culture, equivalent to a single Resx file’s content. If a matching resource for the current culture doesn’t exist – whether it’s because there are no resources at all defined for this culture or whether a resource key is missing - the application falls back to the nearest resource match using a concept called Resource Fallback shown in Figure 2:3. [3] Resource Fallback searches for most relevant resources down the culture hierarchy which means that cultures are searched from most specific to least specific. For example, if an application is compiled with resources in US English (en-US) and the application is executed with a German UICulture (de-DE) the fallback hierarchy looks like this: Firstly, Specific Culture (de-DE) -> secondly, Neutral Culture (de) -> thirdly, Default or Neutral Culture (en-US).

![Figure 2:3](image)

7. Change of language on-the-fly not possible. WPF loads entire BAML document as one entity without looking for individual values. BAML resources are running from outside satellite assembly, it’s same for default culture. Hence to change language using an option on UI will require the application to restart.

8. Use of non-standard tools like LocBaml. The LocBaml tool is not a production-ready application. It is presented as a sample that uses some of the localization APIs and is just an example to help one implement their own localization tool.

9. Non-friendly view is provided to technical writers. Technical writers are the main source for text during entire process and their knowledge is limited by the view
provided within WINGS. Hence, transparency with respect to application’s context is reduced. This may further result in more meetings and future discussions.

10. GUI specifications have to be mapped initially with temporary text by developers. The reason for this is the existence of GUI-Specifications and their manual implementations. Although translation is done externally, mapping is manual.

11. Translation results in more resource files. The very fact that the documents are not only converted into xml files to send across to translator but also received back as xml files, creates more files. Each of which is again vital resource for generated satellite assemblies. “Imagine this for every business component”.

12. The need to get these translated files via Production tool. All the necessary files with respect to GUI text come clean from production tool. The question here is why are these files supplied to production tool? This duplicates the above mentioned files again.
2.3 Method

The purpose of this section is to explain how data was collected and how it was analyzed. My research involved in-depth analysis of a process. Hence case study is the research method employed during this project. Case study is usually used in information system research projects. According to Darren Dalcher and Lindsey Brodie [11] data collection methods for case studies include observation, documentation and interviews, however, other tools can be selected to suit the particular requirements of a case. The data obtained by multiple means from multiple perspectives was rich enough to provide a real insight into the main issues during this project. This method of research was useful in exploring novel situations in real-life settings and in covering different perspectives of the same problem. However, this method presents difficulties in controlling variables and introducing potential biases. [11] Interview is a type of meeting used to collect information verbally. Types of interviews are: structured, semi-structured, unstructured, one-to-one, group and focus groups. [11] The interview technique used to collect preliminary information on this project was unstructured interview. This type of technique is often informal in nature and there are no set questions. [11] Figure 2:1 was the base to conduct interviews. Initially unstructured interview technique helped me understand present localization process at Scania (section 2.2). Observation technique was used to understand the flow of GUI text from each component. Observation can be described as the scrutiny and recording of actions in natural settings. [11] Observation was carried out by mapping step by step to Figure 2:1, starting from developers to configuration management personal. The essential question (“What happens to GUI text in this step?”) asked during the interview process served as a basis to gather sufficient information about each step.

After gathering all the necessary details about each component, documentation technique was employed. Documentation includes both written and non-written documents. Document and record searching was helpful in establishing quantitative information about data and procedures. While following this technique I relied on existing documents which provided solid foundation during my research. Examples of documents are letters, books, journals and organizational documents. Here, I studied a series of articles, starting from Microsoft’s libraries (MSDN) to several articles and blogs on open web. Most of these articles and blogs were written by personal that are/were expert .NET developers.

Qualitative data obtained from interviews and observations opened new directions. However, additional background information proved handy in understanding whole environment and interpreting the context. The information gathered raised questions: Are GUI specifications needed? Can they be replaced (may be by an available localization tool)? The idea here was to construct experiments. Experiment is a process by which existing theories or a new hypothesis is either approved or disapproved. A hypothesis was assumed that removing only GUI specifications from the process (section 2.2) would be sufficient to build new workflow.

First test conducted to remove GUI specifications from present workflow proved the hypothesis correct. This test gave rise to insufficient results in terms of expectations and more questions. It did not provide context for technical writers, did not solve
problem of additional files and did not allow simultaneous development between developers and technical writers. The other question was: are there any other workflows that can be used to replace LocBaml based approach? Further research showed existence of more processes. The argument in section 4.5 compares the behavior of other processes. The discussion in section 4.7 explains the most suitable choice for new workflow.

Although the method and data collection techniques chosen were time consuming and expensive they helped me assess my abilities while testing, listening, seeing and reading.
3. Alternative processes in Localization

Localization of .NET applications can be done in several ways. Although most of these approaches are not approved by Microsoft, my research shows that there are four major approaches that are widely implemented till date. To show how localization is done in each method, brief description of each method is provided in this section. However, technical workflow is explained in Appendix A. These approaches would help answering questions like: What are the different ways of localizing .NET Applications? How to choose correct localization process? The choice depends on resources at hand. So, what are the resources that are required for a particular method of localization? How to match available resources to the ones at hand? What are the best practices to implement a chosen option? How to implement given process or method? [3][4][9]

3.1 Overview of existing processes

3.1.1 Resx Based Approach

The first process is called Resx Based Approach because localizable strings are stored in RESX files. Microsoft Corporation primarily associates RESX files with '.NET'. "The .resx resource file format consists of XML entries, which specify objects and strings inside XML tags."[2f] Some advantages of a .resx file are: it can be written to, parsed and can be manipulated using any text editor (such as Notepad or Microsoft Word). In this approach the localizable strings are stored in external resx files (often located in Properties folder inside project folder) as key-value pairs. The keys defined here are used as identifiers in XAML code, thereby, statically binding these identifiers to their values. Since this file contains everything that is needed to be localized for GUI, localizing this resx file into different locales will yield localized software. For example: the default file is Resources.resx then we can have Resources.sv-SE.resx for Swedish, Resources.en-US.resx for English in United States etc (The file names are used according to best practices). This is the very reason for the approach to be simple and easy to implement. One can make as many number of language resx files as he/she has decided to support in the software. When a project is built every resx file produces binary version of itself. By implementing code behind, each binary version can be accessed independently which can help dynamic language change at runtime. [2f][3][6]

3.1.2 LocBaml Based Approach

The second process is called Simple LocBaml Based Approach because LocBaml tool is used according to Microsoft’s recommendations. In this case the localizable strings are not separated at the beginning rather later during localization process. WPF applications are implemented as any other application in a single language. Now we make it localizable ready: means we are separating the strings from Xaml code. Addition of Uid’s using MSBUILD tool facilitates this. Uid properties are used to track changes between
development and localization. It helps to localize new development changes. Firstly LocBaml is used to parse all the identifiers and their values (UI strings) into a comma separated file. Secondly, these comma separated file is edited for number of languages intended to support. For example: the default is projectname.csv then we can have projectname.en-US.csv, projectname.sv-SE.csv. Lastly, LocBaml is used again to generate satellite assemblies from each comma separated file. Satellite assemblies are resource only files and do not have any application code. Every supported culture has their satellite assembly. Respective code is written in order to access these satellite assemblies. Hence, it evident that this approach compiles the binary files twice. [2b][3][6][9]

3.1.3 LocBaml with Resource Dictionary Based Approach

The third process is called LocBaml with Resource Dictionary Based Approach because resource dictionaries are used to store localizable strings along with LocBaml tool. A Xaml file containing multiple resources, which is shared between multiple projects is called Resource Dictionary. (Sample Resource Dictionary MyApplicationResources.xaml is available in Appendix B.) Every element in Xaml includes Resources property which stores collection of resources called dictionary. Resource collection can hold any type of object which is indexed by a string. This has few advantages: increases efficiency by defining the object once and using it several times, increases maintainability by allowing one to move low-level formatting to a central location and increases adaptability by allowing one to modify resources independently. This approach is certainly better than the second approach when it comes to resource handling. This approach also extends to static resource handling, dynamic resource handling, non-shared resources, application resources and system resources. However, localization process is same as that of Simple LocBaml Based approach. [2b][3][6][9]

3.1.4 Localization by Linking Localized BAML Streams with Resgen Compiled String Resources

Fourth process arises from first two approaches mentioned above. What if one wants to use both capabilities of RESX files and LocBaml tool? This approach not only increases number of steps but also increases complexity of any application. This approach is called Localization by Linking Localized Baml Streams with Resgen Compiled String Resources. In this process, WPF application is developed as any other application in one language and is subjected to MSBUILD and LocBaml tools to add Uid’s and parse the identifiers into separate file. These are then arranged into key-value pairs in respective resx files. The compiled files or the binary files are linked to newly formed resx files, thereby, linking localized BAML streams with Compiled String Resources. When project is built, the XAML is converted to the BAML (Binary Application Markup Language). Similar to first approach necessary code is implemented to access them. [3][6]

3.1.5 Localization by Proprietary Tools

Computer software which is owned under exclusive legal right by the producer is called Proprietary software. The purchaser, or licensee, abides to the given, right to use terms and conditions, by exercising restricted access, such as no modification, no further
distribution, or reverse engineering. In this report I have considered six proprietary tools for localization, namely: Alchemy Catalyst, Visual Localize, Lingobit Localizer Enterprise, Multilizer Enterprise 2010, RC-WinTrans 9 and Sisulizer Enterprise Edition. There are few benefits of using these tools: they reduce overall translation costs, they decrease time taken for each release cycle, they help achieve, return of investments at a faster pace and they can produce higher quality of translations. [3][5][7][8]

An application is made Localization Ready by either adding UIDs or by adding custom classes or by adding namespace for Resx files in XAML code. Once application is ready for localization, any one of the tools (Alchemy Catalyst, Visual Localize, Lingobit Localizer Enterprise, Multilizer Enterprise 2010, RC-WinTrans 9 and Sisulizer Enterprise Edition) can be used to generate localized application. Build process constitutes of addition of UIDs or Resx resources and generating initial satellite assembly. Localization process includes translation and generates locale satellite assemblies. The entire process for localization using the above mentioned tools is described in detail on their respective websites. Plenty of online material as well as helpdesk support are also available.

“Nothing in this world is perfect!” Most of these software’s are still under development phase and more research is to be done so as to serve the needs of both translators and developers. Today they mostly serve much functionality for translators and are used seldom by developers. One such example is the lack of binding ability. These tools have no functionality which would bind the translatable strings or text to their respective ID’s. Hence they have to be used along with LocBaml or Resx bindings.
4. Experiments

4.1 Purpose of Testing

The most important purpose of conducting the following experiments is to remove GUI specifications from existing development process. GUI specifications are the entities that are imported from previous version of SDP3. They are ordinary XML (eXtensible Markup Language) files that contain ID’s for mapping UI strings. These are the files that are manually created by developers at Scania. These files have been included in Appendix B.

Another purpose was to witness the compatibility of proprietary tools with entire localization process. If their compatibility was found limited with respect to entire process, then each tool is tested with other processes so as to fit them into GUI text localization process in SDP3. So, each tool is tested with both LocBaml tool and Resx files. During testing I used different locales as default so as to demonstrate that default locale can be any language.

4.2 Prerequisites for Testing

Microsoft Visual Studio 2008 version 9.0.30729.1 SP, editor over Microsoft .NET Framework Version 3.5 SP1 was used for the testing. LocBaml Sample was downloaded from [2b] and was built as per Microsoft’s recommendations and guidelines. MSBUILD tool is available under .NET framework. Developing platform was Microsoft Windows XP Professional Version 2002 with Service Pack 3. For the purpose of this report I have considered six localization tools, namely: Alchemy Catalyst version 8.0 SP1 Build 8108, Visual Localize version 7.5.2.6, Lingobit Localizer Enterprise version 6.0.5988 (Build on Apr 1 2010), Multilizer Enterprise 2010 version 7.5.1.1200, RC-WinTrans 9 version 9.0.1.6 (Build March 2010) and Sisulizer Enterprise Edition version 2008 (Build 303).

4.3 Plan

Here a piece of code [in Appendix B] is considered (part of SDP3 application). This code consists of basic Xaml document and two resources: Resource Dictionary and Scania logo. The sample code has window which embeds Scania logo, version, validity date and copy write text of Scania. The Resource Dictionary MyApplicationResources.xaml contains two string resources for a button in main Xaml. Sample code was modified in four different ways so as to fit the different processes explained in chapter 4 respectively. The default culture used in all the approaches was sv-SE (Swedish in Sweden). The locale’s that were translated to, are en-US (English in United States), Finnish (fi) and Danish (da). The strings in each of these locales were written in English with prefixes of their culture. For example: OK in original Swedish became en-OK in English, fi-OK in Finnish and da-OK in Danish. The main focus of these tests was to fully remove GUI-Specifications from Xaml code.
4.4 Tests

First test relates to Resx based approach, I started by removing existing Uid’s from Window1.xaml, then created Resx file to store key-value pairs. Updating Window1.xaml with relevant keys and building project resulted in default culture satellite assembly file. Now by copying and pasting the same Resx file couple of times, I created duplicate copies of key-value pairs in Swedish. The next step was to rename these files according to best practice locales. By implementing code required in Window1.xaml.cs, I was able to successfully map these Resx files for different locales. Although I did not implement the mechanism to change the code dynamically, I tested by changing the code for different locale’s. For technical details, see Appendix A. [2f] [3] [6]

Second test relates to LocBaml based approach, I started by removing existing Uid’s from Window1.xaml and also added Default Culture in main project file. Updating Window1.xaml with relevant text the application was restored to basic form. Building the project at this stage helps to check if everything in the application still works. This generates main assembly and satellite assembly for default culture. By running MSBUILD and LocBaml tools on entire project, I added Uid’s and parsed these key-value pairs into CSV file. Now using Notepad I modified these values into culture specific locale CSV files. After this it was time to run the last step in localization and implement code in Window1.xaml.cs to access the same. For technical details, see Appendix A. [2b] [3] [6] [9]

Third approach is similar to the present process i.e. LocBaml with Resource Dictionary. While testing this process, GUI specifications were successfully replaced from Window1.xaml and MyApplicationResources.xaml. Rest of the process is similar to the above mentioned process. Microsoft’s recommendations were followed for this process. For technical details, see Appendix A. Most of the pros and cons are same as above method. I would consider this to be a variant of the above approach. [2b] [3] [6] [9]

Fourth test, Localization by Linking Localized BAML Streams with Resgen Compiled String Resources, was never implemented because of the nature of complexity. This process is toughest to implement as it needs some understanding to how localizable entities are handled by compiler. This process may not be feasible for any application of the size of SDP3. For technical details, see Appendix A. This is another variant which combines both Resx based approach and LocBaml based approach. [3][6]

Further tests were conducted using three tools: Alchemy Catalyst, Visual Localize and Sisulizer Enterprise Edition. Each of these tools is tested against each method mentioned above. All the tools support visual localization of WPF applications. The three tools accept .exe (Binary file) and .dll (Resource Satellite assembly) while Alchemy Catalyst also accepts folders (entire folder structure including subfolders) and Sisulizer Enterprise Edition auto-detects project and solution files of Visual Studio. Once application is built, any tool can be used to import necessary file/files. These are then translated into different locales as per instructions available on the respective websites. The tools have extensible support for XML files, Translation Memory files, glossary files (often text based) and more. The main objectives of testing these tools were: ease of usage, support for visual localization and compatibility with WPF Resource Dictionary. Presently, there are no tools that support
visualization for Resource Dictionaries. Although all tools were easy to use Alchemy Catalyst not only provided better visual results but also have more features in terms of translation memory, flexibility and ability to work closely with .Net applications. [3][5][7][8]

4.5 Differences

<table>
<thead>
<tr>
<th>Features/Techniques</th>
<th>Resx Based</th>
<th>LocBaml Based</th>
<th>Tool Based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design-time Support</strong></td>
<td>No design-time support available</td>
<td>Design-time support depends on operating system locale.</td>
<td>Design-time support available.</td>
</tr>
<tr>
<td><strong>Additional Bindings</strong></td>
<td>Bindings done</td>
<td>Bindings not necessary</td>
<td>Not Applicable</td>
</tr>
<tr>
<td><strong>Type of Binding</strong></td>
<td>Static Binding</td>
<td>Often Static Binding, use of Resource Dictionaries may be dynamic.</td>
<td>Depending on user requirements bindings may be static or may be dynamic.</td>
</tr>
<tr>
<td><strong>Addition of ID’s</strong></td>
<td>Manual</td>
<td>Can be manual or automated.</td>
<td>Not Applicable</td>
</tr>
<tr>
<td><strong>Mapping of Resources</strong></td>
<td>Resources in Resx files are mapped manually because of, lack of design-time support.</td>
<td>Resources in Xaml files are mapped to their properties during development of the product/component.</td>
<td>Can use any of Resx or LocBaml approach.</td>
</tr>
<tr>
<td><strong>Need for Extra Tools</strong></td>
<td>No need for extra tools</td>
<td>MSBUILD, LocBaml, Editor for CSV files</td>
<td>Depending on requirements additional tools may or may not be used.</td>
</tr>
<tr>
<td><strong>Type of Resource Files</strong></td>
<td>All resources are embedded in RESX files.</td>
<td>All resources are embedded in XAML files.</td>
<td>Resources can be embedded in any file supporting standard formats.</td>
</tr>
<tr>
<td><strong>Need for Additional Files</strong></td>
<td>No need for any additional files and/or editors.</td>
<td>CSV files are created and further used to generate satellite assemblies.</td>
<td>No need for any files because the output is satellite assembly.</td>
</tr>
<tr>
<td><strong>Change of Language on- the-fly</strong></td>
<td>Possible</td>
<td>Not Possible</td>
<td>Possible</td>
</tr>
<tr>
<td><strong>Ease of usage</strong></td>
<td>Easy</td>
<td>Difficult</td>
<td>Easy</td>
</tr>
</tbody>
</table>

Tabel 4-1

26
The term design for support in Table 4-1 means the ability to view application under development. Modern technologies have this feature embedded; especially .NET relishes this feature and renders runtime view of the application under development. This indeed helps developers in visualizing the final picture of the application. In order to translate strings to their values in different locales, binding these objects to some ID’s could help. Additional binding in Table 4-1 mean bindings done to resource’s dependency property. Resx based approach does not allow us to set values to non-dependency properties. Usually values in Resx based approach are static means strongly bound to Resx resources. Resx based approach does not provide any support for type conversion (conversion among different data types). For example: usually all dependency properties accept string values, however, there are those which do not and this requires type conversion. On the other hand LocBaml is not supported by Microsoft and is just a sample application. Addition of ID’s can be automated using basic post build scripting. LocBaml implements one-way export; it creates new resource files whenever it exports and one has to manually keep track of all updates to map translated CSV files. One way export detaches change of language on-the-fly ability from the application. These CSV files also contain extraneous amount of layout information. LocBaml with Resource Dictionary based approach is a variant of LocBaml approach. Here Resource Dictionaries are combined with DynamicResource and LocBaml. LocBaml Based approach and LocBaml with Resource Dictionary are difficult processes to implement and manage because of additional CSV files. These files bring extra tools (other editor) into workflow and thus make mapping of resources tedious. Localization process can be accomplished by linking both compiled strings and BAML resources. This not only increases overall complexity of the application but also shares few merits and demerits of both approaches. Since the mechanism to make .NET application, Localization Ready (the ability to add bindings or ID’s to resources), is not a part of any proprietary tool, they cannot substitute the entire localization process. Hence these tools have to be used in conjunction with either Resx files or LocBaml tool.

4.6 Result

Early decision about localization plays a vital role in deciding the right development process. Although existing tools make localization easy, flow and storage of files in different formats make it heavy. [4] In my view present process for localization before building a project need not to be altered. Further usage of synchronization tool should ease the process of localizations at Scania. This tool would behave as a substitute for GUI specifications. The component diagram for the resultant process may look as shown in Figure 4:1.
During steps 1 and 2, WPF application is developed like any other application for localization. In step 3, LocBaml tool is used to add Uid’s to the application. This could be automated using a script or a batch file. In step 4 any tool of choice could be used to read the final binary or .EXE file. The three tools that have been tested for the purpose of my research include functionalities, such as: they can be used to export original Swedish files to translators, they can be used to import the translated files from translators, they can merge translated files to check if any translation is missing for relevant Swedish text, if any translation is missing then original files can be sent again for translation, they have in-built versioning system with which they can check for updates and they can merge these translated files into the application by creating necessary satellite assemblies at specified location. At step 5 these satellite assemblies are committed into main versioning system as a whole application. Since most of the testing for linguistic correctness is done at step 4, further functional testing can be done after step 5.

The role of technical writer in this scenario would be concentrated around the tool. This tool provides context for writing text in the form of GUI representation of actual application and helps technical writers via translation memory, glossaries, spell checking ability etc.

4.7 Discussion

Although LocBaml and Resx based approaches provide complete solutions for localization process, they are heavy processes from Scania’s point of view. Localization of GUI text is a process which involves implementation of GUI, separation of GUI text from GUI, translation of GUI text, merging this text with the application and testing for usability.
This process is again divided into sub-components which are dealt by different departments at Scania. Developers are responsible for implementing GUI and Technical writers for providing relevant text to GUI. In order to satisfy both developers and technical writers, I plotted an idea that is discussed in Result section (previous section 4.6).

Here I chose to use LocBaml tool instead of Resx file because adding Locbaml as a script at the end of the build process is easier than making a Resx file and managing key-value pairs manually in Resx files.

Including a proprietary tool in localization process provides necessary context for technical writers and translators. Since translation process is outsourced at Scania, this tool makes managing different translations from various translators easy. It also helps in merging these translations and versioning them. It adds more features by including: translation memory; reduces overall translation cost, glossaries; for spell checking, What You See Is What You Get editors; visual localization, change of language on-the-fly, management of different versions of translations and automation of localization process.

Development of such a tool is surely possible within Scania but the question here is; is it really necessary? Scania specializes in heavy machine manufacturing and not in localization software development. And why would Scania hire a third party Localization Company to develop such software when they are available. Development of localization tool within Scania is deviation from their business goals and an additional burden on developers.

The localization tool however, comes at a price. Since these tools cannot bind localizable text and make application localization ready they have to be used along with LocBaml tool or Resx files. They are proprietary means they are final products of other companies. The need to buy multiple licenses increases maintenance issues.

The reason to choose Alchemy catalyst was purely based on my observation during experiments. This is the only tool that rendered better visual results in terms of visual localization. Alchemy catalyst shares many similar features with Visual Localize and Sisulizer Enterprise Edition. However, I liked the fact that it is implemented on .NET framework. It would be advisable for both SDP3 and localization tool to be on same platform as it will increase future scalability with respect technological changes.
5. Conclusion and Future Work

The basic goals with which I performed tests are: to find easy localization process, to get design time support while localization, to be able to change languages on-the-fly, to automate most of the process using existing localization tool and to make this process independent of other tools used within Scania.

Localization in .NET is done differently at different stages of the application; depending on this I can majorly classify it as post-build process and pre-build process. Post Build process is where localization is done after the build process. Example: LocaBaml Based Approach. Pre Build process is where localization is done before the build process. Example Resx based approach. I can consider LocBaml and Resx Based to be main approaches and others to be just variation of these processes. Proper understanding of localization process can yield too many possibilities. However, it is developer’s choice to select one of the localization processes that would best fit their requirements.

Among six tools that were used for testing three of them are a part of suggestion (Alchemy Catalyst, Visual Localize and Sisulizer Enterprise Edition) as they are compatible with almost all approaches discussed in earlier Chapter (3). Hence, combination of anyone of the tools with any approach will result in new development process. Depending on the choice of method or process chosen by the developer, necessary input and output files to and from the tool may vary. The solution discussed in results section 4.6 is best with respect to all requirements and expectations. The process may vary with change in requirements and availability of resources.

SDP3 application not only contains GUI text but also database driven content. Hence localization of all the text within SDP3 is mandatory. Suggested solutions can also be mapped for database driven content and more.
Appendix A

The methods have to be explained in technical terms so as to have full and proper understanding of their working. In this section the technical details are enumerated step by step. This does not mean that this is the only way to do things. However, recommended.

A common step during localization is: Creation of the application, either by the use of Visual studio, Expression Blend or command line tools.

1: Localization without Using LocBaml.exe
Step 1) is common.
Step 2) Make and add resource strings for XAML resources.
  • If the application is created in Visual Studio 2008 (VS08), then Resources.resx file is already available under project’s Properties folder.
  • By adding strings to this file, new references and their values are concatenated together.

![Image of Visual Studio Resource Management]

Resources.resx file for WPF application localization.
• Upon addition of strings, change the Access Modifier Property to Public. This produces a public class along with public string property which converts references to portable objects.

Step 3) Edit the UI in XAML.
• By replacing text with static reference to refer string that was used in the previous step.

Consider we had a button in the sample application, now the code will look like:

```xml
<Window x:Class="WPF_Localize.Window1"
 xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
 xmlns:properties="clr-namespace:WPF_Localize.Properties">
 <Grid>
   <Button Margin="30,20,35,26" Name="button1">
     "{x:Static properties:Resources.MainButtonText}"
   </Button>
 </Grid>
</Window>
```

• An addition of xmlns identifier is also noticeable. This point to Resources.resx in Properties folder of the project.

Step 4) Performing localization.
• Create a new Resource.de-DE.resx (de-DE means German in Germany) file in the same location as Resource.resx
• Open this file to edit.
• Enter same reference as Resource.resx and a value which is equivalent to German in Germany and save the file.
• Upon compilation/recompilation, this produces an empty Resource.de-DE.Designer.cs and corresponding DLL. As the code behind is required only once, no duplication is performed.

Step 5) The addition of Culture Property will generate locale specific output. According to Microsoft, best place to do this would be Application’s constructor.

If we say that our project is WPF_Localize then code would look like:

```csharp
{
    public App()
    {
        WPF_Localize.Properties.Resources.Culture = new CultureInfo("de-DE");
    }
}
```

Step 6) Building the project will yield German version of the application. [2f][3][6]
Optional: If we have the click function active for the button then we create another reference in the .resx files and access it like this:

```csharp
private void OnButtonClick(object sender, RoutedEventArgs e)
{
    MessageBox.Show(Properties.Resources.MessageBoxText);
}
```
2: Localization with LocBaml tool

Step 1) is common.

Step 2) By appending “<UICulture>en-US</UICulture>” tag to project_name.csproj file, development language for the project is set to English in United States.

Step 3) By using `MSBuild /t:updateuid project_name.csproj` and `MSBuild /t:checkuid project_name.csproj` from command line, Uids are not only added to the XAML but also gets verified for duplicate values. In this case Uid are unique reference values that keep track of changes to files and identify items for translation.
Step 4) A neutral language resource satellite assembly is created by compiling the project.

Step 5) If LocBaml Tool does not exists then we download the LocBaml tool sample from MSDN site and compile it to generate LocBaml.exe. This has following options:

- **parse or -p**: Parses Baml, resources, or DLL files to generate a .csv or .txt file.
- **generate or -g**: Generates a localized binary file by using a translated file.
- **out or -o [filedirectory]**: Output file name.
- **culture or -cul [culture]**: Locale of output assemblies.
- **translation or -trans [translation.csv]**: Translated or localized file.
- **asmpath or -asmpath [filedirectory]**: If your XAML code contains custom controls, you must supply the asmpath to the custom control assembly.
- **nologo**: Displays no logo or copyright information.
- **verbose**: Displays verbose mode information.

Step 6) Using LocBaml tool is quite simple as it is command line tool and most of the information needed to run this tool is provided on Microsoft’s Library MSDN.

- The first step in here is to parse satellite assembly into .CSV file.
- Output CSV file consists of several fields (their complete description is available in MSDN). By using any tool one can change the content of the field which contains UI values to locale specific language.
- Save the edited file with locale identifier (fr-FR.CSV) (for French in France) and then use this file to build new .resource.dll files.

Step 7) In order to view the new locale specific application change the language from Keyboard options within control panel. [2b][3][6][9]

### 3: Localization with a XAML ResourceDictionary and LocBaml

After Step 1) and Step 2) as shown in the above method, in Step 3) NeutralResourcesLanguage is set by un-commenting the following line in AssemblyInfo.CS which provides Ultimate Resource Fallback Language.

```csharp
```

Step 4) Next step is to add Resource Dictionary to the project. A sample would look like this:

```xml
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="StringResources.xaml" />
    </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Step 5) Merging this into an application a sample would be:

```xml
<Application x:Class="StringLocalizationSample.App"
    StartupUri="Window1.xaml">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="StringResources.xaml" />  
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
Step 6) The build process from this point is similar to build process from Step 3) to Step 6) in second approach.

Step 7) To view the localized application CurrentUICulture is added to the code.

```csharp
public App()
{
    string culture = "fr-FR";
    // culture = "en-US";

}
```

Now if there is any change in the code, for example:

```xml
<Window x:Uid="Window_1" x:Class="WPF_Localize.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="WPF_Localize" Height="154" Width="194">
    <Grid x:Uid="Grid_1">
        <Button x:Uid="Button_1"
            Margin="30,20,35,26"
            Name="button1"
            Content="{DynamicResource buttonText}"
            Click="OnButtonClicked">
        </Button>
    </Grid>
</Window>
```

Code behind would be:

```csharp
// add this to the Window1.xaml.cs, below the Window1() // constructor
```
private void OnButtonClick(object sender, RoutedEventArgs e)
{
    string localizedMessage = (string)Application.
        Current.FindResource("messageText");
    MessageBox.Show(localizedMessage);
}

Now localization process must be repeated, in order to reflect the changes. [2b][3][6][9]

4: Localization by Linking Localized Baml Streams With Resgen Compiled String Resources

This approach shares many steps from the above approaches. Although all steps are same as the second approach [3][6], there is one change, during Step 6) CSV file is not edited for locale specific languages. Instead a string table is used to store the locale specific UI values. The procedure is as follows:

- Here something similar to first approach is done. Creation of .RESX files for supported languages with reference and string values for each locale.
- The most important aspect of this step is linking and compilation of these files. The batch file from [6]:

```bash
cls
resgen .\Properties\Resources.resx
copy .\Properties\Resources.resources bin\debug\en-US
cd .\bin\Debug\en-US
copy ..\wpf_localize.exe
copy C:\LocBaml.exe
locbaml /generate ..\..\..\obj\debug\WPF_Localize.g.en-US.resources
   /tran:trans.csv /cul:en-US /out:..
ren Resources.resources WPF_Localize.Properties.Resources.en-US.resources
al /template:..\WPF_Localize.exe" /embed:WPF_Localize.g.en-US.resources
   /out:WPF_Localize.resources.dll
cd ..\..
```

Here string resources are compiled first and then copied to the working directory. Copying `LocBaml` tool to the same location facilitates build process. Executing `LocBaml` tool generates en-US.resources (en-US is usually default) file. `Al.exe` tool combines string resources with translated .BAML resources yielding satellite resource DLL. Similarly the batch file for fr-FR may look as follows:

```bash
cls
resgen .\Properties\Resources.fr-FR.resx
copy .\Properties\Resources.fr-FR.resources bin\debug\fr-FR
cd .\bin\Debug\fr-FR
copy ..\wpf_localize.exe
copy C:\LocBaml.exe
locbaml /generate ..\..\..\obj\debug\WPF_Localize.g.en-US.resources
   /tran:trans.csv /cul:fr-FR /out:..
ren Resources.fr-FR.resources WPF_Localize.Properties.Resources.fr-FR.resources
al /template:"..\WPF_Localize.exe" /embed:WPF_Localize.g.fr-FR.resources
   /out:WPF_Localize.resources.dll
cd ..\..
```

Change the locale similar to Step 6) of first approach so as to view translated application.
Appendix B

The code that was a part of my testing is included as a part of my work here. Window1.xaml is the base file which contains all the design implementation for the GUI. Window1.xaml.cs contains the implementation for the GUI, the actions when clicked upon buttons etc. MyApplicationResources.xaml is Resource Dictionary file which contains few resources. GUIspecification_AboutSDP3Tool_original and GUIspecification_AboutSDP3Tool_modified are the GUI specification. First is original file that would be developed by the developers and later is the final version that maps to actual text in different locales. lang_sv-SE is the language file (so called at Scania) which contains actual text in Swedish. GUI specifications map text into this file to fetch them at runtime. Depending on the number of supporting locales, one can witness many language files.

Window1.xaml

```xml
<Window x:Class="AboutSDP3DialogExample.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Name="AboutDialog"
    x:Uid="GuiSpecification[Name="APPLICATION/AboutSDP3Tool"]/Workspace[Name="AboutDialog"]/Label"
    Title="ba.Scania Diagnos &amp; Programmer 3"
    FontFamily="Tahoma"
    FontSize="11"
    Localization.Attributes="FontFamily(Font Readable Modifiable)
    FontSize(Font Readable Modifiable) ToolTip(Text Readable Modifiable)"
    SizeToContent="WidthAndHeight"
    ResizeMode="NoResize"
    ShowInTaskbar="False"
    WindowStartupLocation="CenterOwner"
    MinWidth="256"
    MinHeight="128"
    MaxWidth="640"
    MaxHeight="480"
    Closed="AboutDialog_Closed"
    Loaded="AboutDialog_Load"
    Unloaded="AboutDialog_Unload"
    WindowStyle="SingleBorderWindow"
    SnapsToDevicePixels="True" Width="282">
    <Window.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="MyApplicationResources.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Window.Resources>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
    </Grid>
    <StackPanel Grid.Row="0">
```
38
Window1.xaml.cs

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text; 
using System.Windows; 
using System.Windows.Controls; 
using System.Windows.Data; 
using System.Windows.Documents; 
using System.Windows.Input; 
using System.Windows.Media; 
using System.Windows.Shapes;

namespace AboutSDP3DialogExample 
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }

        // Event handlers for UI events
        private void AboutDialog_Load(object sender, RoutedEventArgs e)
        {
            //Subscribe for external events
        }

        private void AboutDialog_Unload(object sender, RoutedEventArgs e)
        {
        }

        private void AboutOKButton_Click(object sender, RoutedEventArgs e)
        {
            //this.DialogResult = true;
        }

        private void AboutDialog_Closed(object sender, EventArgs e)
        {
            //this.DataContext = null;
        }
    }
}

MyApplicationResources.xaml

<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:s="clr-namespace:System;assembly=mscorlib">
    <s:String
        x:Uid="GuiSpecification[Name=APPLICATION/EndSDP3Tool]/Workspace[Name=EndSDP3ToolDialog]/Label"
        x:Key="EndSDP3ToolHeader">ba.Avsluta SDP3?</s:String>

    <s:String
        x:Uid="GuiSpecification[Name="APPLICATION/EndSDP3Tool"]/Workspace[Name="EndSDP3ToolDialog"]/Label"
        x:Key="EndSDP3ToolInfo">ba.Vill du avsluta programmet eller inte - du får välja?</s:String>
</ResourceDictionary>
<?xml version="1.0" encoding="UTF-8"?>
<GuiSpecification xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Name>AboutSDP3Tool</Name>
  <Workspace>
    <Type>Dialog</Type>
    <Name>AboutDialog</Name>
    <Enabled/>
    <IsLocalizable/>
    <Align/>
    <Palette/>
    <Font/>
    <Margin/>
    <Spacing/>
    <StretchFactor>0</StretchFactor>
    <Label ref="text">Scania Diagnos &amp; Programmer 3</Label>
    <MinSizeX>450</MinSizeX>
    <MinSizeY>50</MinSizeY>
    <MaxSizeX/>
    <MaxSizeY/>
    <Style>WinPanel</Style>
    <Bevel>Plain</Bevel>
    <LineWidth>0</LineWidth>
    <MLineWidth>0</MLineWidth>
    <Layout>
      <Type>BoxLayout</Type>
      <Name>Layout1</Name>
      <Enabled/>
      <IsLocalizable/>
      <Align/>
      <!-- <ResizeMode>FreeResize</ResizeMode> -->
      <Orientation>Vertical</Orientation>
      <Palette/>
      <Font/>
      <Margin/>
      <Spacing/>
      <Workspace>
        <Type>Label</Type>
        <Name>sdp3</Name>
        <Enabled/>
        <IsLocalizable/>
        <Align/>
        <Palette/>
        <Font/>
        <Margin/>
      </Workspace>
      <Spacing/>
      <Workspace>
        <Type>Label</Type>
        <Name>sdp3</Name>
        <Enabled/>
        <IsLocalizable/>
        <Align/>
        <Palette/>
        <Font/>
        <Margin/>
      </Workspace>
      <Spacing/>
    </Layout>
  </Workspace>
  <Workspace>
    <Type>Panel</Type>
    <Name>IconAndText</Name>
    <Enabled/>
    <IsLocalizable/>
    <Align/>
    <Palette/>
    <Font/>
    <Margin>3</Margin>
  </Workspace>
</GuiSpecification>
<Workspace>
<Type>Panel</Type>
<Name>Text15</Name>
<Enabled/>
<IsLocalizable/>
<Align/>
<Palette/>
<Font/>
<Margin>0</Margin>
<Spacing>0</Spacing>
<StretchFactor/>
<MinSizeX/>
<MinSizeY/>
<MaxSizeX/>
<MaxSizeY/>
<Style>Panel</Style>
<Bevel>Plain</Bevel>
<LineWidth>0</LineWidth>
<MLineWidth>0</MLineWidth>
</Workspace>
<Workspace>
<Type>BoxLayout</Type>
<Name>Layout15</Name>
<Enabled/>
<IsLocalizable/>
<Align>Left</Align>
<Orientation>Horizontal</Orientation>
<Palette/>
<Font/>
<Margin>0</Margin>
<Spacing>0</Spacing>
</Workspace>
<Workspace>
<Type>Label</Type>
<Name>ValidUntil</Name>
<Enabled/>
<IsLocalizable/>
<Align/>
<Palette/>
<Font/>
<Margin>0</Margin>
<Spacing>0</Spacing>
<StretchFactor>0</StretchFactor>
<Label ref="text">Ba.Valid until:</Label>
<MinSizeX/>
<MinSizeY/>
</Workspace>
<Workspace>
<Type>Label</Type>
<Name>ValidUntilInfo</Name>
<Enabled/>
<IsLocalizable/>
<Align/>
<Palette/>
<Font/>
<Margin/>
<Spacing/>
<StretchFactor>0</StretchFactor>
<?xml version="1.0" encoding="UTF-8"?>
    <Name>AboutSDP3Tool</Name>
    <Workspace>
        <Type>Dialog</Type>
        <Enabled/>
        <IsLocalizable/>
        <Align/>
        <Palette/>
        <Font/>
        <Margin/>
        <Spacing/>
        <StretchFactor>0</StretchFactor>
        <Label ref="text">GuiSpecification[Name="APPLICATION/AboutSDP3Tool"]/Workspace[Name="AboutDialog"]/Label</Label>
        <MinSizeX>450</MinSizeX>
        <MinSizeY>50</MinSizeY>
        <MaxSizeX/>
        <MaxSizeY/>
        <Style>WinPanel</Style>
        <Bevel>Plain</Bevel>
        <LineWidth>0</LineWidth>
        <MLineWidth>0</MLineWidth>
    </Workspace>
</GuiSpecification>

GUIspecification_AboutSDP3Tool_modified
<Type>BoxLayout</Type>
<Name>Layout1</Name>
<Enabled/>
<IsLocalizable/>
<Align/>
<Orientation>Vertical</Orientation>
<Palette/>
<Font/>
<Margin/>
<Spacing/>
<Workspace>
  <Type>Label</Type>
  <Name>sdp3</Name>
  <Enabled/>
  <IsLocalizable/>
  <Align/>
  <Palette/>
  <Font/>
  <Margin/>
  <Spacing/>
  <StretchFactor>0</StretchFactor>
  <Label ref="text">GuiSpecification[Name="APPLICATION/AboutSDP3Tool"]/Workspace[Name="About Dialog"]/Layout[Name="Layout1"]/Workspace[Name="sdp3"]/Label</Label>
  <TextModifier>WordBreak</TextModifier>
  <MinSizeX>450</MinSizeX>
  <MinSizeY/>
  <MaxSizeX/>
  <MaxSizeY/>
</Workspace>
<Workspace>
  <Type>Panel</Type>
  <Name>IconAndText</Name>
  <Enabled/>
  <IsLocalizable/>
  <Align/>
  <Palette/>
  <Font/>
  <Margin>3</Margin>
  <Spacing>3</Spacing>
  <StretchFactor>0</StretchFactor>
  <MinSizeX/>
  <MinSizeY/>
  <MaxSizeX/>
  <MaxSizeY/>
  <Style>Panel</Style>
  <Bevel>Plain</Bevel>
  <LineWidth>0</LineWidth>
  <MLineWidth>0</MLineWidth>
  <Layout>
    <Type>BoxLayout</Type>
    <Name>VersionLayout</Name>
    <Enabled/>
    <IsLocalizable/>
    <Align/>
    <Orientation>Horizontal</Orientation>
    <Palette/>
    <Font/>
    <Margin>3</Margin>
    <Spacing>3</Spacing>
    <Workspace>
      <Type>Label</Type>
      <Name>VersionText</Name>
      <Enabled/>
      <IsLocalizable/>
      <Align>Left</Align>
      <Palette/>
<Workspace>
    <Type>Label</Type>
    <Name>DummyLabel</Name>
    <Enabled/>
    <IsLocalizable/>
    <Align/>
    <Palette/>
    <Font/>
    <Margin>0</Margin>
    <Spacing>0</Spacing>
    <StretchFactor>10</StretchFactor>
    <TextModifier>WordBreak</TextModifier>
    <MinSizeX>0</MinSizeX>
    <MinSizeY>0</MinSizeY>
    <MaxSizeX/>
    <MaxSizeY/>
</Workspace>

<Workspace>
    <Type>Button</Type>
    <Name>AboutOKButton</Name>
    <Enabled/>
    <IsLocalizable/>
    <Align>HCenter</Align>
    <Palette/>
    <Font/>
    <Margin/>
    <Spacing/>
    <StretchFactor>0</StretchFactor>
    <Label ref="text">GuiSpecification[Name="APPLICATION/AboutSDP3Tool"]/Workspace[Name="AboutDialog"]/Layout[Name="Layout1"]/Workspace[Name="AboutOKButton"]/Label</Label>
    <MinSizeX/>
    <MinSizeY/>
    <MaxSizeX/>
    <MaxSizeY>25</MaxSizeY>
    <Default>true</Default>
    <AutoDefault>true</AutoDefault>
    <FocusPolicy>Tab</FocusPolicy>
</Workspace>
</Layout>
</Workspace>

<InternalConnection>
    <Name>OKButtonToModel</Name>
    <IntSender>
        <GUISpecificationName>AboutSDP3Tool</GUISpecificationName>
        <QtObject>AboutOKButton</QtObject>
        <Signal>clicked()</Signal>
    </IntSender>
    <IntReceiver>
        <GUISpecificationName>AboutSDP3Tool</GUISpecificationName>
        <QtObject>AboutDialog</QtObject>
        <SignalOrSlot>accept()</SignalOrSlot>
    </IntReceiver>
</InternalConnection>

<InternalConnection>
    <Name>SetAboutTextInfo</Name>
    <IntSender>
        <GUISpecificationName>StartApplication</GUISpecificationName>
        <QtObject>ApplicationModel</QtObject>
        <Signal>setAboutText(const QString&amp;)</Signal>
    </IntSender>
    <IntReceiver>
        <GUISpecificationName>AboutSDP3Tool</GUISpecificationName>
        <QtObject>AboutText</QtObject>
        <SignalOrSlot>setText(const QString&amp;)</SignalOrSlot>
    </IntReceiver>
</InternalConnection>
<InternalConnection>
  <Name>SetCopyrightTextInfo</Name>
  <IntSender>
    <GUISpecificationName>StartApplication</GUISpecificationName>
    <QObject>ApplicationModel</QObject>
    <Signal>setCopyrightText(const QString&)</Signal>
  </IntSender>
  <IntReceiver>
    <GUISpecificationName>AboutSDP3Tool</GUISpecificationName>
    <QObject>copyright</QObject>
    <SignalOrSlot>setText(const QString&)</SignalOrSlot>
  </IntReceiver>
</InternalConnection>

<GuiSpecification>

<xml version="1.0" encoding="UTF-8"?>
<Lang xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\pttmp\sssbBg_BasApp_2.4_s\compTmp/BasAppProdInterface/technicalArchitecture/ResourceFramework/schema/Lang.xsd">
  <Name>sv-SE</Name>
  <Text>
    <Id>GuiSpecification[Name="APPLICATION/EndSDP3Tool"]/Workspace[Name="EndSDP3ToolDialog"]/Label</Id>
    <Content><![CDATA[Avsluta SDP3-programmet?]]></Content>
  </Text>
  <Text>
    <Id>GuiSpecification[Name="APPLICATION/EndSDP3Tool"]/Workspace[Name="EndSDP3ToolDialog"]/Header</Id>
    <Content><![CDATA[Scania Diagnos & Programmer 3]]></Content>
  </Text>
  <Text>
    <Id>GuiSpecification[Name="APPLICATION/EndSDP3Tool"]/Workspace[Name="EndSDP3ToolDialog"]/MessageBoxButton[ButtonType="Ok"]/ButtonText</Id>
    <Content><![CDATA[OK]]></Content>
  </Text>
  <Text>
    <Id>GuiSpecification[Name="APPLICATION/EndSDP3Tool"]/Workspace[Name="EndSDP3ToolDialog"]/MessageBoxButton[ButtonType="Cancel"]/ButtonText</Id>
    <Content><![CDATA[Avbryt]]></Content>
  </Text>
</Lang>

</GuiSpecification>
Du har antingen SDP3 Configurator eller SDP3. Det bara att använda ett program.

OK

Scania Diagnos & Programmer 3

Scania Diagnos & Programmer 3
<Text><Id>GuiSpecification[Name="APPLICATION/ExpirationMessage"]/Workspace[Name="ExpirationDialog"]/Layout[Name="Layout1"]/Workspace[Name="ContinueButton"]/Label</Id><Content><![CDATA[OK]]></Content></Text><Text><Id>GuiSpecification[Name="APPLICATION/ExpirationMessage"]/Workspace[Name="ExpirationDialog"]/Layout[Name="Layout1"]/Workspace[Name="ContinueButton"]</Id></Text><Text><Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Label</Id><Content><![CDATA[Scania Diagnos & Programmer 3]]></Content></Text><Text><Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainMenu"]/Item[Name="File"]/Label</Id><Content><![CDATA[Arkiv]]></Content></Text><Text><Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainMenu"]/Item[Name="Settings"]/Label</Id></Text><Text><Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainMenu"]/Item[Name="Show"]/Label</Id></Text>
<Text>
<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainToolBar"]/Label</Id>
<Content><![CDATA[Reparation Toolbar]]></Content>
</Text>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainMenu"]/Item[Name="Help"]/Label</Id>
<Content><![CDATA[Hjälp]]></Content>
</Text>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainToolBar"]/Label</Id>
<Content><![CDATA[Avsluta]]></Content>
</Text>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainToolBar"]/Label</Id>
<Content><![CDATA[Om SDP3]]></Content>
</Text>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainToolBar"]/Label</Id>
<Content><![CDATA[Användarinstruktion]]></Content>
</Text>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="mainwindow"]/Layout[Name="mainlayout"]/Workspace[Name="MainToolBar"]/Label</Id>
<Content><![CDATA[Kommunikationsproblem]]></Content>
</Text>
Information om Ombyggnad (fabriksstödd)

Felrapportering

Release notes

OK

SDP3
Kan inte hitta något program som är associerat med denna fil.

Stödda komplettnummer

Information om Kampanj

Starta demoläge

Avsluta demoläge
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemDemoEnd"] -->
<!-- @editorial true -->
<!-- @editorial-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemDemoEnd"]/Content -->
</Text>
<Text>
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuPrint"]/Content -->
</Text>
<Text>
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuPrint"]/Content -->
</Text>
<Text>
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuSave"]/Content -->
</Text>
<Text>
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuEndJob"]/Content -->
</Text>
<Text>
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuOpenFile"]/Content -->
</Text>
<Text>
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuSnapshot"]/Content -->
</Text>
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuOpenFile"] -->
<!-- @editorial true -->
<!-- @editorial-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuOpenFile"] -->
<!-- @ipm-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuSnapshot"] -->
<!-- @editorial true -->
<!-- @editorial-id GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuSnapshot"]/Content -->
<Content><![CDATA[Demo:]]></Content>
<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemUSB"]</Id>
<Content><![CDATA[MenuItemUSB]]></Content>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemVCI"]</Id>
<Content><![CDATA[VCI2]]></Content>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemDtcInHex"]</Id>
<Content><![CDATA[Visa hexadecimalt]]></Content>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemPermission"]</Id>
<Content><![CDATA[Behörighet]]></Content>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemAddDTC"]</Id>
<Content><![CDATA[Alla felkoder]]></Content>

<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/PresentationText[Name="MenuItemDTCsForVehicle"]</Id>
<Content><![CDATA[Felkoder i fordon]]></Content>
Sök
COO/EMS/LAS
Startspärr utan utökat stöldskydd (startnycklar utan transponder)
Startspärr med utökat stöldskydd (startnycklar med transponder)
LAS
Avsluta
<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="StartupDialog"]/Label</Id><Content><![CDATA[Varning]]></Content></Text>
<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="StartupDialog"]/Layout[Name="Layout1"]/Workspace[Name="IconAndText"]/Layout[Name="Layout2"]/Workspace[Name="Text"]/Layout[Name="Layout3"]/Workspace[Name="Caution1"]/Label</Id>
<Content><![CDATA[Denna programvara (SDP3) kan användas för att ändra inställningar i fordonets elsystem.]]></Content></Text>
<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="StartupDialog"]/Layout[Name="Layout1"]/Workspace[Name="IconAndText"]/Layout[Name="Layout2"]/Workspace[Name="Text"]/Layout[Name="Layout3"]/Workspace[Name="Caution2"]/Label</Id>
<Content><![CDATA[Vissa sådana ändringar, eller kombinationer av inställningar, kan påverka fordonets egenskaper negativt och utan förvarning.]]></Content></Text>
<Id>GuiSpecification[Name="APPLICATION/StartApplication"]/Workspace[Name="StartupDialog"]/Layout[Name="Layout1"]/Workspace[Name="IconAndText"]/Layout[Name="Layout2"]/Workspace[Name="Text"]/Layout[Name="Layout3"]/Workspace[Name="Caution3"]/Label</Id>
<Content><![CDATA[Felaktig användning av programmet innebär således en risk för personskador, materialka skador och brott mot gällande lagstiftning.]]></Content></Text>
Jag har läst och är införstådd med SDP3:s användarinstruktion.

Ja

Nej

OK

Avbryt
Glossary

1) Access Modifier option is available in Resource Designers for resource files that you add to a project and in the default Resource Designer (Resources.resx). This option specifies the access level of the My.Resources helper classes that Visual Studio generates in Resources.Designer.cs or Resources.Designer.vb. My.Resources classes provide runtime access to resources. For Visual C# projects, the access modifier can be Internal, Public, or No code generation.

2) Assembly Linker (Al.exe) generates a file with an assembly manifest from one or more files that are either modules or resource files. A module is a Microsoft intermediate language (MSIL) file that does not have an assembly manifest.

3) BAML (Binary Application Markup Language): When project is built, the XAML is converted to the BAML. BAML is binary representation of XAML. BAML is embedded as a resource in final DLL or EXE assembly. BAML is tokenized, means lengthier bits of XAML are replaced with shorter tokens which makes it smaller and faster for parsing at runtime.

4) Controller Area Network (CAN) National Instruments controller area network interfaces are ideal for developing a wide variety of applications ranging from hardware-in-the-loop simulation systems to low-cost diagnostic, measurement, and calibration tools.

5) Documentum is document management software that provides management of document content and attributes such as check-in, check-out, workflow, and version management. The Documentum product is a three-tier, client-server system built on top of a relational database. [10]

6) Extensible Application Markup Language (XAML) is a declarative markup language. As applied to the .NET Framework programming model, XAML simplifies creating a UI for a .NET Framework application. You can create visible UI elements in the declarative XAML markup, and then separate the UI definition from the run-time logic by using code-behind files, joined to the markup through partial class definitions. XAML directly represents the instantiation of objects in a specific set of backing types defined in assemblies. This is unlike most other markup languages, which are typically an interpreted language without such a direct tie to a backing type system. XAML enables a workflow where separate parties can work on the UI and the logic of an application, using potentially different tools. When represented as text, XAML files are XML files that generally have the .xaml extension. The files can be encoded by any XML encoding, but encoding as UTF-8 is typical. [2e]

7) GUI Text – Text written by the Technical Writers into the WINGS system.

8) LocBaml tool is not a production-ready application. It is presented as a sample that uses some of the localization APIs and is just an example to help one write their own localization tool. This tool parses *.resources.dll files to extract locallizable strings and merges localized strings to construct locale specific *.resources.dll files.
9) Markup Extensions are used when it is not possible to hard-code the property value. For example, one may want to set a property value to an object that already exists. Or one may want to set a property value dynamically, by binding it to a property in another control.

10) Microsoft Build Engine (MSBuild) is the new build platform for Microsoft and Visual Studio. MSBuild is completely transparent with regards to how it processes and builds software, enabling developers to orchestrate and build products in build lab environments where Visual Studio is not installed. Use MSBuild /t:updateuid and /t:checkuid to add and check Uid properties in your XAML.

11) Proprietary software is computer software licensed under exclusive legal right by product owner. The purchaser, or licensee, is given the right to use the software under certain conditions, but restricted from other uses, such as modification, further distribution, or reverse engineering. [http://en.wikipedia.org/wiki/Proprietary_software visited on 27th July 2010]

12) Resource Designer is a user interface tool that enables you to manage resources (such as strings, images, icons, audio, and other files) for your project. The Resources page of the Project Designer hosts an instance of the Resource Designer that stores and maintains resources in a single location (Resources.resx). Resource Designer can be also used to edit stand-alone .resx files.

13) Resource Dictionary is another XAML document that stores resources required for the project.

14) Resource File Generator (Resgen.exe) converts .txt files and .resx (XML-based resource format) files to common language runtime binary .resources files that can be embedded in a runtime binary executable or compiled into satellite assemblies. Resgen.exe converts .txt files to .resources or .resx files, .resources files to text or .resx files and .resx files to text or .resources files.

15) Resource is any non-executable data that is logically deployed with an application. A resource might be displayed in an application as error messages or as part of the user interface. Resources can contain data in a number of forms, including strings, images, and persisted objects. Storing your data in a resource file allows you to change the data without recompiling your entire application.

16) ResourceManager class looks up culture-specific resources, provides resource fallback when a localized resource does not exist, and supports resource serialization.

17) Resx file: When viewing a .resx file, you can actually see the binary form of an embedded object (a picture for example) when this binary information is a part of the resource manifest. Apart from this binary information, a .resx file is completely readable and maintainable. A .resx file contains a standard set of header information, which describes the format of the resource entries and specifies the versioning information for the XML used to parse the data. The following example shows what a typical set of header statements in a .resx file might look like.

```xml
<?xml version="1.0" encoding="utf-8"?>
<root>
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
```
18) Satellite assemblies, by definition contain only resource files. They do not contain any application code. Create one satellite assembly for each culture that the application supports. Each localized BAML resource is put in a separate satellite assembly. Because the satellite assemblies are not part of the main assembly, you can easily replace or update resources corresponding to a specific culture without replacing the application's main assembly.

19) UI – Application’s User Interface

20) Uid properties are used to track changes between development and localization. It helps to localize new development changes. Manual manipulation of Uid’s is not recommended by Microsoft. Since this is a unique identifier, duplication may result in strange errors and multiple references. Uid properties are needed in order for WPF localization APIs to work correctly.

21) UltimateResourceFallbackLocation specifies the assembly for the ResourceManager class to use to retrieve neutral resources by using the resource fallback process. “The .NET framework automatically chooses which satellite resources assembly to load based on the application’s Thread.CurrentThread.CurrentUICulture. This defaults to the culture of your Windows OS. So if you are using German Windows, the de-DE\MyDialog.resources.dll loads, if you are using English Windows, the en-US\MyDialog.resources.dll loads. You can set the ultimate fallback resource for your application by specifying the NeutralResourcesLanguage in your project’s AssemblyInfo.*” [2a]

22) Windows Presentation Foundation (WPF) is designed to allow anyone to create dynamic, data driven presentation systems. Every part of the system is designed to create objects through property sets that drive behavior. [2i]. Although when a WPF application is created in VS08, VS08 provides some default files but there is not set standard for file structure within WPF application. It is recommended by Microsoft to use human readable structure for future maintenance and modifications.

23) WINGS is tool widely used at Scania for developing SDP3. It has much functionality and is used internally. In the context of my report: this tool reads xml files sent by developers into Documentum. After editing, WINGS is used to set the status of the document and the same document is sent to external translator. After translation, files are received in different languages into WINGS. These received files are further sent to Production Tool.
References

[2] Microsoft’s MSDN Library:

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/