Examensarbete

License Handling Utility – an application for easier host installation and license management in the Sectra PACS environment

av

Håkan Skogstjärna

LITH-IDIA-EX-ING--05/012--SE

2005-08-29
# License Handling Utility – an application for easier host installation and license management in the Sectra PACS environment

**Abstract**

This report discusses various issues regarding the development of a software license handling utility, able to be distributed via the web in a convenient manner. The work was done at one of the world’s leading suppliers of radiology image management systems, Sectra Imtec AB in Linköping. The application should solve some awkward steps when new workstations are to be added to the Sectra system. Instead of manually adding information, the application should collect and present only the absolute minimum of information in order to complete the task. One task was to read and store the hardware ID of the client computer and another task was to connect to a management server and collect information about available licenses and present those to the user. Finally the utility should be able to update the server with the choices of the user. During the project some lessons were learned regarding security and web distribution of active content but apart from that the project went on as expected.

**Keywords**

ActiveX, dynamic link libraries, web deployment
Examensarbete

License Handling Utility – an application for easier host installation and license management in the Sectra PACS environment

av

Håkan Skogstjärna

LITH-IDA-EX-ING--05/012—SE

2005-08-29

Handledare: Andreas Böckert, Henrik Elnegård
Examinator: Nahid Shahmehri
Abstract

This report discusses various issues regarding the development of a software license handling utility, able to be distributed via the web in a convenient manner. The work was done at one of the world’s leading suppliers of radiology image management systems, Sectra Imtec AB in Linköping. The application should solve some awkward steps when new workstations are to be added to the Sectra system. Instead of manually adding information, the application should collect and present only the absolute minimum of information in order to complete the task. One task was to read and store the hardware ID of the client computer and another task was to connect to a management server and collect information about available licenses and present those to the user. Finally the utility should be able to update the server with the choices of the user. During the project some lessons were learned regarding security and web distribution of active content but apart from that the project went on as expected.

Table of figures

Figure 1 Network diagram of the Sectra PACS system.................................................5
Figure 2 Present configuration workflow diagram ........................................................6
Figure 3 Future configuration workflow diagram .........................................................7
Figure 4 Welcome screen of the License wizard..........................................................30
Figure 5 First dialog, select workstation license.........................................................30
Figure 6 Second dialog, select appropriate add-on package........................................31
Figure 7 Third and last dialog, confirm selections and commit changes.......................31
Table of contents

1 Introduction..............................................................................................................1
1.1 Goal of thesis ..................................................................................................1
1.2 Requirements for the license handling utility .................................................1
1.3 Quality assurance ...........................................................................................1
1.4 Thesis completion, delivery and documentation ............................................2
1.5 Time table .......................................................................................................2
  1.5.1 Total time .................................................................................................2
  1.5.2 Waypoints ...............................................................................................2
1.6 Report disposition ............................................................................................3
1.7 Conventions in this report ...............................................................................3
2 Background ..........................................................................................................4
  2.1 Presentation of Sectra Imtec AB, Linköping ..................................................4
  2.1.1 Description of the Sectra PACS system ....................................................4
  2.2 The IDS5™ workstation installation process ...............................................5
  2.2.1 Present configuration workflow .................................................................5
  2.2.2 Future configuration workflow .................................................................7
  2.3 Basic conditions ............................................................................................7
  2.3.1 Information storage ....................................................................................7
  2.3.2 Communication IDS5™ client – WISE™ server ......................................8
  2.3.3 Existing components needed .....................................................................8
3 Design and decisions made ..................................................................................9
  3.1 Overall application design and decisions .......................................................9
  3.2 Overall design broken into pieces ................................................................10
  3.3 Detailed design ..............................................................................................12
  3.4 Error handling ..............................................................................................17
4 Fault tolerance and testing ..................................................................................18
  4.1 Test setup .......................................................................................................18
  4.2 Test cases .......................................................................................................18
5 Evaluation of the final thesis work ....................................................................19
  5.1 License wizard vs. the requirements ...............................................................19
  5.2 Time planning accuracy .................................................................................20
  5.3 Difficulties during the project ........................................................................20
6 Summary ...............................................................................................................24
7 Reference list .......................................................................................................25
Appendix A, An example wizard............................................................................26
Appendix B, Test results..........................................................................................28
Appendix C, License wizard screen dumps............................................................30
1 Introduction

1.1 Goal of thesis

The goal of this thesis is to implement a utility to simplify software license handling in new installations of the Sectra PACS IDS5™ (Picture Archiving and Communication System, Image Display System) workstation software.

1.2 Requirements for the license handling utility

Following here is a list of requirements that the utility shall fulfil. The origin of the idea for this license handling utility is from a workshop held by Sectra, where Sectra partners asked for a more smooth way to install new workstations in the Sectra PACS system. In the next chapter the Sectra PACS and the installation procedure will be described. Today there is no such thing as automation and every step requires human intervention. With notes from the workshop, Sectra PACS product manager in cooperation with Sectra developers put together the list of requirements. The aim is to make the process automated as far as possible and the need for human intervention minimized.

Requirements on the application:
- The utility should be an easy to use wizard-type utility.
- The application should be web based and should therefore work with Microsoft’s Internet Information Services (IIS). The application should also be bundled with the deployment server.
- It should be able to run on standard platform.
  - Client: Windows 2000 Pro and Windows XP
  - Server: Windows 2000 Server
  - It should not require Microsoft .NET Framework.
- Development should be done in C/C++, html, ASP and/or wed (wed is an in-house developed web scripting language).

Requirements of the functionality:
- It should have some basic user authentication.
- It should detect if the client already is connected to a license.
- It should be able to enumerate free licenses.
- It should be able to define the client in the PACS Infrastructure and add the client hardware ID.
- It should contain rules between client type and add-ons. (i.e. 3D package is not available for cl.net)
- It should be able to launch the IDS5™ installation package.

See “Appendix A, An example wizard” for one early example of how the wizard could appear to the user.

1.3 Quality assurance

Sectra software runs through several test iterations and functionality and security are tested thoroughly. The testing is a part of the product development and therefore documented which of course should be the case with all testing. Because this final thesis work should be done in a finite time of approximately 400 hrs this utility is not incorporated in the product line.
If the application in the end fulfills Sectra’s requirements and Sectra decides to deploy it, the utility will probably run through some tests and the utility will be documented in a proper way. The application will be a kind of stand alone component in the Sectra family. The wizard will still be tested for verification of the functionality and security. A test plan is presented later in this report and the results are given in “Appendix B, Test results”.

One drawback when developing an application like the one presented in this final thesis is compliance to existing versions of the parent software. New versions are released with new functionality and different appearance. In this work’s inception the test lab was the primary place where the testing of the application should be done, unfortunately the application does not work with Sectra’s latest release that occupies the lab today, as of May 2005. Therefore the testing had to be done with a temporary machine set-up and performed by the writer.

1.4 Thesis completion, delivery and documentation

The goal of this final thesis work is to implement a fully functional license wizard that meets the posted requirements. The test sessions will reveal if the goal is met and the requirements fulfilled. If this is the case the license wizard will be delivered to Sectra and the thesis work is completed. Otherwise, in case the wizard appears not to be fully functional as intended it will not be delivered to Sectra and the thesis work will be considered, from certain points of view, as a failure.

The documentation of this thesis work includes this report and source code comments. Apart from this there will be no further documentation of the wizard. In case of delivery, Sectra will produce the documentation needed that is compliant to their system.

1.5 Time table

Since the writer has the skill of a beginner in software development the planning and time estimating is somewhat of a running issue. As with most software development it is not possible to foresee the time every part takes, so the schedule presented below estimates the time for the different phases of thesis work.

1.5.1 Total time

The thesis work is at the 10 credits level i.e. it should be done in approximately 10 weeks of full time work, which is 400 hrs. Start date of the work was March 21st. 10 weeks from March 21st is May 27th, plus one additional week to balance all the holidays and the end date will be June 3rd.

1.5.2 Waypoints

The planning is very rough knowing the difficulty in planning a software developing project but here are some waypoints that split the time line in some general phases:

- March 21st – April 1st: Background research, problem definition and thesis description.
- April 2nd – April 8th: Software design.
- April 9th – May 20th: Implementation and problem solving.
- May 21st – June 3rd: Time to sum up the work and finish the documentation.
In parallel to the software development the documentation will get its sheer bit of work because it is easy to forget things that should be in the report. In the end of the third phase the supervisor will review the source code and any errors or faults will be corrected during the last phase of the project.

1.6 Report disposition

Chapter “1 Introduction” is a brief introduction to this final thesis work. The goal is presented along with a list of requirements that the utility shall meet. A preliminary schedule is presented and some other information regarding this report is also found in the introductory chapter.

In chapter “2 Background” the company where this work is done is presented and the system that the utility shall cooperate with is also described. The goal with this final thesis work is also clarified a bit when the current situation is compared to the intended future situation.

Chapter “3 Design and decisions made” contains the discussion behind the different decisions that were made when this utility was developed. Different aspects concerning the application as such and detailed discussions about the source code can be found in this chapter.

The following chapter, “4 Fault tolerance and testing” in conjunction with “Appendix B, Test results” describes various test cases, the idea behind the different cases and how the application reacts when tested.

In chapter “5 Evaluation of the final thesis work” the project is evaluated through different aspects: time consumption, difficulties, lessons learned and so on.

The last chapter is a summary that concludes this final thesis work bringing up in short the pros and cons of the whole project.

1.7 Conventions in this report

Abbreviations and acronyms are written out the first time they appear. After that either the abbreviation or the written out words can be used, it is up to the circumstances in the text.

When C++ source code, command line syntax or other type of code is referenced or written this font and size is used.

Footnotes are used when references are made to sources or to the Reference list.

Quotes are written within quotation marks “” and the font is in italic style.
2 Background

This chapter’s purpose is to inform the reader of the background for this final thesis work. Sectra Imtec AB is presented along with the PACS system in which the license handling utility is intended to work. The current situation, which the utility hopefully will make a lot easier, is explained. The utility will inherit functionality from the existing system and that is also described in this chapter.

2.1 Presentation of Sectra Imtec AB, Linköping

Sectra Imtec AB is a company of the Sectra Corporation\(^1\). The company is located in Linköping near the Linköping University. The business area of Sectra Imtec AB is medical systems with radiology image management systems as core component. The main product of Sectra Imtec AB is the PACS system. The acronym PACS is used of all vendors of radiology image systems. Sectra also offers the Sectra MicroDose Mammography™ system that takes advantage of a new technology that delivers high quality images with a significantly lower radiation dose.

The PACS functionality can be enhanced with add on packages. Two main packages are the Sectra Orthopedic Package™ and the Sectra Osteoporosis Package™.

“With the Sectra orthopedic package, the orthopedic surgeon can perform pre-operative templating for joint replacement surgery in a film free environment.”

“The Sectra osteoporosis package addresses the need of an early and precise assessment through a fully automated BMD (Bone Mineral Density) measurement from a standard CR image of the hand in just few seconds”. \(^2\)

In addition to the above-mentioned packages there exists a line of smaller add on packages with special functionality to be integrated with the different versions of the IDS5™ workstations.

2.1.1 Description of the Sectra PACS system

There are two editions of the Sectra PACS, the Sectra PACS Clinical Edition™ for small and mid-sized enterprises and the Sectra PACS Enterprise Edition™, which is a true modular and scalable mission critical system for the large enterprises. The Clinical Edition can be upgraded to an Enterprise Edition if the need for a mission critical 24/7 solution arises. In short a PACS system saves the images on a server. PACS clients can connect to the server and upload the images. One big benefit of this architecture is that the images are available for several users at the same time. Even the user access to the images is taken care of by the server.

The components of the Sectra PACS are:

- PACS core – Sectra WISE™ (Workflow Information Service Engine) management system and the Sectra ImageServer™ for storage and archiving purposes. WISE™ is the brain in Sectra PACS. It keeps track of all the information in the PACS environment.
  - The ImageServer™ comes in three different versions:
    - ImageServer™/s, a short-term online storage that supports RAID (Redundant Arrays of Inexpensive Disks) solutions
    - ImageServer™/fs, a file system storage, supporting RAID as well as SAN

---

\(^1\) http://www.sectra.se
\(^2\) Quotes from http://www.sectra.se/medical
(Storage Area Network) and HSM (Hierarchical Storage Management). ImageServer™/xd, for storing on an external DICOM (Digital Imaging and Communications in Medicine) archive.

- Workstations – a range of Sectra IDS5™ workstations customized to fit different needs of imaging professionals.
  - IDS5™/dx.net - High-end diagnostic workstation
  - IDS5™/cl.net - Advanced review station for clinicians
  - IDS5™/web - Easy-to-use web-based solution
  - IDS5™/qa.net - Quality assurance station for radiographers
  - IDS5™/home - A diagnostic workstation for home use
  - IDS5™/send - Teleradiology Send-station
  - IDS5™/doc.net - Workstation for paper free radiology
  - IDS5™/mx.net - A dedicated mammography review workstation

- Add-ons – extended usability for the IDS5™ workstations are retrieved through the rich variety of add-on packages.

- Integration – a complete palette of integration tools are offered by the Sectra PACS including DICOM, industry standard protocols and Sectra Medical API™ (Application Programmers Interface).

As shown in Figure 1 all workstations but the IDS5™/web communicates directly with the WISE™ server. Instead the IDS5™/web talks to a web server called IDS5™/net which in turn talks to the WISE™ server. This structure enables the use of firewalls between the workstation and the web server.

**Figure 1 Network diagram of the Sectra PACS system**

### 2.2 The IDS5™ workstation installation process

#### 2.2.1 Present configuration workflow

When installing a new client the IDS5™ software is not distributed from a server. Instead the installation is done on the client host itself. On the client, the installation
package is invoked from the deployment server and the entire configuration is done from the client side.

After the initial installation of the IDS5™ software, the host needs to be defined in the PACS Infrastructure on the WISE™ server. This is done with the web based WISE/tools managing application supplied from the WISE™ server. In WISE/tools, “PACS Infrastructure” is selected. Then the “New…” link is selected under the host menu. In the dialog that appears the hardware ID must be supplied. Usually this is done with a PACS help application called nt_id.exe, but there is still the need to copy and paste the ID into the dialog box.

The next step is to bind the IDS5™ workstation to a software license. In case there is a site license installed on the WISE™ server this is not needed. If there are only hardware licenses, some management steps must be taken. Back in WISE/tools, “Licenses” is selected. The listing includes several different license types describing both workstations and add-on packages. An appropriate license is selected and modified. In case the workstation shall have any extra functionality, licenses for this are also modified. After this the client configuration is finally done.

The state machine\(^3\) in Figure 2 presents this configuration workflow in a more visual way. In the state machine it is assumed that the client host is new in the PACS system and that the IDS5™ software is already installed.

---

\(^3\) Facts about state machines were found at http://www.agilemodeling.com/artifacts

---
2.2.2 Future configuration workflow

The aim with this final thesis is to implement a license handling utility that minimizes the need for human intervention when new hosts are configured and defined in the PACS environment. From the deployment server the technician invokes the license handling utility. When the application is invoked it reads and stores the workstation’s hardware ID. After the presentation screen a list of available IDS5™ workstation licenses is presented. The information in the list is gathered from the WISE™ server. The technician selects a license for the workstation and on the next screen a list of available add-on packages is shown. After selecting packages the last screen is a confirmation screen that presents which choices the user has made. If the user confirms this, the wizard commits the changes to the PACS Infrastructure. This includes adding the host to the PACS Infrastructure if it does not exist there, modifying licenses and terminate. If the computer already was bound to some licenses the wizard selects those when the available licenses are displayed. The user is able to change those selections and when the user confirms the choices in the last step the wizard resets the old licenses and updates the chosen ones.

A state machine is presented in Figure 3 for this configuration workflow.

![Figure 3 Future configuration workflow diagram](image)

2.3 Basic conditions

The basic conditions for the license wizard includes how the information is stored on the WISE™ server, in what way actions are invoked and carried out and which software components that will be used in order to get the required functionality of the license handling utility.

2.3.1 Information storage

All information is stored in a regular database that runs on the WISE™ server. For this thesis work there are two areas of interest in the database. The first area concerns what can be found under the PACS Infrastructure menu in WISE/tools. The information there originates from a table called W_CONFIG. This table holds information about all registered computers and if the computer acts as a workstation or a server. The type of workstation or server is also registered in this table. The second area of interest concerns what is found under the Licenses menu and the information presented there is gathered from the W_LICENSE table. As the name reveals the information concerns licenses. The table holds all kinds of Sectra licenses, both server and workstation of every kind as well as add on packages.
2.3.2 Communication IDS5™ client – WISE™ server
The communication between the IDS5™ workstation and the WISE™ server is carried out via remote procedure calls. On the workstation side the calls originate from a C++ class called wise_client. Every call from the workstation is invoked from such an object and that class holds methods for all functionality. The first action when a communication channel is set up is to invoke a socket connection. Then the wise_client connects to the WISE™ server via that socket. Via the wise_client, calls to the database are made to get lists of data to work with. The calls are of the form:

\[
\text{wise}_\text{client} \to \text{get}_\text{data}(\text{search}_\text{criteria}, \text{list})
\]

The search_criteria determines how wide the search will be, from return all posts to return post equal to a specified variable. The list is a RogueWave® List, see next paragraph for an explanation, and the exact type is specified depending on the get_data() method.

2.3.3 Existing components needed
The most obvious component that will be used is the wise_client with its functionality. In order to get the used methods to work several other components will also be used, for example license handling components and some list types specific to the Sectra PACS to mention a few. One comment about the lists, Sectra has imported a set of classes from Rogue Wave® Software\(^4\) for managing lists, vectors and strings instead of using the standard C++ library equivalences. Sectra lists are derived from the RogueWave® lists so that they can contain Sectra specific items.

Another Sectra component that the license wizard will use is the component able to get the hardware ID as well as the NT Security ID (SID). The hardware ID is read from the network adapter and the SID is retrieved from the guest account of the computer. The hardware ID is a unique identifier bound to the network adapter, it is also known as the MAC (Media Access Control) address. If the network adapter is replaced the computer must be redefined in PACS Infrastructure since the new adapter does not have the same ID as the old adapter. In some cases the SID is used instead as an identifier for the computer. The SID is an identifier for the installation of the Microsoft Windows operative system and is not bound to any hardware of the computer. Which of the IDs that shall be used is up to the WISE™ administrator.

\(^4\) See the reference list
3 Design and decisions made

In this chapter the wizard will be thoroughly examined. First there is a section that runs through the top level design and which decisions that was made concerning the overall functionality. In 3.2 the big picture will be split up and the separate parts of the application will be reviewed. Finally in chapter 3.3 the header files of the wizard with methods and variables will be explained. Also in chapter 3.3 the logic in the methods will be discussed.

3.1 Overall application design and decisions

The Sectra PACS environment is a complicated networked system. The big picture displays a complex view, but broken into pieces all functionality that the wizard needs is already implemented in some way. The intended idea behind this final thesis work was to develop a wizard or guide of some sort to simplify the license management when installing new IDS5™ workstations as described earlier in this report.

Because the wizard should be able to be invoked from a web page it was thought of as being one single complete application with all required functionality built into it and the application should not rely on the client system for any Sectra specific functionality. Dynamic link libraries (dll) and ActiveX objects are objects that can be designed to work this way. Of course, both ActiveX objects and dynamic libraries rely on the operative system for window management and data communication, but they can execute as applications. When the web browser visits a page that has an embedded ActiveX object the object is downloaded in the background and invoked when it has been registered on the client system. Many of Microsoft’s own applications rely on ActiveX technology for their functionality. The most obvious is Internet Explorer. The difference between a dll and an ActiveX object is that the ActiveX object acts like an application. The dll on the other hand exports its functionality for other applications to use. ActiveX objects often have a graphical appearance and the possibility to add a Graphical User Interface (GUI) to a dll was also favourable when the facts were collected. When the decision should be made the ActiveX/dll was the preferred solution.\footnote{Facts about ActiveX and dlls from http://msdn.microsoft.com/library and the referenced book “Programming with Microsoft, Visual C++ .NET”}

Are there other possible solutions to the license handling utility? The answer is probably yes. Active Server Pages (ASP) and Component Object Model (COM) objects could perhaps be used. ASP is developed by Microsoft and therefore guaranteed to run on Microsoft IIS. The COM object is only an interface and acts as a layer between the hardware and the software. It seemed more natural to use an ActiveX object that easily downloads to the local computer than to use ASP and COM. Even though there is a possibility of writing client-side script the idea behind ASP-scripts is that they run on the server. The knowledge about developing dlls at Sectra Imtec was also a beneficial factor when the decision was made.

A third alternative was to develop the utility as an executable file. An executable file is possible to add to a web page. The application can then be downloaded to the client system and the user can run it. This idea was discarded however, because of the number of dialog boxes and warnings that the web browser will display every time that the user will run the application. Compared to the ActiveX alternative that only
displays warnings the first time the user visits the web page the idea with an executable file was dropped.

Another issue when the web solution was considered was to keep the number of Internet Explorer alert dialog boxes to an absolute minimum. On the Internet today there exists lots of malicious code that can download itself from a web page and damage the client computer in many different ways. The default setting in Internet Explorer blocks unsigned code. If the code is signed but the certifier is unknown, Internet Explorer alerts the user with a dialog box. Therefore it was necessary to make the code as secure as possible considered the download and install issue.

After the first weeks of research and collecting facts it became obvious that it should be hard to incorporate all needed functionality into one single component. The existing methods depend on other methods and to implement only the needed functionality would include a lot of overhead in the code. Since a major part of the required functionality already was present in the Sectra PACS environment developing the wizard was somewhat a matter of addressing the right methods and evaluate and take care of the result that those methods can give. The code should then be extended with a GUI that is capable of displaying the results to the user and respond to user input.

Therefore the wizard ended up as being an ActiveX/dll component, embedded in a web page. On the same page there is also a cabinet file. Packed in the cabinet is the wizard dll along with other dlls that the wizard requires for its functionality. The license wizard is in all that matters an ActiveX object but the file extension is “dll”. When the wizard is referred to in this report both of these two conceptions are used.

There is no difference to the user if the wizard comes as a single ActiveX component or bundled with a dozen extra dynamic link libraries. The only noticeable difference is a couple of more seconds of download time. On the other hand the wizard comes bundled with the dynamic link libraries that it is developed and tested with and therefore guaranteed to work with. This way the “dll hell” is avoided and the client system acts only as a container in which the utility can run. The expression “dll hell” originates from the problems that arise when an application is developed with, and depends on a certain set of libraries. When the application later is deployed the client system can have all the required libraries installed but of other versions. Version conflicts like this can result in erroneous behaviour of the application.

### 3.2 Overall design broken into pieces

**Hostname and Ethernet address**

The client computer’s Ethernet address is required in the license wizard. It is used in comparison to stored Ethernet addresses and also if the computer needs to be defined in the PACS Infrastructure. If a computer is to be defined the computer’s hostname is also required. To retrieve the Ethernet address an existing class in the IDS5™ environment is used. This class can retrieve both the Ethernet address as well as the NT Security ID and this is done in the constructor of the wizard class Install_info. If the computer is to be defined in the infrastructure the wizard reads the hostname and updates the infrastructure along with the Ethernet address.
User authentication

There is no special login method implemented in the License wizard. In the constructor of the core class of the wizard there is a call to a method from the wise_client class, “authenticate_user”. This method tries to get the username and password from environment variables. In case the method fails the WISE™ server might prevent the user access to some methods.

License handling

To determine if the client already is bound to some licenses the list of licenses is retrieved from the WISE™ server. The licenses in the list are then checked if they match the clients Ethernet address or SID. If a match is found the license is copied to a local list in the wizard. Unbound licenses are found in a similar way but the licenses’ hardware ID is compared to a “0”. If the license does not belong to the type “IDS”, it is discarded.

Every single license represents one record in the database. All those records are returned when the licenses are collected. Instead of saving all those records in the utility, a variable is used to store the number of free licenses for each type. Those duplicates are found when the wizard stores the license in the local list. Before storing the license it iterates over the internal list to find licenses of the same type. If a duplicate is found the “duplicates” variable is increased in the already stored license and the second license is discarded.

If the license is for an IDS5™ workstation, it is stored in a workstation list and if the license is for an add-on package, it is stored in an add-on list. This difference between the licenses is revealed when the wizard compares the high bit and low bit values to predefined constants in a descending scale. If no match is found when all possible workstations are tested the final test is towards the different add on packages. The high and low bits mentioned above refers to the anatomy of the license key. A license key consists partially of two values that define which functionality the license key shall give the workstation. The high bit value is 14 bits long and the low bit value is 32 bits long. Every bit position is equivalent to some functionality in an IDS5™ workstation. Other parts of the license key determine expiration date and how many workstations the license is valid for.

PACS Infrastructure

The license wizard retrieves information from the PACS Infrastructure with different existing search methods. These methods are designed to get specific data, for example one method gets license information and another is able to get information about which hosts currently are defined in the PACS Infrastructure. With the use of an in parameter the search method constrains the query and the result will therefore be more specific. As mentioned previously all data are saved in a regular database and it is these search methods that are responsible for the SQL (Structured Query Language) queries to this database.

The search methods return the result in a list and depending on which search method that was used the number of items in the list varies. The list of licenses contains one item per license while the list that is returned from the W_CONFIG table only contains one item. To read data from a license the only requirement is to iterate over the list and use a get-method to retrieve the wanted value. It is a bit more complicated to retrieve data from a list with only one item since this item in the case of the host
search contains all registered hosts. To be able to read the different values in the list item, the item must be parsed to a Tagparse structure. After that the data is possible to traverse with calls to the Tagparse structure and the values can be compared to the local values. The Tagparse structure is similar to the XML (eXtended Markup Language) structure with tags and sub tags. Every tag and value are referred to with pointers and it is possible to traverse the structure with calls like: get_next_tag().

To update the PACS Infrastructure the sequence of work is the opposite. The licenses are updated with a method “update license” with a pointer to the license that has to be updated. Again it is a bit more work to update the list of hosts. With the Tagparse object available the logic is to traverse to the right point in the structure and there add the appropriate tags.

**Safe for initialisation and scripting**

An ActiveX object can state that it is safe for initialisation and safe for scripting in order to be accepted by Internet Explorer. This is of course a serious matter and it relies on the developer to confirm that the code really is safe. When Internet Explorer loads an ActiveX object it verifies that the object is marked “safe for initialisation” and if the object is referred by a script on the page IE verifies that it is marked as “safe for scripting”. There are two ways to mark an ActiveX safe in this way. The first is to add entries in the registry that IE will read. Calling certain functions when the ActiveX registers itself in the system can add those required entries in the registry. The second way is to implement an OLE interface called IObjectSafety. With this interface IE can query the ActiveX directly to get its safety state and to request it to change to safe mode. The license wizard implements the first alternative. Code is added in the wizard’s control constructor, licwctrl.cpp and a “helper” class is imported to carry out the registry actions.

**3.3 Detailed design**

In this section the C++ classes and their methods will be explained. Since the code is a property of Sectra Imtec AB only the header files will be examined here. To create the license wizard ActiveX object Microsoft’s Visual Studio .net 2003 was used. When a powerful IDE (Integrated Development Environment) like that is used nearly all code skeletons are created automatically. All essential classes and functionality are built and the remaining part to write is the specific functionality that the ActiveX will have. In this case a GUI based on dialog boxes with list boxes to present available licenses to the user.

The wizard contains four dialog classes where the first dialog is merely a yes and no button dialog embedded in a web page. The other three dialog classes are in order:

- Select an available workstation license dialog box class
- Select zero or more add on packages to bind to the workstation dialog box class
- Confirm your choices and commit dialog box class

The dialog classes contain as little logic as possible and they rely on the under lying class Install_info for data retrieval and storage.

**Class Install_info**

This is the core of the License wizard. Install_info contains all methods for collecting and storing data and for updating the infrastructure.

---

6 See the reference: Signing and marking ActiveX controls
Public functions and variables

Install_info(const char* host, const char* service)
Default constructor that takes the WISE™ hostname and corresponding port. The constructor connects to the WISE™ server, creates the wise client object and retrieves the Ethernet address and the NT Security ID. In the constructor all available licenses are collected. The reason for all this functionality in the constructor is that the information is needed for the utility to work. The IDs are used when search results are compared and when the wizard in the end defines the computer in the PACS Infrastructure if it is not already defined. The available licenses are used in list boxes in the dialog part of the application.

~Install_info()
Default destructor.

denum ids5ws{ dxnet, qanet, clnet, docnet, home, mxnet, send, addon }
This enumeration is used when a license is evaluated. The corresponding ids5ws type is stored in a license_info struct. It distinguishes between the different types of licenses.

collect_num_free_licenses()
This function is responsible for the data retrieval from the WISE™ server. The function only copies data from licenses that either is unbound or bound to the client computer. When a free license is found the function makes a call to resolve_license in order to get further information from the license key. The functions return type is void.

license_info {
   RWCString appl, lictype, check, hw_id, sid, packages
   ids5ws wstype,
   int duplicates, x_pkgs_h, x_pkgs_l
}
This is a struct that is used to store information about the available licenses. At the time of design it seemed to be a little simpler with a struct instead of a class. Perhaps a class would have been a better choice. Implemented as a class some functionality could have been moved from Install_info, like the functions get_licinfo() and the get_pkgs_names(). RWCString is the RogueWave implementation of a C-string.

get_valid_licenses(bool get_ws)
Here is a RogueWaveTemplatePointerSingleLinkedList. The list contains object of type license_info. The parameter determines if the function should return workstation licenses, get_ws == TRUE or if the returned list should be add-on packages list, get_ws == FALSE.

get_users_choice()
This function returns a pointer to the users_choice_list_m.

commit_changes()
The confirm dialog box class calls this function when the user clicks the Finish button in the last dialog of the wizard. First it determines if the computer is defined in the PACS Infrastructure with a call to exists_in_wise(). If it is not the function calls the add_pc_to_wise() function. After those checks the function calls the set_licenses() function. If all steps are carried out without problems the function returns TRUE otherwise FALSE and the user is informed of the result.

get_licinfo(license_info* plio)
This function is called in the dialog classes when the list boxes are initiated. The returned RWCString is a string representation of either the IDS5™ type or the type of add-on package together with the number of available licenses. It is in this function that the enum variable is used in a switch clause.

get_hwid(), get_sid()
These functions return the computers Ethernet address and NT Security ID in form of RWCStrings. The functions are used when the dialog class determines if there are any licenses that shall be highlighted in order to notify the user that they are already bound the current computer.

Private functions and variables:

wise_c_m
The wise_client object is responsible for all of the calls to the WISE™ server. This pointer is used throughout the entire process of licenses management in the wizard.

hwid_m, sid_m
These variables are of type RWCString and used to store the local hosts both identifiers. They are then used to determine which licenses that are bound to the local host.

host_m, service_m
These two variables are used when the wise_client object is created. The type is const char*. When the wizard initiates and tries to read the registry, those variables are used to store the host and port of the installation server.

workstations_list_m, addons_list_m
In these lists the Install_info stores license_info objects. The lists are of type RWTPtrSlist.

users_choice_list
In this list the GUI stores what the user has selected in the dialog boxes. The list is maintained throughout the license management process. The items in the list are of type license_info.

resolve_license(license_info* plio, license_data* k)
Resolve_license() updates the license_info object with information about which workstation or add-on package the license is. To retrieve this information from the bit values of the license key the function uses bitwise AND against predefined constants of the exact values of standard workstation licenses. When a matching value is found the function makes a call to get_pkgs_names(). The last call is to store_license().

store_license(license_info* plio, RWTPtrSlist<license_info>* the_list = NULL)
Store_license() stores the license in the corresponding list, workstations_list_m or addons_list_m. Before the function stores the license the lists are checked for duplicates. The function returns TRUE if the license was stored, otherwise FALSE.

get_pkgs_names(license_info* plio)
This function returns the name of the packages that the license contains. If there is no package name it returns an empty string.

set_licenses()
Commit_changes calls this function. For every license in the users_choice_list_m set_license() tries to update the License part of the WISE™ database. Set_licenses() first releases any licenses currently bound to the computer. Then it iterates over the users_choice_list and updates the licenses in the WISE™ server. If it is successful the function returns TRUE otherwise FALSE. The result is then passed back to commit_changes() function.
set_hwid(RWCString id), set_sid(RWCString id)
Set functions used by the Install_info constructor. Return type void.
exists_in_wise(), add_pc_to_wise()
These functions are used to determine if the computer is defined in the PACS
Infrastructure. It is in these functions that the Tagparse structure is used since the
search result only contains one item. Exists_in_wise() returns TRUE for obvious
reasons and add_pc_to_wise() returns TRUE if the computer was successfully defined
in the PACS Infrastructure. The user is informed of the result.

Class License_wizardApp
This class is responsible for registering the ActiveX object in the registry. No custom
modifications.

Class License_wizard_control
The license_wizard_control class is responsible for setting up a connection to the WISE
server. In the License_wizard_controlFactory() of the class, the implementation of safe
initialisation and safe scripting is found. The constructor reads the registry of the local
host to get the IDSS™ install directory. The constructor also tries to get the hostname
and port, of the WISE™ server that was used for installing the workstation software,
from the registry. After that the constructor creates an Install_info object. A pointer to
this object is then passed on between the subsidiary classes. The Start_dialog class
creates the first graphical object. The Start_dialog object is created in the
license_wizard_control's onCreate() and the pointer to the Install_info object is passed as
a parameter to the Start_dialogs set_iip() function.

Private variables
start_dialog_m
The first graphical object in the GUI is an object of the Start_dialog class. In the
license_wizard_control's onCreate() function the object is created and embedded in the
first Welcome page of the wizard.
install_info_mp
This is a pointer to the Install_info object. The pointer is passed along to the Start_dialog
object via a function of that class. To the other dialog classes the pointer is passed as a
parameter to each constructor of the classes.

Class Start_dialog
This start dialog only contains two buttons, “Exit and close” and “Start wizard”. The
dialog is embedded on a web page that welcomes the user to the wizard. The “Exit
and close” buttons closes the window and the “Start wizard” button displays the first
dialog of the license management sequence the Select_ids5_dialog. When the user
clicks the Start wizard button the Select_ids5_dialog is created and the Install_info
pointer is passed as a parameter to the constructor.

Private variables
install_info_mp
A pointer to the Install_info object.

Class Select_ids5_dialog
This dialog box lists all available IDSS™ workstation licenses. It is mandatory to
select one license before the next step. The default constructor is rewritten to take the
Install_info pointer as a parameter. In the constructor of the class a reference to the workstations_list_m is retrieved along with a reference to the users_choice_list_m. In the function onInitDialog() the list box is filled with the licenses found in the workstations_list_m. If the computer already is bound to one license this one is highlighted in the listbox. The list users_choice_m is used to determine if the user comes back to the dialog from the select add-on dialog. If this is the case the previous chosen IDS5™ license is highlighted. When the user clicks the “Next” button the selected entry is added to the users_choice_list_m. If no entry is selected the wizard halts and prompts the user to chose one license.

**Private variables**

ids5_listbox_m
This variable is connected with the data exchange functionality to the graphical list box object. The data exchange functionality passes data between the variable and the graphical list box. The type is CListBox.

install_info_mp
A pointer to the Install_info object.

workstations_list_m
This RWTPtrSlist is a copy of the workstations_list_m from the Install_info object.

**Class Select_addon_dialog**

This class is very similar to Select_ids5_dialog, the difference is that the class displays available add on packages in the list box. If the computer is bound to an add-on package license this license is highlighted in the list box. The constructor is rewritten here too to take the install_info_mp as a parameter. Like the constructor of Select_ids5_dialog, references to the lists addons_list_m and users_choice_list_m are retrieved. The function onInitDialog() adds entries to the list box of the dialog and like the previous class any earlier made choices are highlighted. When any of the buttons “Next” or “Back” is clicked the selections from the list box are added to the users_choice_list_m.

**Private variables**

addons_listbox_m
See explanation for the ids5_listbox_m.

install_info_mp
A pointer to the Install_info object.

addons_list_m
This is a copy of the addons_list_m from the Install_info object.

**Class Confirm_dialog**

This is the last dialog box of the license wizard. The content if the users_choice_list_m is presented to the user and the user is encouraged to verify that the list is correct. It is possible for the user to go back through the wizard and alter any incorrect choices. When the user clicks the “Finish” button the dialog calls the Install_info function commit_changes(). If all changes are carried out correctly an alert is displayed stating that the update was successful. If instead the “Cancel” button was clicked all data is discarded and the wizard exits to the Welcome page.

**Private variables**

confirm_listbox_m
See explanation for the ids5_listbox_m.

install_info_mp
A pointer to the Install_info object.

users_choice_list_m
This is a copy of the users_choice_list_m from the Install_info object.

3.4 Error handling
Roughly errors can be divided into external and internal. External errors originate from the world outside the license wizard, like network and server services errors. Internal errors originates from the wizard itself, it can be logical mistakes or perhaps an unexpected result from a function. In the following paragraphs some of the areas concerning external and internal errors are discussed.

The networked environment in which the license wizard runs includes a number of situations that can be erroneous. Conceptually the lowest layer, the physical link can go down resulting in no connection between the client computer on which the wizard is run and the WISE™ server. If the network is free from problems, services on the WISE™ server can be stopped. Except from the WISE™ server service there is also the database manager service. Even if they are not likely to stop the wizard must be able to handle a “no response” error from these remote services.

Assume that the surrounding environment is error free the next possible cause for an error is the PACS Infrastructure. There might be no available workstation licenses in the infrastructure. The purpose of the wizard is to bind a computer to a workstation license and this situation appears to the license wizard as an error and must therefore be taken care of.

If the IDS5™ workstation software is not installed on the client computer when the wizard is invoked this will result in an error. The wizard tries to read some Sectra specific values from the registry to use when it connects to the WISE™ server. If those values are absent the wizard cannot connect to the server or create the wise_client object and this prohibits the initialisation of the Install_info object.

Most of the errors are severe and prohibits the license wizard from carrying out the license management. The only minor error that is handled within the wizard is if the user does not select an IDS5™ workstation license in the first dialog. In this case the wizard informs the user that a license must be selected before the user can move on to the next dialog. For all other errors the wizard displays an alert with some informative error text. When the user confirms the alert the wizard closes in a controlled way. No data is saved and no configuration changes are made and the state of the WISE™ server is the same as before the wizard was invoked.
4 Fault tolerance and testing
In this chapter the wizard’s fault tolerance will be checked. Presented is a list of different cases that the wizard should be able to handle. This chapter also contains test cases to see if there are some software errors in the wizard. The testing is not exhaustive and therefore the wizard cannot be claimed as being completely free from bugs. From the test sessions during the development of the wizard, the wizard is known to be fully functional.

4.1 Test setup
Server: Windows XP Professional Version 2002 Service pack 2, Internet Information Services 5.1
Client: Windows 2000 Professional Service Pack 2, Internet Explorer 6.0
The client is new to the WISE™ server and therefore not present in the PACS Infrastructure. Internet Explorer is used as web browser and the default security and scripting settings are used.

4.2 Test cases
The wizard is deployed with a cabinet file containing the dynamic link libraries that the wizard depends upon.

The first set of test cases is written to ensure fault tolerance behaviour when the client computer does not meet the requirements of the wizard. The requirement of the wizard is that prior to running the wizard the Sectra PACS IDS5™ workstation software must be installed and the test cases will check if the wizard deals with this in an appropriate way.

The wizard also depends upon a networked environment with a running WISE™ server. Therefore the second set of test cases focuses on how the wizard deals with an unreliable network and how it handles if the WISE™ server goes down. In these cases the IDS5™ workstation has been installed on the client.

The third case tests how the wizard can handle the absence of free licenses in the WISE™ server.

The test cases are described in “Appendix B, Test results” along with the test results.

Case 1a: The License Wizard is invoked before the IDS5™ software is installed.
Case 1b: The client registry is corrupted and there is no IDS5™ install directory value.
Case 1c: The client registry is corrupted and the name and port of the WISE™ server is absent.

Case 2a: The network goes down right after dll installation.
Case 2b: The WISE™ server goes down after the installation of the dynamic link libraries.
Case 2c: The WISE™ server goes down during the wizard steps.
Case 2d: The WISE™ server goes down right before the user commits.

Case 3: No free IDS5™ licenses in the WISE™ server.
5 Evaluation of the final thesis work

In this chapter the entire process of this final thesis work will be evaluated. First the list of requirements is reviewed. Does the wizard fulfil all requirements or not? Secondly the time planning is placed under the magnifying glass. Did the project meet its deadlines? At last some of the difficulties that arose during the project are discussed.

5.1 License wizard vs. the requirements

As part of the evaluation the License wizard will here be compared to the requirements that were listed in the beginning of this report. Which requirements are fulfilled and which are not. In the cases where the requirement is not fulfilled the reason for that is explained.

The utility should be an easy to use wizard-type utility
The definition of an “easy to use” application varies from user to user. When the GUI was designed, care was taken to make the appearance as clear and clean as possible. The information and data displayed should be unambiguous and the wizard should prevent any faults that a user possibly could do. The writer’s opinion is that the requirement is fulfilled.

The application should be web based and should therefore work with Microsoft’s Internet Information Services. The application should also be bundled with the deployment server
On the deployment server the wizard is published on a web page and downloads as a dynamic link library when the user selects it. The web server used is Microsoft Internet Information Services. The requirement is fulfilled.

It should be able to run on standard platform. Client: Windows 2000 Pro and Windows XP. Server: Windows 2000 Server. It should not require Microsoft .NET Framework
The wizard is tested on both Windows XP and Windows 2000 Professional and no problems were encountered. The requirement is fulfilled.

Development should be done in C/C++, html, ASP and/or wed (wed is an in-house developed web scripting language)
The language used is C++ and for the GUI and ActiveX code Visual C++. The requirement is fulfilled.

It should have some basic authentication
The only authentication in the wizard is when the wise client object makes the call “authenticate user”. No further actions are taken. To raise the security level a step would be to implement a mandatory login to the WISE™ server, however this is not implemented. The requirement is partially fulfilled.

It should detect if the client already is connected to a license
When the wizard collects the licenses it compares the clients Ethernet address with those stored in the licenses in the WISE™ server. If a match is detected the license will be highlighted when the wizard displays available licenses. The requirement is fulfilled.

It should be able to enumerate free licenses
This requirement is somewhat of a cornerstone to the license wizard. If a license is detected as bound to the current machine it is declared as available since the wizard reconnects it when the user commits. The requirement is fulfilled.

**It should be able to define the client in the PACS Infrastructure and add the client hardware ID**

When the wizard is invoked it retrieves the IDs of the computer. They are later used in the code. The wizard searches for the local host’s Ethernet address in the PACS Infrastructure when the user commits the changes. If the wizard does not find the computer it is defined in the infrastructure. The requirement is fulfilled.

**It should contain rules between client type and add-ons. (like 3D package is not available for cl.net)**

This requirement is not fulfilled. From discussions with the supervisor this requirement was marked as not relevant at this time. The user should be aware of which licenses to bind the computer to.

**It should be able to launch the IDS5™ installation package**

This requirement is not fulfilled. The wizard reads the registry of the client computer to get the installation directory of the IDS5™ software. The registry is also used to get the hostname and port of the WISE™ server from where the IDS5™ was installed. Therefore the wizard depends on the IDS5™ workstation to be installed and the requirement is not considered as necessary.

5.2 **Time planning accuracy**

Scheduling a project is not an easy task to do and with no real experience it is even harder. The project was intended to take around 10 weeks/400 hours of full time work. When I first draw the time line I tried to imagine the separate phases of the work. First there was the research and reading existing code phase. Secondly came the design phase. The third phase was bound to be the implementation phase and finally was the gather all loose ends and get it together phase. With these four phases in mind the time line was drawn. The introductory phase got two weeks. The design phase got a single week. The implementation phase was the longest and stretched over six weeks and the last phase got the remaining two weeks of time. The observant reader notices that the sum is eleven weeks instead of ten. Because of the holidays during the spring one additional week was added to the project.

With the project in the rear mirror an evaluation is at hand. The phases tended to reach into each other. When the research was done some different designs were tested. During the design phase the implementation started quietly. The final phase is somewhat divided between code reviewing and report writing. This is natural though since the implementation ended the first day of the last phase and the supervisor should review the code at the end of the implementation phase. The bottom line is that the schedule was followed rather strictly and the waypoints were held within a day or two. This situation is very satisfying of course and strengthens the self-confidence of the project staff.

5.3 **Difficulties during the project**

In this section some of the difficulties of the project will be highlighted. The perfect project that runs smoothly from start to finish does probably not exist. Every project encounters problems or drawbacks at some time so even this one. In the beginning of
the project the first challenge was to overcome the ocean of existing code. It was not possible to understand it all or get an overview of it and therefore the design was a bit tricky. When the supervisor reviewed the code he pointed at a function that already was implemented in the environment. Things like this happen and it is no big deal. Compared to the different courses at the university coming out into real life was like a cruel awakening. At the school the code is prepared with comments describing classes and functions and skeletons for the student to fill with code. In the real life the code is not commented that rigorously and the purpose of the classes is not always that clear. It is much up to the developer to get a grip of the situation and that can be the source of some problems during a beginner’s project.

Except from the usual null pointer exceptions that tend to pop up when the language is C++ another issue that took some time to solve was how to distribute the wizard. Running an ActiveX object on the machine that produced it is not a problem since it is registered locally. The problem was that on a client machine it refused to load and the download log just stated that it was a download failure. After browsing the web for some days the corner stone of the deployment building was that the ActiveX needed to be signed and time stamped. Internet Explorer must ascertain two issues. The first issue is that the control has not been modified since it was signed and second issue is that the claimed author actually is the author\(^7\). To sign a file Microsoft supplies a tool called SignCode. The syntax for signing a file when the company holds a trusted certificate, here named mycredentials.spc, and a private key named myprivatekey.pvk is:

\[
\text{C:}\text{\textbackslash}>\text{SignCode -spc mycredentials.spc -v myprivatekey.pvk your_file.dll}
\]

The command is executed from a prompt in the same directory as the file to be signed. SignCode is also used to stamp the file with a time:

\[
\text{C:}\text{\textbackslash}>\text{SignCode -t http://timestamp.verisign.com/scripts/timstamp.dll -x your_file.dll}
\]

Files that can and should be signed are:
- ActiveX objects “.ocx”
- Cabinet files “.cab”.
- dBase Catalog Quicken IntelliCharge Categorization File “.cat”
- Dynamic link libraries “.dll”
- Executables “.exe”
- Setup information user control “.ctl”

The second corner stone was that an ActiveX needs run time licenses to load on a computer that is not licensed to use the control\(^8\). These run time licenses can be distributed with a license package file that is embedded in the web page. This license package file is created with a license package authoring tool distributed by Microsoft. The routine is simple just start the tool which is a window based application, select which ActiveX objects that needs to be included in the license package. Save the license package and close the tool. In the beginning of the web page that includes the object another object tag is added containing a reference to the license package file.

\(^7\) See reference: ”Signing and Checking code with Authenticode” and “Signing and Marking ActiveX controls”

\(^8\) See reference: ”How to use licensed ActiveX controls in Internet Explorer”
The third and last corner stone in the deployment building (it is a triangular house) was to solve the dependencies. Since the wizard depends on other IDS5™ libraries to work these must be present on the client machine. If the libraries are present the system must be able to find them. Windows searches per default in the environment variable PATH. If the dependant objects are not found there the wizard will not start. One possible, but ugly, solution was to add the IDS5™ install directory to the PATH variable. This solution was however discarded. Another solution was to change the current directory to the IDS5™ install directory when the wizard was started. This did not either fully solve the problem. The final solution was to compress the wizard along with all its dependants bundled together with an INF file in a cabinet. The INF file specifies which files that need to be present or downloaded for the ActiveX to run. Below here is a short explanation of the content of an INF file.

```
[version]
; version signature (same for both NT and Win95) do not remove
signature="$CHICAGO$"
AdvancedINF=2.0

[Add.Code]
mainfile.dll=mainfile
otherfile1.dll=otherfile1
otherfile2.dll=otherfile2

[mainfile]
file-win32-x86=thiscab
clsid= { 44922388-369B-47BD-B73B-3D34A0A806B1}
FileVersion=1,0,0,1
DestDir=11
RegisterServer=Yes
```

The first tag is mandatory and concerns the Windows family. Add.Code specifies which files the main file needs and where in the INF file information about those files are located. The main file is listed first with the dependants following after. IE reads the INF file from top to bottom but installs the dependants in the other order, in this way all dependencies are solved when the main file is ready to be installed. Instead of the keyword “thiscab” an URL can be used if the file can be found somewhere on the Internet. Every ActiveX and dll has a unique identifier called clsid. On the webpage where the object is embedded it is referred to by this clsid. To distinguish between versions the FileVersion is used. If the version does not matter it can be left blank. DestDir specifies where the unpacked content will be installed. The alternatives are 10 for \windir\, 11 for \windir\system32\ and blank for \windir\cache\. RegisterServer tells IE if the file shall be registered on the client system.

To create a cabinet file Microsoft supplies a utility called cabarc. To create a cabinet with the above example INF file the syntax for this command line tool is as follows:

```
C:\>cabarc -s 6144 n mainfile.cab mainfile.dll otherfile1.dll otherfile2.dll install.inf
```

The files are typed in the same order as they appear in the INF file. To leave some space for signing the option –s is used and the numbers refers to how many bytes that will be reserved for this purpose.

---

* See reference: "Creating a CAB file"
When Internet Explorer reads the object tag on the web page it downloads the cabinet. Then IE opens the cabinet and reads the INF file to see if there are any unresolved dependencies in the system. If there is, IE downloads and installs these missing files in the system. The content is registered and the wizard will finally be able to execute on the client machine.

Another difficulty was to reach into the different classes from RogueWave® Software. Especially the template linked lists took some time to get acquainted with. They all have functions for common tasks regarding lists, like inserting items first and last and return the first and the last item of the list. When a list no longer is used the list should be emptied and destroyed with a call to the clearAndDestroy() function of the list. The error that was made in the code regarded assigning one list to another like:

second_list = first_list;

The second list gets copies of all the pointers of the first list, which leads to the situation that both lists refer to the same items. If now the first list is deleted with a call to clearAndDestroy() and then someone tries to read from the second list this leads to a memory violation because the items are already destroyed.
6 Summary

This summary ends this report as well as the final thesis work. It has been stimulating to work at Sectra Imtec and the developers have always been kind in response to the questions from the novice writer. The project was well prepared by the supervisors at Sectra, Andreas Böckert and Henrik Elnegård, with a clear and concise task to solve. Their common opinion was that the task was able to solve in ten weeks and they were proved to be right. The task was to develop a license handling utility that should be able to solve an installation sequence problem when new workstations are added to the existing environment. As described in chapter “2.2.1 Present configuration workflow” and chapter “2.2.2 Future configuration workflow” the process would be much more effective if the utility was successfully implemented.

The utility should be able to be invoked from a web page and the decision was to make an ActiveX object with all required functionality included. The final License wizard has all required functionality but the wizard relies on other dynamic link libraries to get the functionality needed. It became obvious that it was not an easy task to implement the wizard’s entire functionality in one object. Therefore the wizard only has the core functionality and depends on other objects for the remaining parts. A major part of the functionality needed was already implemented in the IDS5™ software and it only required some investigation to find it. The wizard was then extended with specific functionality that used the existing functions in a proper way. To make the wizard complete the last task to solve was to add a user interface to the ActiveX object. With the aid of Visual Studio .net this was solved and the final product fulfils the requirements of the work.

One issue along the way was to understand the concept of the ActiveX object and the precautions that was required in order to be able to distribute the object via Internet. Code must be signed and time stamped to prevent Internet Explorer from alerting the user with warnings. The distribution also demands run time licenses to be available at download time otherwise the object will fail to load on a client computer. The last security issue was to implement the safe for initialisation and safe for scripting functionality. When an ActiveX object initialises Internet Explorer checks if the object is safe by reading registry keys, if IE can not find those keys it alerts the user with a warning that the object has not proclaimed itself as safe for this action. The same procedure is followed if a script on the same web page refers to the object. All security information was found on the web and clear examples showed how to solve the separate issues.

ActiveX technology is really useful when an object with isolated functionality shall be distributed via the web. Unfortunately distribution with ActiveX objects is often considered to be a security risk and therefore great care must be taken when such objects are developed and distributed. Once those tasks are accomplished the distribution and use of an ActiveX object is really convenient.
7 Reference list

“Programming with Microsoft, Visual C++.NET 6th edition (Core reference)”

Various pages from Microsoft’s MSDN library (visited throughout the thesis work, April to June 2005)
http://msdn.microsoft.com/library

Signing and Checking code with Authenticode (first visited at 2005-04-07)

Signing and Marking ActiveX controls (first visited at 2005-05-19)

How to use licensed ActiveX controls in Internet Explorer (visited at 2005-04-08)
http://support.microsoft.com/kb/q159923/

Creating a CAB file (first visited at 2005-05-17)

Sectra Corporation web site
http://www.sectra.se

Agile Modeling Artifact (visited at 2005-04-25)
http://www.agilemodeling.com/artifacts

Rogue Wave® Software (visited throughout the thesis work, April to June 2005)
http://www.roguewave.com

Rogue Wave® Essential Tools Module Reference Guide (visited throughout the thesis work, April to June 2005)
http://www.roguewave.com/support/docs/sourcepro/toolsref/index.html
Appendix A, An example wizard
Andreas Böckert wrote this example wizard and the rest of this appendix is a quote from that working paper.

- The first step is to select type of client to install. At this point the wizard connects to WISE™ and asks for unassigned licenses.

  This wizard will take you through assigning the license key to this client and install the workstation
  Start by selecting the type of client you want to install:
  - IDS5/dx.net (5 licenses available)
  - IDS5/qa.net (2 licenses available)
  - IDS5/cl.net (12 licenses available)

  Step 1 of 3

- The second step is to select what add-ons that should be connected to the client.

  Continue by selecting the add-ons you want to install:
  - IDS5/dx.net (5 licenses available)
    - Scanning option (1 licenses available)
    - 3D package (4 licenses available)
    - Orthopedic package (1 licenses available)

  Step 2 of 3
• The third step is to make the modifications required in PACS Infrastructure and connect the client to the license(s)

When you click finish you will install:
IDS5/dx.net, 3D package and orthopedic package

• When the wizard is complete, the IDS5 installation package is launched from the deployment server.
Appendix B, Test results

Case 1a: The License Wizard is invoked before the IDS5™ software is installed.
On the server: No special modifications.
On the client: The IDS5 workstation software is not installed on the client. The wizard is invoked from the deployment server.
Expected result: The wizard will state that there was a problem reading the registry. When the dialog is confirmed the wizard should terminate and close.
Actual result: After a couple of seconds an error dialog stating that there is a problem reading the registry. When the dialog is confirmed the wizard terminates and closes.

Case 1b: The client registry is corrupted
On the server: No special modifications.
On the client: Registry – the value of the key HKLM/Software/Sectra/Environment/SECTRA is deleted. The wizard tries to read the install directory prior to creating the wise client object.
Expected and actual result: The wizard handles this case in the same way as case 1a.

Case 1c: The client registry is corrupted
On the server: No special modifications.
On the client: Registry – the values of the keys HKLM/Software/Sectra/Environment/WISE_HOST and WISE_PORT are deleted. The wizard tries to create a wise client with NULL values for HOST and PORT.
Expected result: The wizard reads the registry but cannot connect to the WISE™ server and therefore throws an exception. When this is confirmed the wizard closes.
Actual result: The wizard throws an exception stating: “There is no connection to the WISE™ server”. The wizard closes when the dialog is confirmed.

Case 2a: The network goes down right after dll installation.
On the server: No special modifications.
On the client: IDS5™ is installed. The network cable is unplugged during the invocation of the wizard.
Expected result: Depending on the exact time of the failure the wizard will throw an exception. Upon confirmation of the error dialog the wizard will terminate and close.
Actual result:
- If the network goes down before the dll download is finished the page will be displayed with an empty space where the license wizard object should be.
- If the network goes down during the dll initialization the wizard throws one of the errors from the constructor. These are socket related, WISE™ related and user authenticate related. Upon confirmation of the error dialog the wizard closes.
- If the license wizard is loaded it will display available licenses from the WISE™ server because these are fetched upon creation of the license wizard object. Not until the user confirms the choices and commits with the “Finish” button the wizard complains. Error dialog states that there is an internal error with code 02. Code 02 is when the wizard fails to get a successful search result from the WISE™ server. When the dialog is confirmed the license wizard closes.
Case 2b: The WISE™ server goes down after installation of the dll
On the server: The WISE service is stopped after the wizard has been invoked.
On the client: No special modifications.
Expected result: The wizard will throw an exception when it tries to access the WISE server.
Actual result: The wizard throws an exception stating: “There is no connection to the WISE server”. When the dialog is confirmed the wizard closes.

Case 2c: The WISE™ server goes down during the wizard steps
On the server: The WISE™ service is stopped after the wizard has been invoked.
On the client: No special modifications.
Expected result: This failure is equivalent to case 2a and the third result.
Actual result: License wizard internal error, code 02. When the dialog is confirmed the wizard closes.

Case 2d: The WISE™ server goes down right before the user commits.
See case 2c.

Case 3: No free IDS5 license
On the server: No available IDS5™ licenses.
On the client: No special modifications.
Expected result: The wizard will throw an exception stating that there is no free license.
Actual result: When the license wizard is invoked it throws the “There is no free license” exception. When this error dialog is confirmed the wizard terminates and closes.
Appendix C, License wizard screen dumps

This appendix presents the visual experience of the license wizard. Figure 4 displays the first screen that appears to the user. The dialog contains some information about the wizard. The dialog in Figure 5 presents the available licenses that are found in the WISE™ server.

Figure 4 Welcome screen of the License wizard

Figure 5 First dialog, select workstation license
When the user has selected a workstation license in the first dialog, Figure 6 is a screen dump of the next dialog in the chain. Here the different add-on packages are listed. The user must not select a package and is free to move on to the final step. Figure 7 shows the final stage of the License wizard. Here a list that contains what the user has selected in the previous steps is presented. If the user clicks Finish the changes are made to the PACS Infrastructure and the wizard closes. Otherwise the user can go back to the previous dialogs and correct any mistakes.

Figure 6 Second dialog, select appropriate add-on package

Figure 7 Third and last dialog, confirm selections and commit changes